Universidad Autónoma de Madrid
Escuela politécnica superior

PhD in Computer Science

PhD Thesis

REAL-TIME SOFTWARE TECHNOLOGY AND ITS IMPLEMENTATION IN EXPERIMENTAL NEUROSCIENCE CLOSED-LOOP PROTOCOLS

Author:
Rodrigo Vicente Amaducci Szwarc
Advisor:
Pablo Varona Martínez

November 2022

REAL-TIME SOFTWARE TECHNOLOGY AND ITS IMPLEMENTATION IN EXPERIMENTAL NEUROSCIENCE CLOSED-LOOP PROTOCOLS

Author:
Rodrigo Vicente Amaducci Szwarc
Advisor:
Pablo Varona Martínez

Grupo de Neurocomputación Biológica (GNB)
Escuela Politécnica Superior
Universidad Autónoma de Madrid
November 2022

Abstract

Understanding neural dynamics is a complex task since they are highly non-linear, partially observable (we can only look at a few brain components simultaneously) and are always changing and adapting due to the neural interactions and their associated plasticity. The predominant protocol used to study such dynamics, known as the stimulus-response paradigm, consists in delivering some predefined stimuli into the system, recording its response and analyzing it offline afterwards. Therefore, this protocol is rather limited when it comes to comprehending transient neural dynamics, which are constantly receiving feedback from hundreds or thousands of elements.

Closed-loop experimental techniques in neuroscience were created as a way of overcoming these difficulties, allowing to observe and control neural systems with stimuli generated online based on their ongoing activity. Unfortunately, it is not always easy to implement this kind of protocol since they require online or real-time technologies as well as more elaborate experimental designs. For these reasons, many neuroscience laboratories and researchers overlook the use of the closed-loop paradigm.

The goal of this thesis is the development of real-time software technology for the implementation of effective closed-loop protocols in experimental neuroscience. To this effect we have carried out three projects during which we have worked with diverse real-time data acquisition, event detection, stimulation and associated analysis techniques. These technologies have been validated at different description levels of the nervous system.

The first project consisted in the development of RTHybrid, a software application aimed at the construction of hybrid circuits composed of living biological neurons and computer-simulated ones, useful for the study of neural dynamics and the role of specific components in neural circuits. For this purpose, it includes a neuron and synapse model library along with a set of automatic calibration algorithms that ensure a correct interaction among all involved components. Since in many cases this closed-loop interaction requires the compliance of very strict temporal restrictions, this tool has been implemented to run over real-time operating systems. RTHybrid proper functioning was validated with *in vitro* experiments using invertebrate living neurons from a central pattern generator (CPG) circuit.

For the second project we built FLC-Hybrot, a hybrot (hybrid robot) whose locomotion is controlled by the neural activity of a functional living crab CPG. The robot uses the dynamical invariants of the living circuit to organize its locomotion and sends sensory feedback to the neurons, altering their behavior and, therefore, the hybrot coordinated motion achieved from this hybrid closed-loop. The biological and electronic components of the hybrot communicated through an online Bluetooth connection. This is the first time a dynamical principle arising in a functional living circuit is validated to control a robotic device with sensory feedback.

Lastly, the third project carried out as part of this thesis work consisted in the development of CNN-ripple, a real-time detection and intervention system of the electrophysiological events known as sharp wave ripples (SWR) using deep learning techniques. We designed and implemented a convolutional neural network (CNN) capable of identifying these events in both offline and online contexts in signals acquired with different types of probes, as well as in distinct

animals' species. To favor and facilitate the use of this method we added it as a plugin to the Open Ephys platform, one of most extended acquisition software applications for experimental neuroscience. To reduce the detection latencies caused by data transmission mechanisms, we implemented a Raspberry Pi-based system that conducts many critical operations at hardware level. CNN-ripple proper performance was validated in several *in vivo* experiments with mice, including several closed-loop ones with optogenetic stimulation.

In order to encourage the standardization and dissemination of closed-loop techniques in experimental neuroscience, all protocols and technologies developed as part of this thesis are available in open-source repositories.

Resumen

Comprender las dinámicas neuronales es una tarea compleja debido a que estas son altamente no lineales, parcialmente observables (solo podemos registrar unos pocos componentes del cerebro de manera simultánea) y se encuentran en un proceso de constante adaptación y cambio debido a la interacción entre las neuronas y la plasticidad asociada a ellas. El protocolo predominante para el estudio de dichas dinámicas, conocido como el paradigma de estímulo-respuesta, consiste en introducir estímulos estereotipados al sistema, registrar su respuesta y analizarla a posteriori de manera *offline*. Por lo tanto, este protocolo presenta limitaciones a la hora de interpretar dinámicas neuronales transitorias que en cada momento están recibiendo retroalimentación de cientos o miles de elementos.

Las técnicas experimentales de ciclo cerrado (*closed-loop*) en neurociencia surgen como una manera de hacer frente a estos problemas, puesto que permiten observar, controlar y estimular los sistemas neuronales con una interacción online basada en su propia actividad. Sin embargo, no siempre es sencillo implementar esta clase de protocolos ya que requieren de tecnologías de tiempo real (*real-time*) así como diseños experimentales más elaborados. Debido a esto, muchos laboratorios e investigadores en neurociencia desestiman el uso del paradigma de ciclo cerrado.

El objetivo de esta tesis es el desarrollo de tecnología software de tiempo real para la implementación de protocolos de ciclo cerrado en neurociencia experimental. Para ello se han llevado a cabo tres proyectos en los cuales se ha trabajado con diversas técnicas de adquisición de datos, detección de eventos, análisis y estimulación electrofisiológica en tiempo real. Estas tecnologías han sido validadas en diferentes escalas de descripción del sistema nervioso.

El primer proyecto ha consistido en el desarrollo de RTHybrid, una aplicación software para la construcción de circuitos híbridos entre neuronas biológicas vivas y neuronas simuladas por ordenador, útiles en el estudio de las dinámicas neuronales y el rol de elementos específicos dentro de los circuitos. Para ello incorpora una librería de modelos de neurona y de sinapsis, así como una serie de algoritmos de adaptación automática para llevar a cabo una correcta interacción entre todos los elementos involucrados. Puesto que en muchos casos dicha interacción de ciclo cerrado debe realizarse respetando unas restricciones temporales muy estrictas, se ha implementado esta herramienta sobre sistemas operativos de tiempo real. Se ha validado el correcto funcionamiento de la aplicación en experimentos *in vitro* con neuronas de invertebrados vivas procedentes de un circuito Generador Central de Patrones (CPG, siglas en inglés).

Para el segundo proyecto se ha construido el FLC-Hybrot, un robot hibrido (hybrot) cuya locomoción está controlada por la actividad neuronal de un CPG funcional y vivo de un cangrejo. El robot utiliza los invariantes dinámicos del circuito neuronal para organizar su movimiento y a su vez envía retroalimentación sensorial a las neuronas alterando su comportamiento y, por lo tanto, también el movimiento coordinado del hybrot resultante de este ciclo cerrado híbrido. Los componentes biológicos y electrónicos del hybrot se comunican a través de una conexión Bluetooth. Este proyecto supone la primera validación del uso de un principio dinámico emergente de un circuito vivo funcional para controlar un dispositivo robótico con retroalimentación sensorial.

Por último, el tercer proyecto llevado a cabo como parte de esta tesis ha sido el desarrollo

de CNN-ripple, un sistema de detección e intervención en tiempo real de los eventos electrofisiológicos conocidos como *sharp wave ripples* (SWR) mediante técnicas de *deep learning*. Se ha diseñado e implementado una red neuronal convolucional (CNN) capaz de identificar estos eventos tanto en contextos offline como online en señales registradas con diferentes tipos de dispositivos, así como en distintas especies animales. Para favorecer y facilitar el uso de este método, se ha añadido como un *plugin* a la plataforma Open Ephys, uno de los softwares de adquisición cuyo uso está más extendido en neurociencia experimental. Para reducir las latencias en la detección provocadas por el envío de datos, se ha implementado un sistema que hace uso de una Raspberry Pi para ejecutar el software, realizando muchas de las tareas más críticas a nivel de hardware. Se ha validado el funcionamiento de CNN-ripple en múltiples experimentos *in vivo* con ratones, incluidos varios de ciclo cerrado con estimulación optogenética.

Con el objetivo de favorecer la estandarización y diseminación de las técnicas de ciclo cerrado en neurociencia experimental, todos los protocolos y tecnologías desarrolladas como parte de esta tesis se encuentran disponibles en repositorios de código abierto.

Keywords

Closed-loop neuroscience, Real-time technology, Neurotechnology, Hybrid circuits, Hybrots, Sharp-Wave Ripples detection

Palabras clave

Neurociencia de ciclo cerrado, Tecnología de tiempo real, Neurotecnología, Círcuitos híbridos, Hybrots, Detección de sharp-wave ripples

Agradecimientos

Empezar agradeciendo a mi director Pablo por la oportunidad de realizar esta tesis así como por toda su ayuda y guía a lo largo de estos años. Agradecer también a Paco y Rafi todos sus consejos, ideas y aportaciones que han permitido que este trabajo haya llegado a ser lo que es. También dar las gracias a Liset por darme la ocasión de trabajar con ella, y a toda la gente de su laboratorio por hacerme sentir como uno más (y en especial a Teresa, que me ha tenido que aguantar durante tantos experimentos).

Por supuesto dar las gracias a toda la gente con la que he tenido la suerte de compartir laboratorio durante todo este tiempo: Manu, Irene, Aarón, Carlos, Guille, Ángel L., Ángel F., Vinicio, Jessica, Alicia, Blanca, Alberto y Pablo; no solo por sus imprescindibles aportaciones en los trabajos realizados, que directamente no se podrían haber hecho sin ellos, sino sobretodo por ser gente maravillosa con la que ha sido un placer compartir comidas, congresos, cervezas y karaokes (y espero que haya muchos más).

A mis padres y a mi hermano, por apoyarme siempre y mostrar interés e ilusión por todas las decisiones que he tomado: si he llegado hasta aquí es sobretodo gracias a ellos. A mis amigos, que aunque creo que todavía no saben que he estado haciendo exactamente todos estos años sé que siempre están ahí, animándome y aguantándome cuando las cosas no salen bien, y compartiendo conmigo los buenos momentos.

A Andrea, por haber compartido conmigo esta etapa de nuestras vidas.

Contents

Figures index

1 Introduction

1.1 Observing neural dynamics and implementing successful closed-loop interactions

Humanity has tried to understand the structure and functioning of the brain and the nervous system since prehistoric ages [Finger, 2001, Mohamed, 2008], looking for answers to how they work (or why they sometimes do not and cause mental health issues). These efforts have intensified in the last decades thanks to technological developments in biology, physics, electronics and computer science. However, studying neural systems dynamics is still a task hindered by several factors.

First of all, they are intrinsically non-linear because their function is to process information that is found in multiple distinct spatial and temporal scales, with some neural activity and events spanning during several seconds while others just last for some milliseconds, and producing electrical responses in a wide amplitude range. In addition to this, neurons are connected to other cells and their behaviour is affected by the interaction with them, sometimes receiving thousands of inputs and sending their information through a cascade of connections that can involve millions of neurons simultaneously. Even synapses, the connections between neurons, change accordingly to previous and ongoing activity through diverse transient learning and adaptation mechanisms.

Moreover, despite all the groundbreaking technological innovations achieved in the last years, we are currently able of accessing just a few of all the elements involved in such neural dynamics, which can be thousands, thus making the system only partially observable at each specific moment.

The third reason is the predominant use of the stimulus-response paradigm in experimental neuroscience, which is based on recording the behaviour of the neural system under the effect of different stimuli and studying the obtained data afterwards [Meehan and Bressler, 2012]. However, this approach is often unable of capturing the temporal singularities of the observed neural patterns, especially when it comes to their intrinsic variability and its meaning. This prevents a fully understanding of the neural activity, which is non-stationary and highly complex due to the influence received from its context and previous events' feedback.

One solution to overcome these difficulties is to interact online with the living system through

closed-loop techniques, which are capable of generating precise stimuli based on the ongoing dynamics (Figure 1.1). This approach provides a valuable new insight into these transient neural processes, allowing a more detailed characterization of them and unveiling time-dependent mechanisms. Additionally, it provides researchers novel tools to control normal and pathological dynamics, as well as to conduct goal-driven interactions and exploration with biological elements, favoring the design of more flexible and automatic experiments [Chamorro et al., 2012, Potter et al., 2014, Roth et al., 2014, Varona et al., 2016].

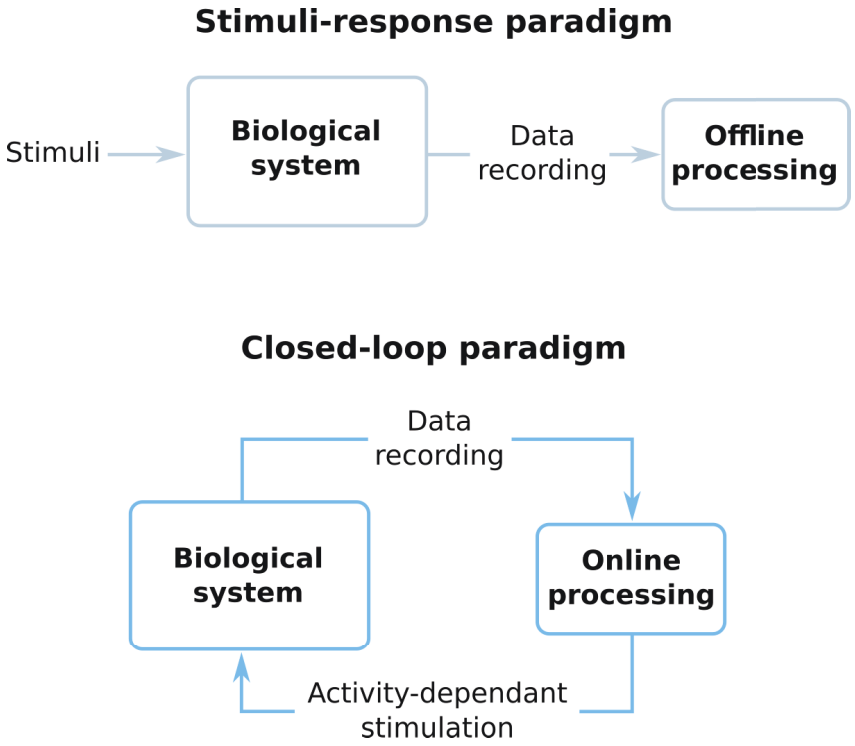

Figure 1.1: Representation of the classical stimulus-response paradigms and closed-loop protocols.

However, implementing closed-loop protocols is not always a simple task. Sometimes the experiment's own complexity is what hampers the design of online interaction paradigms, but in many cases the challenge comes from the technical difficulties to comply with certain temporal constrains. These restrictions can even be in the order of milliseconds or lower, both for data acquisition and activity-dependent stimulation of biological components during the experiments. Therefore, real-time technologies are required to face these issues [Christini et al., 1999, Muñiz et al., 2005, 2008, 2009].

Real-time can be defined as the capacity to perform a periodic task and respond to asynchronous external events in a strict time slot [Furht et al., 1991]. Computational power is indeed important for a real-time system, since it must carry out the assigned task within the established boundaries, but equally important is its ability to perform it with the required temporal precision, neither sooner nor later. Some operations may be fast to compute but still must be conducted in the appropriate moment. Many neurons follow precisely a similar behavioural pattern: slow activity (in the millisecond time scale) but precise subcellular sequential dynamics or spiking coding.

Numerous computer systems employ real-time technology nowadays, from common ones as mobile phones and on-board computers to others more specialized, such as NASA's Curiosity Rover landing device. Not all tasks are equally sensitive to latency or data loss, therefore existing

two categories of real-time. Soft real-time is used when some deadlines can be missed and the system still works but performance will degrade if too many are lost, for example an online music or video streaming application can lose a few packages and the service goes on unaffected. Hard real-time is employed when every deadline must be hit or the system fails, like the computer systems of a nuclear plant or satellite communication systems [Shin and Ramanathan, 1994].

Electronic components often have the power and precision required to easily build hardware-based real-time solutions [Robinson and Kawai, 1993, Masson et al., 1995, Broccard et al., 2017]. Moreover, currently there exists a wide range of low cost microcontroller boards with great computational power that can be utilized in efficient and adaptable implementations [Desai et al., 2017]. On the other side, software-based solutions are simpler to program and handle, providing great flexibility and a lot of usability options to the users. Furthermore, nowadays personal computers' hardware capability is more than enough to comply with most real-time requirements and their large memory sizes allow the implementation of more complex protocols. However, modern general purpose operating systems (GPOS), such as GNU/Linux, Windows or MacOS, are multitask environments with internal schedulers, which assign computer resources to different running tasks following specific policies [Stallings, 2012]. These schedulers can not be controlled by the users, hence it can not be ensured that a given task will run without interruptions and, therefore, real-time can not be assured. In order to run protocols compliant with a set of established temporal boundaries using software-based real-time, another system framework, known as real-time operating system (RTOS), is needed.

Regardless of the kind of technology employed to build a real-time system, it must be capable of carrying out data acquisition, data processing and stimulation of the biological elements within the required temporal boundaries. Data recording must be performed online independently of the acquisition technique used, both for neural signals or any other kind of input information utilized (e.g., video tracking, fMRI, electrocardiogram signals, etc.). The system then uses the registered data to generate stimuli based on the ongoing activity of the living subject, which can be analyzed and characterized using several distinct methods (e.g., neural events detection, spatial location of the subject, etc.). Activity-based stimuli are then inserted back into the biological system, whose response to them is again recorded, starting a new cycle of the closed-loop protocol. A wide variety of software and hardware can be utilized together in different steps of this process to achieve real-time performance.

Closed-loop and real-time technologies can be used in combination with several neuroscience recording and stimulation techniques. For example, there have been closed-loop experiments using fMRI [Rana et al., 2016], optogenetics [Krook-Magnuson et al., 2013, Prsa et al., 2017], EEG setups [Arrouët et al., 2005, Sitaram et al., 2016], neuroprostheses [Levi et al., 2018], and numerous types of activity-dependent stimulation configurations, such as the ones that use simultaneous electrophysiological and video tracking [Muñiz et al., 2011], acute mechanical stimulation [Muñiz et al., 2008], electric signaling during behavior [Forlim et al., 2015, Lareo et al., 2016] or drug microinjection [Chamorro et al., 2009, 2012].

Despite the great benefits that closed-loop protocols provide, the lack of flexibility or the difficulties in the design, implementation, installation and utilization that many real-time solutions present cause many researchers to overlook these experimental paradigms or to implement them using not optimal tools.

1.2 Thesis goals and structure

This thesis main objective is the design and development of novel real-time software technologies for its use in diverse scenarios in experimental neuroscience and, more specifically, for the

implementation of closed-loop paradigms with activity-dependent stimulation. This goal was carried out throughout different projects framed in several electrophysiology contexts with the purpose of studying and understanding multiple neural phenomena and dynamics. Moreover, on each of these works we implemented different real-time tools and applied them to distinct tasks and environments:

- Construction of hybrid circuits by connecting biological living neurons with mathematical neuron models simulated in a computer. Hybrid circuits are a powerful yet undeveloped tools to study neuronal dynamics and the role of specific components in neural circuits. Hybrid circuit interactions often require hard real-time precision in the order of microseconds and, therefore, a software application with soft and hard real-time capabilities was developed.
- Implementation of a hybrot by combining a robotic body with a biological "brain". A living functional neural circuit was employed to modulate the locomotion of the artificial body while it also received feedback information from the electronic sensors, forming a real-time closed-loop interaction between the various components to achieve flexible coordinated motion.
- Detection and intervention of electrophysiological events in real-time using a deep learning-based method. A convolutional neural network was designed in order to identify the events known as sharp-wave ripples in both offline and online scenarios with optimal performance and minimal latencies.

All the developed tools have been designed as user-friendly and open-source solutions in order to standardize and disseminate the use of real-time software technology in neuroscience. Furthermore, all of them have been validated in numerous experiments using distinct recording and stimulation techniques and animal species. This enabled the application of real-time and closed-loop protocols over several nervous systems levels, from motor neural networks of crustacean invertebrates to the hippocampal formation of rodents. Due to this variety, both in the experimental setups and the intrinsic nature of the events that were subject of this analysis, various real-time protocols were employed, including soft and hard real-time solutions in software and hardware environments. Regardless of the particular details of these specific validation experiments, the developed tools can be applied and utilized in a wide range of contexts and laboratories (e.g., to create hybrot using other neural systems, online detection of different electrophysiological oscillations and events, adaptable interactions, etc.).

This document is therefore structured in different chapters dedicated to each of the previously mentioned projects. Every one of them include their own introduction and conclusions sections where specific details are addressed in depth. The overall structure of this work is organized as follows:

- **Chapter 1. Introduction**: in this first section we present the main motivation and overall context for the development of this work, as well as its general goals.
- **Chapter 2. RTHybrid: a real-time software application to build hybrid circuits between living and model neurons**: this first chapter of the thesis work is about the design and development of a software application to build hybrid circuits with biological neurons and neuron models simulated in a computer. It can handle a hard real-time interaction in hybrid circuits by running over two different open-source RTOS, chosen after a performance and usability analysis of all available alternatives. Validation tests of hybrid circuits built using *in vitro* living neurons are presented as part of this section. RTHybrid is available as open-source software `https://github.com/GNB-UAM/RTHybrid`.

- **Chapter 3. FLC-Hybrot: functional living circuit dynamics driving the locomotion of a hybrid robot with closed-loop sensory feedback**: the second chapter of the thesis addresses the construction of a hybrot (hybrid robot) whose movement is modulated by the real-time activity of a functional living central pattern generator (CPG). FLC-Hybrot also include sensors that send feedback current to the CPG according to its surroundings, altering the neurons behaviour and thus forming a closed-loop interaction among biological and electronic components. Results from the validation trials using living CPGs to coordinate the hybrot are also detailed. All the software required to implement FLC-Hybrot can be accessed at `https://github.com/GNB-UAM/FLC-Hybrot`.

- **Chapter 4. CNN-ripple: an offline and online deep learning based detection method for real-time closed-loop intervention of sharp-wave ripples**: the third chapter of this thesis describes the development of a real-time detection system for the neurological events known as sharp-wave ripples (SWR) using deep learning techniques. To this end, a convolutional neural network (CNN) was designed and implemented. This method is capable of identifying SWR in both offline and online scenarios with a great rate of precision and sensibility while also reducing the detection latency. We developed plugins with this tool for one of the most extended electrophysiology acquisition and recording software, as well as a stand-alone version, in order to facilitate its use in closed-loop experiments. Open and closed-loop experimental sessions were conducted in order to validate the adequate performance of the proposed method. CNN-ripple is available as a GitHub repository in `https://github.com/PridaLab/cnn-ripple`.

- **Chapter 5. Conclusions and discussion**: in this last section we summarize and discuss the general results of this work and the developed tools.

- **Bibliography**

- **Appendices**:
  - **Appendix A**: links to the open-source repositories.
  - **Appendix B**: neuron and synapse models included in RTHybrid.
  - **Appendix C**: Publications related to this thesis.

2

RTHybrid: a real-time software application to build hybrid circuits between living and model neurons

2.1 Introduction

The first project carried out as part of this thesis was the development of RTHybrid, a software application designed to build hybrid circuits between living neurons and artificial neurons. This program includes a neuron and synapse model library, as well as a set of automatic calibration algorithms, to facilitate the construction of hybrid circuits in a simple and quick way. In order to ensure an accurate closed-loop interaction among all the components, and also to promote its accessibility and dissemination, the implemented tool works on various open-source real-time operating systems. Experiments with living neurons from invertebrates were conducted in order to validate the proper functioning of the developed technology.

The first section consists of an introduction of the project's motivation, the state of the art and the different elements involved in the development. The second one contains a detailed description of the materials and methods employed, followed by an explanation of the software implementation and the validation results. Finally, the overall outcome of the project is discussed.

2.1.1 Motivation and state of the art

As mentioned in the previous chapter, activity-dependent closed-loop techniques are very useful for the in depth study of complex non-linear neural dynamics. One of such protocols is the one known as hybrid circuit, which consists in the assembly of a neural circuit built with both living neurons and artificial neurons. They are a powerful tool to explore and characterize neural networks activity, as well as a mean to assess the role of circuit components (cells, ionic or synaptic currents, network topology building blocks, etc.) on specific neuron and network dynamics, e.g., see [Yarom, 1991, Pinto et al., 2000, Szücs et al., 2000, Varona et al., 2001, Masson et al., 2002, Nowotny et al., 2003, Oprisan et al., 2004, Arsiero et al., 2007, Chamorro et al., 2009, Grashow et al., 2010, Brochini et al., 2011, Kispersky et al., 2011, Wang et al., 2012, Thounaojam et al., 2014, Hooper et al., 2015, Norman et al., 2016, Broccard et al., 2017, Wang et al., 2022a]. At the same time, hybrid circuits are particularly suitable to validate hypotheses

formulated with computational models that address sequential neural dynamics [Torres et al., 2020, Garrido-Peña, 2019].

Artificial neurons can be simulated from computational models, and synapse models are also required to implement the connections between the various elements. The behaviour of these models depends on the activity recorded from the living neuron while they also affect its functioning, thus forming a closed-loop interaction between the living and computational components (Figure 2.1). Usually, the dynamic clamp technique is applied to biological neurons in order to record their activity while also injecting a specific synaptic feedback current into them [Robinson and Kawai, 1993, Sharp et al., 1993, Prinz et al., 2004b, Destexhe and Bal, 2009, Nowotny and Varona, 2015]. This technique allows the implementation of any kind of synaptic current that can depend on the presynaptic and postsynaptic membrane voltage. For a correct performance of the hybrid circuit it is necessary that all components, both living and computational, interact in real-time and usually within intervals in the order of microseconds. Hence, hard real-time technology is required.

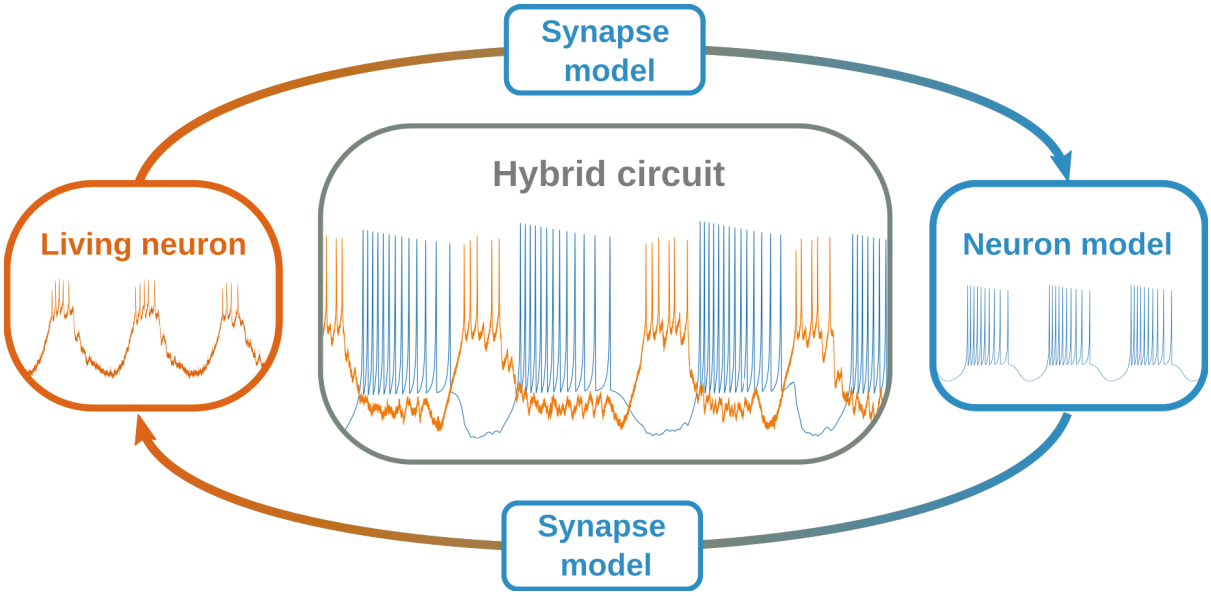

Figure 2.1: Example of a closed-loop hybrid circuit. The circuit is created by bidirectionally connecting a living neuron with a computational neuron model through synapse models.

Numerous real-time tools for experimental neuroscience exist already. Some of them are hardware-based [Franke et al., 2012, Tessadori et al., 2012, Müller et al., 2013, Desai et al., 2017]. Several software tools have been designed, particularly for dynamic clamp electrophysiology experiments, both following soft- [Pinto et al., 2001, Nowotny et al., 2006, Linaro et al., 2014, Ciliberti and Kloosterman, 2017, Hazan and Ziv, 2017] and hard real-time approaches [Christini et al., 1999, Dorval et al., 2001, Biró and Giugliano, 2015, Patel et al., 2017]. All of them use distinct platforms and RTOS, which have diverse purposes and architectures, hence presenting different advantages and disadvantages.

The goal of this project was the development of RTHybrid, a software application that enabled the creation of hybrid circuits in a flexible and easy way [Amaducci et al., 2019]. For this purpose, it contains a neuron and synapse model library, which can be expanded by the users, as well as an intuitive user interface. RTHybrid is a free and open-source program that runs over GNU/Linux operating systems and can handle both soft and hard real-time specifications, being compatible with the two currently most extended Linux-based RTOS. This work also includes an analysis and comparison of several available RTOS, in terms of usability and performance,

carried out in order to determine the most suitable environments for RTHybrid development. In addition to its stand-alone software implementation, RTHybrid is also available as a set of plugins for the RTXI platform, a popular hard real-time data acquisition and control application for biological research [Patel et al., 2017].

2.1.2 Biological basis

Living neurons are an essential component of hybrid circuits. To build such circuits it is not only necessary to be able to record the activity of the cells and stimulate them in real-time, but it is also equally important to understand their basic functioning and how they transmit information to each other through synaptic connections. The developed application was validated in a real experimental environment, building hybrid circuits using neurons from a specific living circuit known as central pattern generator (CPG).

Neurons

Neurons are the principal component cells of the nervous system [Cajal, 1911, López-Muñoz et al., 2006, Kettenmann and Verkhratsky, 2008, Kandel et al., 2013]. They are in charge of receiving, processing and transmitting information. This is achieved by generating action potentials (also called spikes), which are sudden changes in their cell membrane's electric potential due to the dynamics of the ionic concentrations inside and outside the cell. The membrane potential is regulated by means of the gating of ionic channels, allowing some specific ions through in each case, like sodium, potassium and calcium channels. These action potentials can be transmitted as electrical signals to other neurons through synaptic connections.

The neuronal membrane potential changes over time, giving rise to a wide range of behaviour influenced by the impulses received from other neurons. This way, a neuron can be usually silent and only generate spikes from time to time while in other cases display a continuous spiking pattern (Figure 2.2 left), even intercalating silence periods with spikes bursts (Figure 2.2 right).

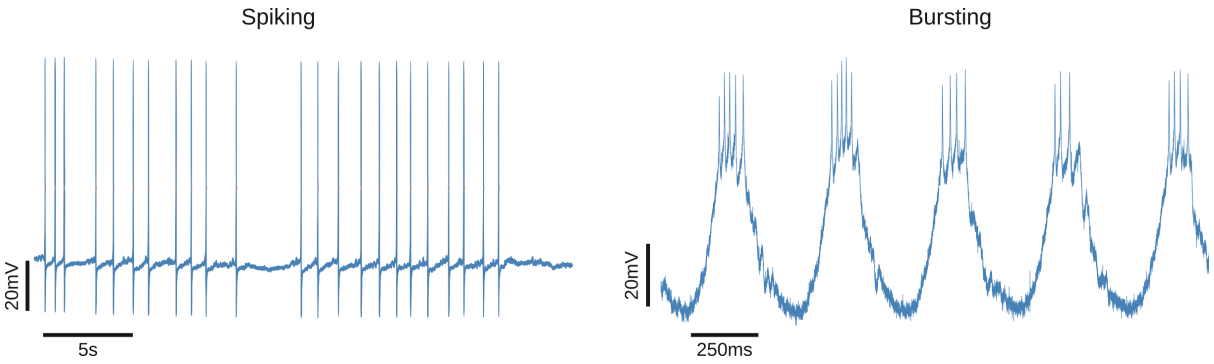

Figure 2.2: Example of two different neuronal membrane potential activity. The one on the left, from a right parietal ganglion neuron of a *Lymnea stagnalis* snail, displays an irregular spiking activity. On the right, a neuron from a *Carcinus maenas* pyloric CPG shows a regular bursting activity, grouping several spikes in a single burst that is followed by a quiescent period before the next burst. Signals kindly provided by Alicia Garrido-Peña and Irene Elices, respectively.

Synapses

Connections between neurons are called synapses and they mediate information transmission among these cells [Kandel et al., 2013]. The sending neuron is known as the presynaptic neuron

while the receiving one is called postsynaptic neuron. A synapse can have an excitatory or inhibitory effect in the postsynaptic neuron activity depending on its biophysical implementation. Moreover, the synapses themselves can also be classified according to the employed transmission mechanism as electrical or chemical.

Electrical synapses directly transmit electric impulses between both neurons, so they are very fast and generally work bidirectionally. This kind of synapse takes place when two neurons are so close to each other that some proteic structures, known as gap-junctions, can be created among them and the ions can pass straight through them. This leads to the synchronization of their membrane voltage [Connors, 2017].

On the other hand, chemical synapses can occur when both neurons find themselves more separated, and so they need to establish the connection via chemical components known as neurotransmitters, which travel from one to the other. When a presynaptic action potential reaches the synapse, this releases the neurotransmitters which then adhere to the neuroreceptors present in the postsynaptic neuron membrane, causing its ionic channels to open and thus altering the postsynaptic membrane potential [Feng, 2009].

Central pattern generators

Central pattern generators, commonly known as CPGs, are neural networks which control rhythmic motor functions (such as walking, swimming, breathing, etc.) through the generation of sequential rhythmic patterns [Marder and Eisen, 1984, Grillner, 2003]. They typically have what is called a non-open topology, i.e., every neuron has at least one synaptic input from another neuron of the circuit (Figure 2.3 left) [Huerta et al., 2001]. At the same time this architecture is frequently based on oscillatory circuits with pairs of neurons with mutual synaptic inhibition [Miller and Selverston, 1982, Sakurai et al., 2014]. Despite of these networks being mostly formed by neurons with irregular dynamics, the reciprocal inhibition between them typically produces a regular behaviour when they are connected to other neurons of the circuit [Elices and Varona, 2015, 2017]. This rhythm negotiation in terms of alternated sequential activity, which derives in strong rhythmic patterns, is the reason why CPGs can efficiently coordinate motor functions [Selverston et al., 2000].

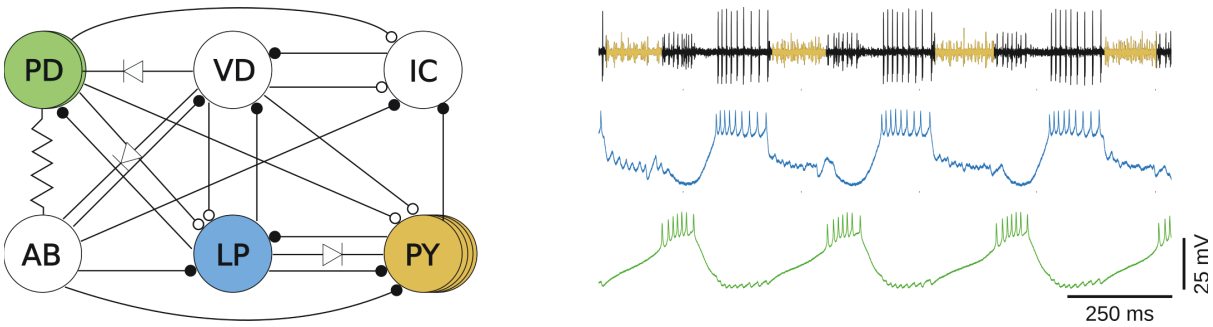

Figure 2.3: Left: schema of a *Carcinus maenas* pyloric CPG structure, representing its various neurons and the connections between them. One-way electrical connections are represented with a diode symbol, while a resistor indicates a bidirectional one. Fast chemical inhibitory synapses are depicted with a black circle next to the postsynaptic neuron, and white circles represent slow chemical inhibitory connections. **Right:** example of the triphasic rhythm produced by the LP, PD and PY neurons of a pyloric CPG. LP (blue trace) and PD (green trace) signals were captured intracellularly while the PY activity (yellow trace) can be inferred from an extracellular recording.

Validation tests for RTHybrid were performed using neurons from a pyloric CPG of a *Carcinus maenas* (i.e., shore crab), which displays a very robust triphasic rhythm between the bursts of the LP (blue trace), PD (green trace) and PY neurons (yellow trace over extracellular recording) (Figure 2.3 right).

2.1.3 Neural recording and stimulation techniques

Electrophysiology deals with the study of the electrical properties of biological cells and tissues, such as membrane potentials or ionic currents and their associated function [Covey and Carter, 2015]. Electrophysiological techniques have been applied in experimental neuroscience for 200 years, when it was discovered that the nervous system operations are based on electrical activity [Galvani, 1791, Scanziani and Hausser, 2009].

Since then, numerous recording techniques have been developed, employing a wide range of materials and configurations to capture neural signals with different temporal and spatial resolutions [Cogan, 2008, Hong and Lieber, 2019, Chen et al., 2017]. During the last five decades the amount of neurons that can be simultaneously registered has doubled approximately every seven years, following Moore's Law [Stevenson and Kording, 2011]. Similarly, a great variety of stimulation mechanisms has emerged since the first days of electrophysiology, such as electrical injection [Histed et al., 2009], drug microinjecton [Chamorro et al., 2009] or optogenetics [Cavanaugh et al., 2012, Diester et al., 2011], among many others. The different techniques used in this project are summarized below.

Intracellular recordings

Intracellular recordings consist of registering neuronal membrane potential fluctuations, caused by the flow of ions over time. They are obtained by introducing a glass electrode with electrolyte solution into the cell [Lalley, 2009]. In order to cause the least possible damage to the cell, and to have a high impedance, the electrode must be really thin. This technique allows great spatial and temporal precision in the observation of neural activity events, such as spikes and bursts, and even some others that would be very difficult to capture otherwise, like the quantization of neuron's resting membrane potential or the effect that synaptic inputs have on it.

However, it requires a complicated procedure and it is not always easy to locate and penetrate a specific neuron. It also demands a costly and highly specialized equipment to carry out multiple concurrent intracellular recordings. Moreover, great stability is needed to perform this technique, since just piercing its membrane entails a huge risk for a neuron's integrity. Figure 2.2 displays some examples of intracellular recordings from different neurons.

Extracellular recordings

Extracellular recordings consist of registering the activity of neurons or nerves with electrodes that are situated near the cells, but not inside [Windhorst, 2009]. Different kinds of electrodes can be utilized to record the activity from both individual neurons or the joint activity of a group of them.

When the electrodes are placed just next to a neuron's membrane, without entering inside, single-unit recordings are obtained, where the action potentials of an individual cell can be identified. When an electrode is placed near a nerve the activity of several individual cells can be recorded (Figure 2.4 top). In other cases the electrode can acquire multi-unit signals that represent the collective activity of the various neurons located nearby. These signals are known as local field potentials (LFP) and, even thought they do not provide detailed insights of the

neurons behaviour, they capture the rhythms, trends and oscillations of the collection of neurons (Figure 2.4 bottom) [Hong and Lieber, 2019]. Single spikes from the neighbouring neurons can also be identified in LFP traces. Numerous spike sorting methods have been and are still being developed in order to relate these action potentials captured in extracellular signals to their source neuron [Rey et al., 2015].

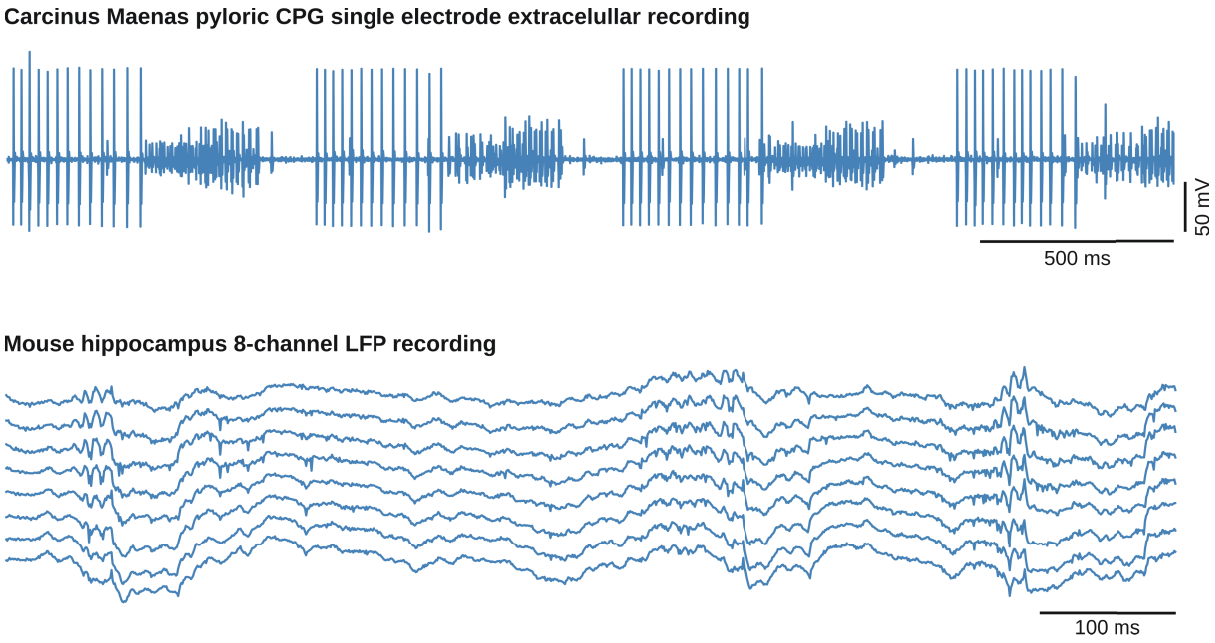

Figure 2.4: Example of two different extracellular recordings. The first was acquired with a single electrode in a *Carcinus maenas* pyloric CPG and the activity of several individual neurons of that circuit can be identified on it by the amplitude and waveform of the action potentials. The second is from a mouse hippocampus and consists of 8 simultaneous LFP recordings that display the aggregated activity of all nearby cells. CPG signals kindly provided by Irene Elices and LFP recordings by Teresa Jurado-Parras, from De La Prida's lab at Instituto Cajal (CSIC).

Another technique used to acquire extracellular recordings is the one known as electroencephalography (EEG), which allows to register the collective activity of neurons in a non-invasive manner by placing electrodes on the scalp. For this reason, it is a widely spread tool both in research and clinical environments, especially with human subjects [Cohen, 2017].

Electrical injection stimulation

Given that neurons' activity is expressed in terms of their membrane potential and its ionic currents, it is possible to modify their behaviour by injecting electrical current into them. Electrical stimulation has an analogous effect to electrical synapses, exciting or inhibiting the cells according to the inserted current. This technique was first employed in 1870 [Fritsch and Hitzig, 1870] and since then has become one of the most utilized neural stimulation methods. Similarly to the recording options, electrical stimulation can be intracellular [Hodgkin and Huxley, 1952a, Suk et al., 2017] or extracellular [Fitzsimmons et al., 2007, Histed et al., 2009, Hughes et al., 2021].

Dynamic clamp

The first electrophysiological real-time activity-dependent stimulation technique, called voltage-clamp, was developed sixty years later in order to better understand previous findings by [Cole and Curtis, 1939] and [Hodgkin and Huxley, 1939]. This was later used in experiments to characterize the electrical properties of neuronal membranes which made possible the first mathematical model of the action potential [Hodgkin and Huxley, 1952a]. The voltage-clamp technique uses an intracellular electrode to measure the neuron's membrane potential and a second one to inject the necessary current to keep the membrane potential "clamped" at a constant value. The amount of current required to keep the potential fixed on the same value is derived in real-time from the measured membrane potential, thus forming a closed-loop.

New possibilities, such as the space clamp, the patch clamp and the single-electrode voltage clamp were developed after the original voltage clamp. [Robinson and Kawai, 1993, Sharp et al., 1993] introduced a new technique called dynamic clamp, which allows the user to control the neuronal membrane potential by creating new conductances through the injection of simulated ionic or synaptic currents based on the measured membrane potential. Therefore, more complicated experiments can be performed, such as connecting living neurons with computational models in real-time to create hybrid circuits, and more properties of the neurons and their connections can be studied [Prinz et al., 2004a, Nowotny and Varona, 2015].

2.1.4 Neuron and synapse models

At least one biological neuron connected to a computationally-simulated one is required to build a hybrid circuit. This simulated neuron can be computed using mathematical models, that is to say, equation systems that represent a neuron's dynamics. Likewise, synapse models are required in order to establish a connection between living and computational elements so the conductance transmission among the two neurons can be calculated [Torres and Varona, 2012].

Neuron models

Neuron models are equations systems that describe the behaviour of a neuron over time taking into account possible external inputs. There are very realistic models able of reproducing with great precision the activity of a specific neuron, at the expense of a high computational cost. Some others are less complicated and lighter and can be used to represent neural dynamics in a simplified way, such as continuous spiking, bursting, irregularity, etc.

The Hodgkin-Huxley model was the first one to realistically reproduce the membrane potential of a neuron as a function of its ionic currents with a system of non-linear differential equations [Hodgkin et al., 1952, Hodgkin and Huxley, 1952a,b,c]. The original model described the total current flowing through the membrane of an excitable cell with sodium (Na), potassium (K) and leak (l) ionic channels using Equation 2.1,

$$I = C_m \frac{dV_m}{dt} + \bar{g}_K n^4 (V_m - V_K) + \bar{g}_{Na} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l) \tag{2.1}$$

where V_m represents the membrane potential and C_m the capacitance. This model was used as cornerstone for many biophysically realistic neuron models that came later. All of these models represent the membrane potential of the cell with a generalized version of the Hodgkin-Huxley equation (Equation 2.2):

$$\frac{dV_m}{dt} = \sum_i I_i + I_{ext} \tag{2.2}$$

where I_{ext} represents any possible external currents and each ionic channel is described with an expression of the form

$$I_i = \bar{g}_i(V_m - V_i) \tag{2.3}$$

with $\bar{g}_i$ representing the maximal conductance of that ionic channel and V_i the reversal potential (membrane potential value at which there is no flow of these specific ions to either side of the membrane to the other). Each particular ionic channel can incorporate activation or inactivation variables described by Equation 2.4, with α_x and β_x being rate constants that depend on the membrane voltage. For the Hodgkin-Huxley model, the sodium channel has the n activation variable while the potassium channel has an activation variable m and an inactivation one h, with their constants being the ones described in Equations 2.5 and 2.6, where $V = V_{rest} - V_m$.

$$\frac{dx}{dt} = \alpha_x(V_m)(1-x) - \beta_x(V_m)x \tag{2.4}$$

$$\alpha_n(V_m) = \frac{0.01(10-V)}{exp(\frac{10-V}{10}) - 1} \qquad \alpha_m(V_m) = \frac{0.1(25-V)}{exp(\frac{25-V}{10}) - 1} \qquad \alpha_h(V_m) = 0.07exp(-\frac{V}{20}) \tag{2.5}$$

$$\beta_n(V_m) = 0.125exp(-\frac{V}{80}) \qquad \beta_m(V_m) = 4exp(-\frac{V}{18}) \qquad \beta_h(V_m) = \frac{1}{exp(\frac{30-V}{10} - 1)} \tag{2.6}$$

This kind of models are known as conductance-based models, and ionic channels can be added or removed from them to generate new dynamics. Several conductance-based models can be combined to create more realistic ones, including a detailed spatial description of the morphology of a neuron. These models are called multi-compartmental, since every subset of equations describes one compartment or an isopotential part of the whole neuron. They can have from two to any number of compartments, with the corresponding increase in computational cost.

On the other hand, integrate-and-fire models are much more simpler representations of the membrane potential, focusing only in producing spikes given some inputs that change over time [Abbott, 1999]. The shape of the generated activity is not usually very realistic, but this kind of models are also much less computationally demanding. There exist also many intermediate models in between integrate-and-fire and conductance-based ones, able of reproducing some realistic neural characteristics and dynamics with little computational cost [Torres and Varona, 2012].

Synapse models

Synapse models describe the connections between two neurons as a function of their membrane potentials, and there exist models for both electrical and chemical synapses. Resulting current for a electrical synapse consists simply in the difference of membrane potential between the postsynaptic and presynatic neurons, multiplied by the synaptic conductance, so it can be modelled just using Ohm's Law (Equation 2.7, where V_{post} and V_{pre} are the post and presynaptic potentials respectively and $\bar{g}$ is the conductance).

$$I_{syn} = \bar{g}(V_{post} - V_{pre}) \tag{2.7}$$

Chemical synapses can be derived following the same principle, describing the current generated by the changes in membrane conductance produced by the linking of neurotransmitters to neuroreceptors, thus including more elements to represent their more complex functioning. Chemical synapses can modulate various dynamics, such as gradual, immediate, fast, or slow ones, and even some that change over time and are modelled using differential equations. Models of chemical synapses can include the nonlinear dependence of the presynaptic and postsynaptic membrane potentials [Torres and Varona, 2012].

2.1.5 Real-time operating systems

For this project we decided to utilize real-time environments based on software because of the higher flexibility they provide when it comes to adding and using different models and also setting the various parameters of hybrid circuits experiments. When a computer performs a given task, there is always some latency between the moment when this task is expected to be accomplished and when it is actually done, as well as some jitter of these latency values, due to the performance of the operating system scheduler [Stallings, 2012]. In general purpose computers these schedulers can not be controlled by the users, hence it can not be ensured that a given task will run without interruptions and, therefore, real-time performance can not be assured (Figure 2.5 left). Real-time operating systems (RTOS) are needed in order to run protocols accomplishing a set of established temporal restrictions with software-based implementations. There exist several commercial RTOS solutions but they have not been considered for this work due to their high prices and also to promote the dissemination of open-source developed technology.

There are different ways of implementing a RTOS, but all of them rely in two common points: first, their scheduling algorithms, many of which are preemptive, meaning that they can interrupt a task running on the processor without its permission; secondly, the way they handle hardware interruptions. There are full real-time oriented RTOS and others that are based on GPOS, which are converted into real-time environments by patching their kernel. In this work we analyzed three open-source solutions based on GNU/Linux.

RTAI

Real-Time Application Interface, mostly known as RTAI, is one of the first and most extended open-source solutions to get a Linux-based RTOS. Developed since 1996 by Paolo Mantegazza [Mantegazza et al., 2000], it is a dual-kernel solution, meaning that it uses a microkernel that handles the real-time functionalities along with the standard kernel (Figure 2.5 center). This patch became one of the most reliable and widely used, and was the cornerstone for later RTOS projects as Xenomai.

RTAI functioning is based on a Hardware Abstraction Layer (HAL), over which run both Linux standard kernel and a microkernel with the real-time capabilities. Interruption requests (IRQ) generated by hardware components are first intercepted by RTAI kernel [Barbalace et al., 2008] and, in case they do not alter real-time performance, they are sent to the HAL, which resends them to Linux. In this structure, Linux is treated as a low priority process managed by RTAI real-time kernel, which handles the interruptions and can preempt the other operating system tasks when considered necessary.

At first, the HAL implementation used was RTHAL, which collected all the pointers to functions and data structures relevant for time critical operations in a single structure, so it

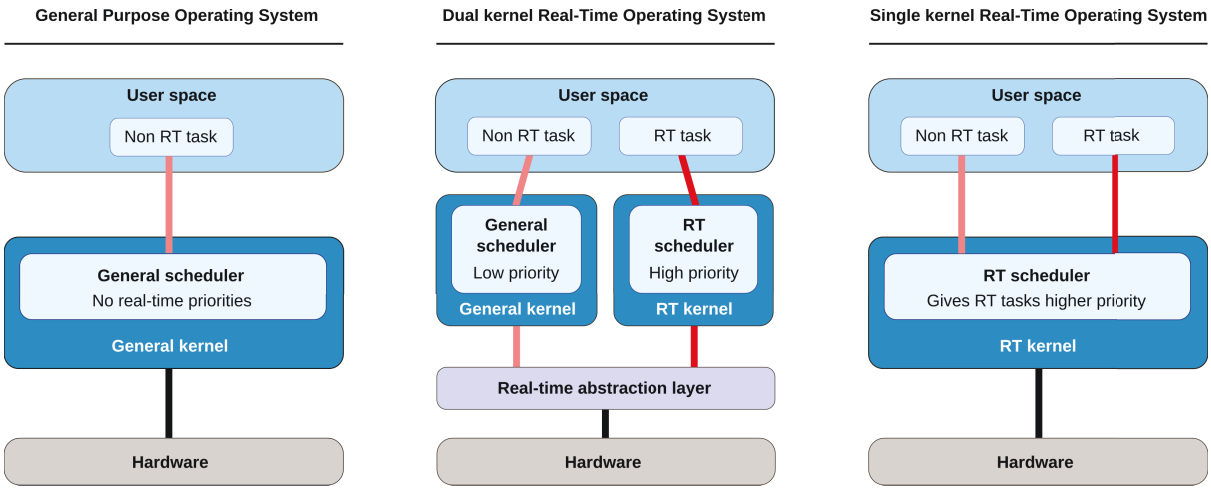

Figure 2.5: Left: general purpose operating system architecture. All processes created by the user are handled by the system's scheduler, which assigns computer resources to different running tasks following specific policies that can not be controlled by the user. **Center:** dual kernel real-time operating system architecture. A real-time kernel is used alongside the general kernel. An abstraction layer handles the computer resources and gives priority to the real-time kernel. **Right:** single kernel real-time operating system architecture. There is only one kernel but its scheduler and some other elements are designed to give real-time tasks preferential priority.

was easier to replace all these standard kernel functionalities with RTAI ones when hard real-time was needed. This idea was similar to the used by other RTOS projects active in the late 90's, RTLinux [Yodaiken, 1999]. Since there was a dispute for the patent, and to avoid further problems, this implementation was replaced in RTAI 3.0 by an open-source one, ADEOS, which offered slightly greater maximum latencies but works just as fine for most applications [Zhang et al., 2006].

ADEOS handles the different operating systems that are mounted over it as domains with priorities, allowing and managing the shared usage of hardware resources among them [Raghavan et al., 2005]. The highest priority domain receives the IRQs first and, if it is not interested, sends them to the pipeline, which propagates them to the next highest domain, and so on.

Four different schedulers are included in RTAI [Dozio and Mantegazza, 2003], with preemptive implementations of policies FIFO (First In First Out, the process does not leave the processor until it finishes its task or is preempted), Round-Robin (same as FIFO, but with limited time slots) and EDF (Early Deadline First, the user assigns estimated execution times to each process, and the smaller this time is, the higher the priority). The four schedulers are Uniprocessor (UP), optimized for one processor computers; Symmetric Multi Processor (SMP), able of running tasks in any CPU or fix it to one particularly; Multi Uniprocessor (MUP), which forces the task execution into a specific CPU in return for a better use of its resources and performance; and NEWLXRT, which unifies the previous three and is able of handling both RTAI and Linux processes.

RTAI is implemented as kernel modules that are loaded when they are necessary, except for the basic module that always needs to be loaded. Therefore, by default, real-time programs must work in kernel space and be loaded as modules. Kernel programming is quite limited since only functions and libraries loaded in kernel can be used, so a RTAI extension is included to allow the use of real-time services in user space. This extension is called LXRT and when the real-time initialization function is called from user space it creates a task in the microkernel,

linked to the Linux process, that provides real-time utilities.

However, no system calls or kernel services can be used while working in hard real-time mode in user space, since this would migrate the task execution back to Linux scheduler control until the standard kernel operation is completed, which prevents ensuring that the time restrictions will be respected and might cause unexpected behaviours. Hence, this practice is not recommended unless there is complete awareness of which calls and services the program is going to use [Racciu and Mantegazza, 2006].

Most of the modules that RTAI incorporates are related to Inter-Process Communication (IPC), i.e, mechanisms to communicate processes running over the same domain or different ones. These services include FIFO queues, shared memory, semaphores, mailboxes, remote procedure calls, event flags and message queues. In addition to these, it also includes its own malloc allocation implementation, tasklets for periodic or event-triggered functions or POSIX-threads support.

Xenomai

Xenomai is a dual-kernel solution that provides hard real-time support over a Linux kernel, trying to be as integrated with it as possible [Yaghmour, 2003] (Figure 2.5 center). Born in 2001, and merged with the RTAI project in 2003, Xenomai started a path on its own in 2005, becoming the open-source dual-kernel preferred tool for obtaining a RTOS.

In order to provide the most comfortable user experience possible it emulates the system calls and structure from other architectures and RTOS, such as POSIX, VxWorks, pSOS+, VRTX, ulTRON and RTAI, calling these libraries "skins". It also includes a new API, called Native, that allows developing real-time applications without previous knowledge of any of the other APIs, and the Real-Time Driver Model (RTDM), which provides a development interface for real-time devices drivers.

One of the most important elements of its architecture is an evolution of the ADEOS pipeline, called in this case I-Pipe or Interrupt Pipeline, which Xenomai uses as a virtual pipeline between hardware components, the standard Linux kernel and its own real-time microkernel. I-Pipe organizes the system in domains that share a common address space, which allows a process to use both microkernel and Linux kernel resources if needed, where Xenomai real-time domain has the highest priority and treats Linux kernel as a lower priority process. When an event arrives to the pipeline, Xenomai handlers manage it first and decide which domain will take care of it. I-Pipe architecture is easily exportable to other CPUs, but has to be specifically adapted to the Linux version that is going to be used.

The different APIs emulators provide the basic operating system resources that Xenomai needs, since having a distinct kernel for the real-time capabilities isolate its processes from the standard one services. The emulators are build combining and specialising the different basic blocks provided in the nucleus module. The main elements of this module are the following:

- A real-time threads object controlled by Xenomai's scheduler. It is preemptive and able of managing multiple priority levels, as well as scheduling types as FIFO and Round-Robin. All the skins thread management (priority management, preemption, thread suspend, etc.) are based on this abstract object.
- An IRQ handling object to connect to hardware handlers, using a simple mechanism that fits in the complex solutions that the different emulators implement.
- A memory allocation object that is bounded by predictable latencies.

- A synchronization object that implements the thread lock mechanisms for resource access (by priority, FIFO, etc.) on which mutexes, queues, semaphores and mailbox are based.
- A time management object, setting its own timebase, with support to nanoseconds and clock ticks, so the different skins timers can be used separately but concurrently.

The dual-kernel approach carries problems to use Linux kernel utilities in real-time programs, since they are not bound to latency restrictions and may cause unexpected behaviour on the real-time side, and because both kernels work independently. Xenomai deals with this issue by creating its real-time threads from standard POSIX threads, keeping all their functionalities while running in non-critical-time mode, and using the Real-Time Shadow extension, that allows the Linux-like task to be handled by Xenomai's scheduler when time restrictions are needed.

Each skin also implements a new set of system calls, that are included in libraries that replace the standard Glibc in the microkernel. As mentioned before, both kernels work separately, which can cause troubles in different situations, for example, if a Linux task is in the middle of a critical section and is preempted by a Xenomai one, that also makes changes on that section, causing the first one to fail when it is woken up. To prevent this from happening there exist two running modes: the primary one, controlled by the microkernel, and the secondary one, controlled by the standard kernel. Each kernel system calls can only be done from the proper mode and Xenomai puts each thread in the correct one depending on the system calls invoked. This scheduler change is called domain migration.

Preempt-RT

Since the apparition of the RTLinux project mentioned before, numerous efforts have been made in order to make Linux kernel fully preemptable [Dietrich and Walker, 2005], and therefore turn it into a RTOS by itself without the need of microkernels (Figure 2.5 right). Nowadays, all the changes and improvements made in this regard can be found in the Preempt-RT patch, which offers a viable alternative to dual-kernel systems.

This project was started and directed for many years by Ingo Molnár, and since then it has been mainly supported and funded by the Open Source Automation Development Lab (OSADL), until 2015, when it was transferred to The Linux Foundation, meaning more funding, developers and support. Currently, more than 80% of the Preempt-RT patch is already included in the mainline Linux kernel, due to the fact that part of the improvements solve some general problems that were found in the kernel when it was made preemptive, mainly related with locks and race conditions, and help increasing the standard kernel stability and performance.

The modifications made by this patch to the vanilla kernel in order to make it fully preemptive are the following:

- Re-implementation of the mutexes to make kernel spinlocks preemptive. An spinlock is a software lock over a resource where the locking process checks constantly if the resource is available (active waiting) so the scheduler thinks it is active even if it not really doing any task. They are useful for short and important locks, as kernel ones.
- Critical sections protected by *spinlock_t* and *rwlock_t* are now preemptive. In kernel space no preemptive critical sections can be created with *raw_spinlock_t*.
- Implementation of priority inheritance for kernel spinlocks and semaphores to solve priority inversion [Sha et al., 1990]. Priority inversion is a situation that happens when a process prevents another one with higher priority of completing its task. This may happen, for

example, if a low priority process (1) creates a lock over a resource and then is preempted by a higher priority one (2), but then a process with priority higher than both of them (3) requires access to the locked resource, unsuccessfully until process 2 releases the processor and lets process 1 complete its task and release the lock. This is a problem since there is no way of calculating the time that the higher priority process will be waiting.

Priority inheritance is a mechanism to solve this issue, consisting in giving to a process that is holding a lock over a resource the priority of the highest priority process that is waiting at that moment for that same resource until it releases the lock. By doing so, at the previous example, process 1 would have the same priority as process 3, thus process 2 would not be able of preempting it before releasing the resource.

- Interrupt handlers are run as preemptive kernel threads [Henriques, 2009]. In this manner, these threads have a specific priority and cannot preempt another process running at that moment with a higher one. In any other aspect, these threads imitate the behaviour of the standard handlers, like having CPU affinity, for example.

- Implementation of a new set of high resolution timers. Linux kernel measures time in jiffies, a variable time unit defined by kernel's constant *HZ*. Alarms storage is implemented through a timer wheel, an structure divided in buckets. The first wheel's layer represents the next 256 jiffies in the future, one per each of the 256 buckets that it contains. The next layers also contain 256 buckets, but each one of them represent 256 jiffies. For example, if an alarm is programmed for 20 jiffies in the future, it will be saved at the 20th bucket of the first layer, but if it is for 276 jiffies in the future it will be placed in the second layer. When the time represented by the first layer passes, the elements stored in the first bucket of the second layer have to be rehashed to the unitary buckets of the first one, an operation that have an O(n) cost, where n is the number of elements moved. Moreover, the computational cost of adding or removing a timer from the wheel is O(1), being a really cheap operation in contrast to rehashing.

  While trying to find an improvement to this structure, Thomas Gleixner [Gleixner and Niehaus, 2006] realised that there were two kind of timers being stored in the timer wheels: the action timer and the timeout timers. The former are those used by processes to be notified of an event and are removed after that, thus if they are programmed for a far future moment they will be rehashed many times, but added or removed just a few, so using a timer wheel for these timers is quite inefficient (lots of O(n) cost operations against few O(1)). On the other hand, timeout timers are triggered when an event does not occur (for example, a network package that does not arrive). These kind of timers stay a short time at the timers wheel (few O(n) costs) but are added and deleted frequently (many O(1) costs), so they do benefit from the timers wheel paradigm.

  With this in mind, Gleixner designed the following solution: he implemented a new structure, called hrtimers, that would store the action timers using red/black trees, instead of hash tables, having these an O(log n) cost for adding or removing elements (O(1) for the first element), but no cost for rehashing since the tree is already sorted. Additionally, hrtimers work in nanoseconds instead of jiffies, so its performance depends on the hardware clock and not a software constant. This improvement was added to the mainline Linux kernel 2.6.

Commercial RTOS

There exist multiple commercial options, which are specifically designed to be hard real-time operating systems. Their elevated price is a main drawback for many research groups and

laboratories, that cannot afford them. Moreover, they are usually oriented for their use in embedded systems, therefore lacking the flexibility and usability than GPOS offer.

One of the most widely used commercial RTOS is VxWorks, created by Wind River Systems in 1987 and nowadays used at BOEING-777 airplanes and NASA's Curiosity Rover, among others. In 1982, the company Quantum Software Systems released QNX Neutrino, one of the first real-time kernels that reached the market and used in a wide range of embedded systems, from mobile phones to cars. Microsoft also developed a real-time version of its Windows OS in 1996, called Windows Compact Embedded, which they used for their Windows Phones. Other proprietary software-based real-time solutions are RTOSWin, that runs paravirtualized real-time software next to Windows; xPC, a MATLAB's toolbox that runs the real-time functionalities in another device; or National Instruments' LabVIEW Real-Time, that works in a similar way than xPC. Finally, it is important to emphasize that there are works and papers comparing different RTOS [Hambarde et al., 2014][Aroca and Caurin, 2009] and they show that performance of open-source solutions is similar or even better in some cases.

2.2 Materials and methods

2.2.1 RTOS benchmarking

Performance on the three RTOS described in the previous section, RTAI, Xenomai and Preempt-RT, was measured and compared among them and also to a GPOS with no real-time capabilities. Specifications of the computers used for these tests can be found in Table 2.1. Most modern computers currently have multi-core processors, i.e., one component with several independent processing units. Linux also allows to isolate specific cores, so the scheduler will not assign them any task, and to manually bound an specific task to this empty core. We have also analyzed how this core isolation affected the performance in both real-time and non real-time implementations.

	Operating system	Kernel version	RAM	Processor	Cores
Non real-time	Debian 9	4.9.0-4	16 GB	Intel Core i7-4790 3.6 GHz	4
Preempt-RT	Debian 9	4.9.0-4	16 GB	Intel Core i7-4790 3.6 GHz	4
Xenomai 3.0.5	Ubuntu 16.04	4.9.90	16 GB	Intel Core i7-4790 3.6 GHz	4
RTAI 3.4	Ubuntu 10.04	2.6.34.5	4 GB	Intel Core i7-2600 3.4 GHz	4

Table 2.1: Software and hardware specifications of the computers used on the benchmarking tests for each operating system and RTOS used in this study. Note that the computer used for non real-time, Preempt-RT and Xenomai 3 tests was the same (we will call it Computer 1 hereafter) and the one for RTAI was different (Computer 2).

The benchmarking procedure consisted in a latency test, measuring the time difference between when an action was expected and when it really happened (see Figure 2.6). The benchmarking program[1] consisted in a periodic loop with a frequency of 20kHz that sent a digital 0 or 1 to a digital acquisition (DAQ) device alternately on each iteration, thus producing a 100μs square-wave signal. After sending the corresponding value the program slept until the next interval arrived: the time interval between the real and the expected awaking time was the measured latency. An Agilent MSO7104A oscilloscope was used as an external temporal reference and *stress*[2], a workload generator software, was utilized to create a worst case scenario with all processor cores and the file Input/Output system running at full capacity. To send

[1] Source code of the latency test: `www.github.com/RoyVII/Latency_tests.git`
[2] *stress* software website: `www.people.seas.harvard.edu/~apw/stress/`

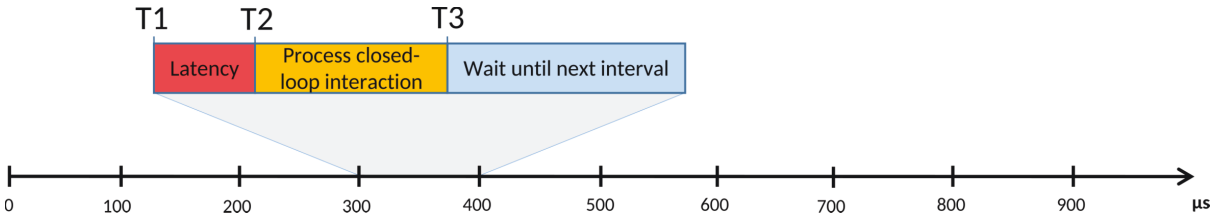

Figure 2.6: Representation of a typical real-time process, consisting on a succession of fixed and equally long time intervals that are repeated on each iteration of a periodic loop. For any of these intervals, T1 is the time when it should start, but, due to the system capacity or to its resource management algorithms, this is delayed until T2. The difference between the real start time (T2) and the expected one (T1) is known as latency. After T2, the process computes its iterative task, which finishes at T3. If T3 happens before the next interval start time, the process waits until that moment. On the other hand, if T3 occurs later than the expected start time of the next interval, as a result of a high latency value, a long computational time or both, it would cause a failure on the real-time system. Real-time tasks can be classified according to their tolerance of such events: soft real-time means that some deadlines can be missed under a determined limit, while hard real-time does not tolerate any failure.

the signal to the oscilloscope a National Instruments PCI-6251 board and a BNC-2090A DAQ device were used.

2.2.2 Experimental setup

In order to validate the proper functioning of RTHybrid, it has been tested in a real experimental environment. Hybrid circuits were built by bidirectionally connecting a neuron model simulated by the application and a living neuron, using chemical graded model synapses. For these tests, a computer with a 4-core Intel Core i7-6700 3.40 GHz processor and 16 GB RAM memory, as well as a National Instruments PCI-6251 board with a NI BNC-2090A DAQ device, were used. Measures of the time spent by each task in the interaction cycle were also conducted to evaluate their contribution to minimum and maximum computational cost for each of the real-time platforms analyzed.

The membrane potential of the biological component of the experiment was recorded using *in vitro* electrophysiology in neurons of the pyloric central pattern generator (CPG) of adult *Carcinus maenas*, bought in a local fish store and kept in artificial sea water. Before the dissection, the crab was anesthetized by introducing it in the freezer for 20/30 minutes. The stomatogastric ganglion, dissected following the standard procedure, was attached to a Petri with Sylgard cold saline dissolution (13-15^{o}C kept by a microcontroller and always perfused) using pins. The saline had the following composition (in mM): 433 $NaCl$, 12 KCl, 12 $CaCl_2 \cdot 2\ H_2O$, 20 $MgCl_2 \cdot 6\ H_2O$, 10 $HEPES$, adjusted to pH 7.60 with 4M $NaOH$. Neurons were identified after desheathing the ganglion by their membrane potential waveforms and their corresponding spike times in nerves. Intracellular recordings were performed using 3 M KCl filled microelectrodes (50 $M\Omega$) and a DC amplifier (ELC-03M, NPI Electronic, Hauptstrasse, Tamm, Germany). For details on the preparation see [Elices et al., 2019].

2.2.3 Computational models

Various neuron and synapse models are already included in RTHybrid's models library (see Appendix B for a detailed description). At the time of writing this document, the neuron

models present in RTHybrid by default comprise:

- **Izhikevich, 2003**: model from [Izhikevich, 2003] that reproduces spiking and bursting behavior of known types of cortical neurons. It has two differential variables (x and y), four parameters (a, b, c and d) and the external current input (I_{ext}).
- **Hindmarsh and Rose, 1984**: model of a spiking and bursting neuron from [Hindmarsh and Rose, 1984]. It has three differential variables (x, y and z), two parameters (r and s) and the external current input (I_{ext}). The third equation dz/dt gives the model a great range of behavioural dynamics for x, the membrane potential, including a chaotic regime.
- **Rulkov, 2002**: spiking-bursting model from [Rulkov, 2002] that uses an iterated map. It has two variables (x and y), three parameters (α, σ and μ) and the external current input (I_{ext}).
- **Ghigliazza and Holmes, 2004**: spiking-bursting conductance-based model from [Ghigliazza and Holmes, 2004]. It has one variable (v) and several parameters related to its four different ionic currents: I_{Ca}, I_K, I_{KS} and I_L, plus the external current input (I_{ext}), the capacitance (Cm), and parameters δ and ϵ.
- **Wang, 1993**: membrane potential oscillations model from [Wang, 1993]. It has one variable (v) and various parameters related to its five different ionic currents: I_{Na}, I_{Nap}, I_K, I_{Ks} and I_L, plus the external current input (I_{ext}), the capacitance (Cm), and parameters σ, ρ, ϕ and τ_m.
- **Komendantov and Kononenko, 1996**: conductance-based model from [Komendantov and Kononenko, 1996] based on the neurons of a Helix Pomatia snail CPG. It has one variable (v) and several parameters, related to its eight different ionic currents: I_{Na}, I_K, I_B, $I_{Na(TTX)}$, $I_{K(TEA)}$, $I_{Na(V)}$, I_{Ca} and I_{Ca-Ca}, plus the external current input (I_{ext}) and the capacitance (Cm).
- **Nowotny et al, 2008**: two-compartments conductance-based model from [Nowotny et al., 2008] based on the LP neuron from the pyloric CPG of a *Panulirus interruptus* lobster. For each of the compartments it has numerous variables and parameters.

Synapse models already included in RTHybrid's library are:

- **Electrical**: synapse model following the equation $I = g * (V_{post} - V_{pre})$, where g is the synapse conductance (in μS), V_{post} the postsynaptic neuron potential (in mV) and V_{pre} the presynaptic neuron potential (in mV).
- **Golowasch et al, 1999**: graded chemical synapse model from [Golowasch et al., 1999]. This model includes two different dynamics, one fast and one slow.
- **Destexhe et al, 1994**: chemical synapse model from [Destexhe et al., 1994].
- **Greenberg and Manor, 2005**: graded chemical synapse model from [Greenberg and Manor, 2005]

	Computer 1			Computer 2		
μs	**Min**	**Max**	**Mean ± Std**	**Min**	**Max**	**Mean ± Std**
Measured by computer	99.5	100.5	100.0 ± 0.1	99.9	100.1	100 ± 0
Measured by oscilloscope	90.0	110.0	100 ± 3	90.0	110.0	100 ± 3

Table 2.2: Results of the internal clock precision tests for the computers described in Table 2.1, in microseconds. A $100\mu s$ signal, generated by the computers, was recorded with each computer and an external oscilloscope. Minimum, maximum, mean and standard deviation values of the recorded signal period are displayed at the table. When compared to the signal period measured by the computers, the oscilloscope registered a $\pm 10\mu s$ inaccuracy in the precision of the computers clocks.

2.3 Results

2.3.1 RTOS benchmarking and comparison

Computers internal clock validation

Tests were conducted to certify that the internal clocks of the systems specified in Table 2.1 were capable of working with the required microsecond precision. These consisted in three 10-second tests on each platform, generating a square-wave signal of period $100\mu s$, which was recorded with an external oscilloscope. When compared to the signal period measured by the computers, the oscilloscope registered a $\pm 10\mu s$ inaccuracy in the precision of the computers' clocks (see Table 2.2). This is an acceptable margin for our purposes with sampling rates for the hybrid circuit electrophysiology ranging from 10 to 20 kHz.

RTOS usability comparison

RTAI, Xenomai and Preempt-RT were studied and compared in terms of installation requirements, usability and user-friendliness. This analysis is summarized in Table 2.3.

	RTAI	**Xenomai**	**Preempt-RT**
Installation	Kernel patching	Kernel patching	- Kernel patching - Debian repositories
Programming	API for C language	Various APIs emulators, including POSIX	Standard POSIX code (works also without real-time)
Documentation	Scarce and old	- Up-to-date - Few examples but active mailing-list	Plenty: standard POSIX documentation
Support and maintenance	Discontinued	Currently active	Currently active (Linux Foundation project)

Table 2.3: Usability and accessibility characteristics for each real-time solution explored in this study.

RTAI installation requires to patch a vanilla Linux kernel with its patch and then compile it, which is a long and tricky operation, even for experimented users. The last version of the official installation guide dates from 2008 [Monteiro, 2008]. Utilization of the real-time functions

provided by the platform is done through its own API, which is very powerful and complete, but documentation and examples are scarce. The safest way to achieve real-time is implementing the programs as kernel space modules, which carries many impediments. User space real-time can be reached using the LXRT library, although its use is discouraged for non-senior RTAI programmers [Racciu and Mantegazza, 2006]. Currently, this tool is still maintained, but not regularly: version 4 last maintenance was in 2013 and in May 2017 version 5 was released, and patch 5.3 in May 2021.

Patching and compiling a Linux vanilla kernel is the only way of installing Xenomai and its developers provide an up-to-date guide[3]. Since Xenomai's main purpose is to offer an open-source alternative to proprietary RTOS, it includes different APIs, intended to emulate other environments and libraries, such as VxWorks, pSOS+ and even POSIX. All of them, as well as their own API, are accessible from user space. Complete and updated documentation is available for all APIs. Although the user community is not very large and there are not many examples to be found, there is a mailing-list where questions can be asked. The project is currently active (Xenomai 3 was released in October 2015) and it is maintained and updated frequently. An Ubuntu 16.04 distribution already patched with Xenomai 3.0.5 can be downloaded from our website[4].

Similarly to the dual-kernel implementations, the typical way to install Preempt-RT is by patching and compiling a vanilla kernel following the instructions that can be found at the official website[5], these being are significantly simpler than the ones for RTAI and Xenomai. An alternative possibility for Debian distributions is to install it from its repositories as any other package[6]. Despite the real-time patch, the system is still a normal Linux, so the standard POSIX library can be used and all its documentation is valid. In 2015 the project was transferred to The Linux Foundation, becoming the "official" Linux real-time solution.

RTOS benchmark analysis

RTAI, Xenomai and Preempt-RT were tested using the method described in 2.2.1. Each trial consisted in a five minute run of the test program under stress, with a frequency of 20kHz, and was repeated twice on every platform. We measured the maximum, minimum and mean latency values, as well as the jitter, and the results can be seen in Figure 2.7. Distribution of latency values during these tests is shown in Figure 2.8.

RTAI obtained the best performance scores, getting 3.65μs as maximum latency, even running on an older machine. Xenomai was not far from this performance, with a maximum latency of 5.66μs. Preempt-RT had slightly worse results, reaching a maximum latency of 15.94μs, but still acceptable for our purposes. The system without real-time is not reliable when millisecond precision or below is required, as it goes over the millisecond barrier in these tests (indicated in red in Figure 2.8). Nevertheless, as mentioned in section 2.2.1, Linux operating systems allow to isolate a processing core and bind a specific task to it. In this scenario, the task running over the isolated core will never be interrupted by any other users tasks, but system processes can still use this core. When this is done in RTAI or Xenomai it has little impact on their already good latency values, but we observe a remarkable improvement of latency values with both non real-time and Preempt-RT operating systems. Without a RTOS, it can be an useful tool in soft real-time environments.

[3] https://source.denx.de/Xenomai/xenomai/-/wikis/Installing_Xenomai_3

[4] Ubuntu 16.04 with Xenomai 3.0.5 Live CD/USB: http://arantxa.ii.uam.es/~gnb/rtubuntu-16.04.4-1.0-desktop-amd64.iso

[5] https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/preemptrt_setup

[6] https://packages.debian.org/search?searchon=names&keywords=linux-image-rt

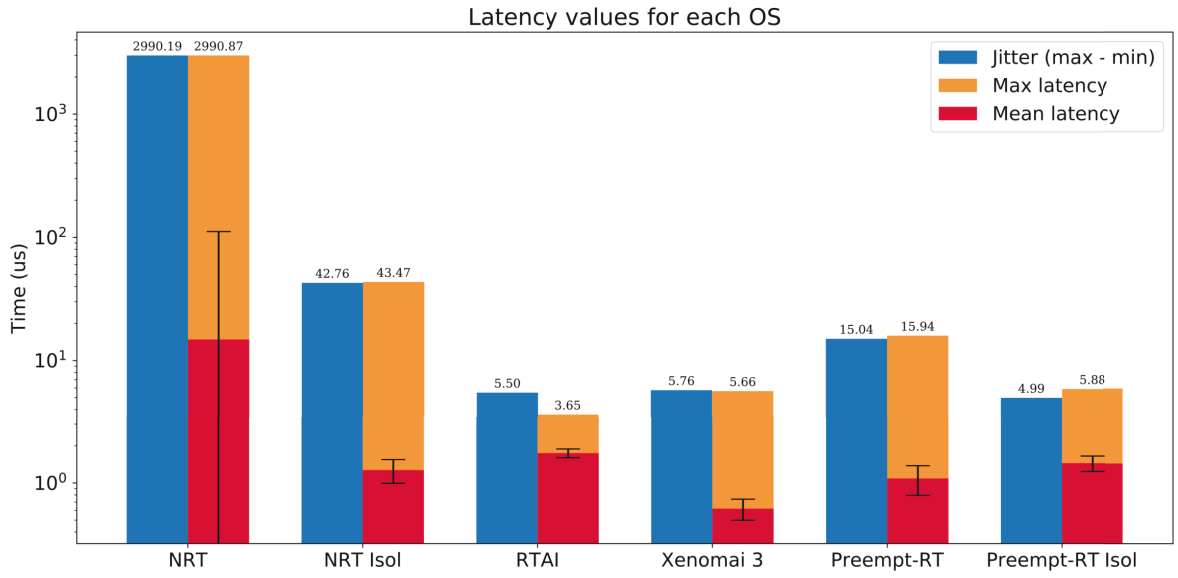

Figure 2.7: Summarized results of the real-time benchmarking tests for each OS: No real-time (NRT), No real-time with an isolated core (NRT Isol), RTAI, Xenomai 3, Preempt-RT and Preempt-RT with an isolated core (Preempt-RT Isol). Time axis is represented in log-scale. Blue bar represents the jitter, calculated as the difference between the maximum and minimum latency. Orange bar represents the maximum latency. Red bar is the mean latency, error bars indicate the standard deviation. Numbers on top of each bar correspond to the largest jitter and latency, respectively. For 20kHz trials the real-time constrain is $50\mu s$, which is only exceeded in this case by the OS with neither real-time capabilities nor an isolated core.

Unfortunately, regarding the analyzed RTOS, the better their performance is, the worse their usability and user-friendliness. As summarized in Table 2.3, RTAI is quite difficult to install and use, even for experimented users, while Preempt-RT is the most accessible, since there are not many differences with a normal GPOS. Due to this results, RTHybrid was developed to run over both Xenomai and Preempt-RT to balance performance and user-friendliness.

2.3.2 RTHybrid implementation

RTHybrid is a free and open-source software application. It has been implemented as a stand-alone program to facilitate its installation, configuration and use by any kind of user. It runs over any Linux environment, including those patched with Preempt-RT or Xenomai 3 solutions to enable hard real-time capacities. It is written in C/C++ language and compiled using GCC 6.3. It incorporates an intuitive and user-friendly graphical user interface (GUI), designed using Qt 5.10 framework and compiled using QMake 3.0, from where the experiments can be configured and launched (Figure 2.9). It can also be executed from command-line in non-GUI mode, loading the experiment settings from an XML file, allowing the users to run scripts that launch a consecutive series of experiments automatically. All the data from the hybrid circuit experiments, including recordings of both neurons membrane potentials and synaptic current values as well as the real-time system latencies, are saved into plain text files during the procedure.

This program employs an architecture consisting of three processing threads to ensure an optimal performance in a hard real-time scenario (Figure 2.10). The main thread starts the application, both when using the graphical interface mode or the command-line one. This thread is in charge of collecting all of the experiment various parameters entered by the user,

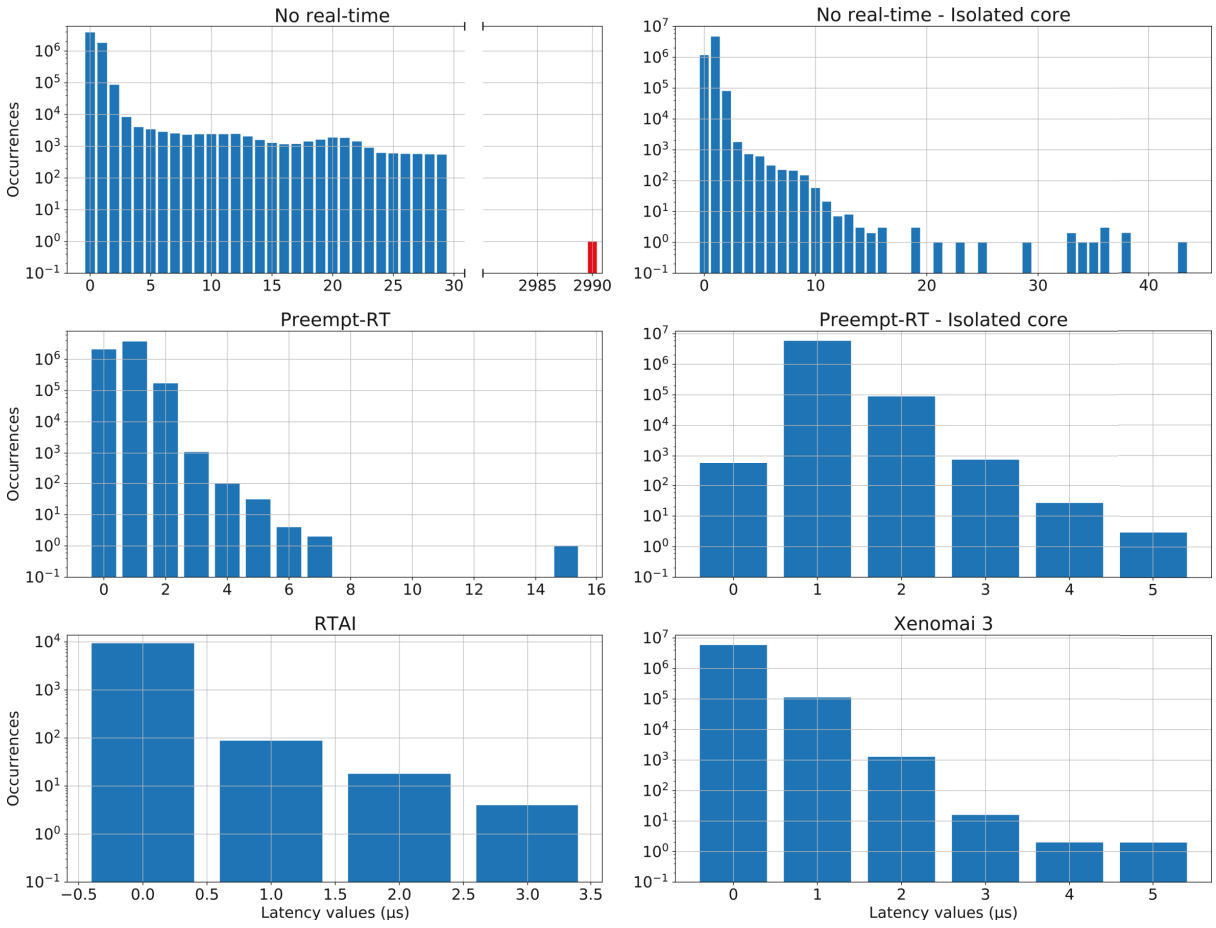

Figure 2.8: Distribution of latency values for each operating system during the real-time benchmarking tests. Red bars correspond to latency values that have exceeded the $50\mu s$ limit established for 20kHz trials, which is the case for the "No real-time" scenario. Vertical axes are in log-scale. Note that the x-axis scale are different for each test. This analysis was performed on the same data than Figure 2.7.

either through the GUI or an XML file. Once this is done, it prepares the environment for the launching of the experiment in real-time, creating the files where all data will be stored and also two new processing threads and a message queue that will work as communication method between them. When both children threads have completed their corresponding tasks the parent thread joins them, frees all memory that had been used and finishes the experiment, remaining on standby until the next execution.

The first child thread, or real-time thread, is in command of the real-time interaction between the simulated and biological components. To achieve this its thread priority is set as the maximum available, which depending on the operating system (i.e., standard Linux, Preempt-RT or Xenomai 3.0) will determine how it is handled by the scheduler. When it is launched, this thread starts the communication with the data acquisition board using an API to work with Comedi open-source drivers, compatible with several National Instruments and other manufacturers boards[7]. Xenomai includes its own set of drivers derived from Comedi ones, called Analogy. When the real-time thread finishes its task, this connection is closed. Once all the necessary setup actions for the interaction has been completed then a periodic loop begins, containing the real-time task to be performed. Every iteration of this loop comprises the following

[7] https://www.comedi.org/hardware.html

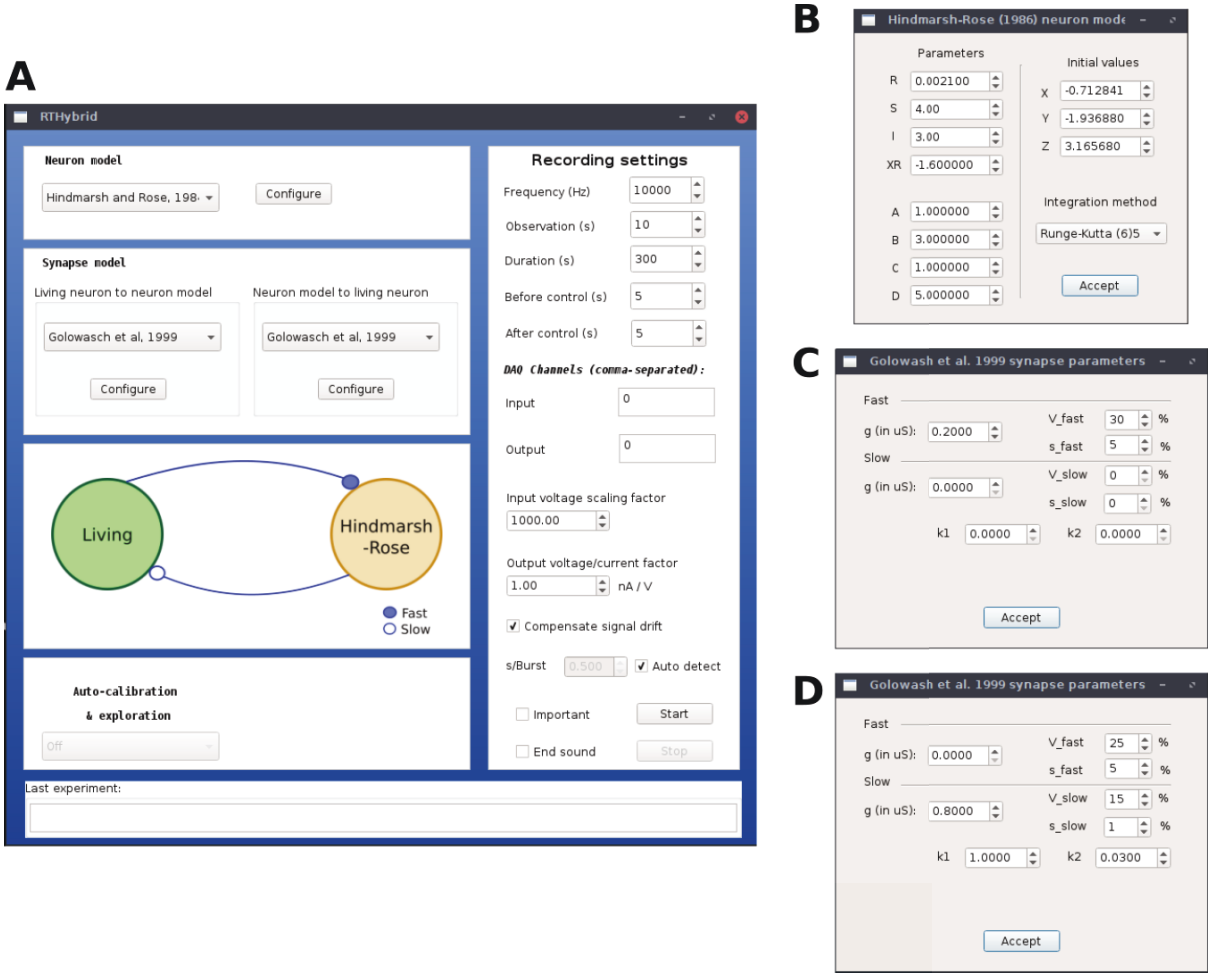

Figure 2.9: **A:** illustration of RTHybrid graphical user interface to build hybrid circuit interactions. Users can select neuron and synapse models, as well as their parameters, and the experiment settings, such as input/output DAQ channels, sampling frequency, duration, etc. **B:** prompt dialog box with the configuration for the selected neuron model, e.g., [Hindmarsh and Rose, 1984] model in this case. **C:** prompt dialog box with configuration for the living to model neuron synapse model selected, e.g., [Golowasch et al., 1999] model in this example. **D:** same as C but in the opposite direction.

steps:

1. Sleeping until it is time for the next interval of the real-time task.
2. Writing data to the DAQ board.
3. Sending data to the other child thread through the message queue.
4. Reading from the DAQ board.
5. Calculating the next point of the neuron model as well as the synapse models for the connections both from the living neuron to the model and vice versa.

In order for an optimal interaction to happen, it is necessary to carry out some calibration tasks to ensure that all models are working in the same spatial and temporal scales as the living neuron. Moreover, the living neuron signal may contain artifacts or experience drifting from its

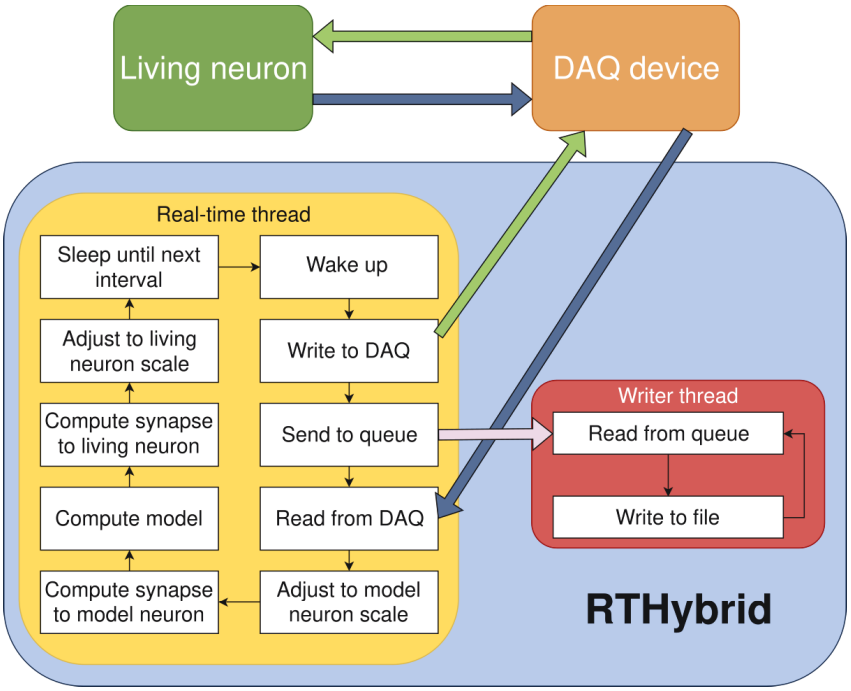

Figure 2.10: Diagram of RTHybrid architecture. The computer receives the membrane voltage from the living neurons through the DAQ device, computes the neurons and synapses models and sends current back to the biological cells. The main program process creates two sub-threads, *real-time thread* and *writer thread*, to manage the different tasks of the program. Both threads communicate through inter-process communication (IPC) message queues.

initial range during the experiment. These issues need to be dealt with in real-time. The user can determine the amount of time that the program must allocate for this calibration process, during which there is no interaction between both neurons. Likewise, observation periods can be set at the beginning and end of the experiment, during which the activity of both neurons is recorded but there is no interaction.

Figure 2.11 displays the mean time consumed by each operations performed inside the real-time loop for different neuron models, both over Preempt-RT and Xenomai 3.0. These results are also shown in detail Table 2.4. The models were run at a 10kHz frequency for five minutes (300 seconds), i.e., each model test contained 3 million intervals of $100\mu s$ duration. Neuron models were bidirectionally connected through a graded chemical synapse model to a hardware-implemented Hindmarsh-Rose model that generated bursting activity at the same characteristic rate of a pyloric CPG cell [Pinto et al., 2000]. Synaptic conductances were set to $g = 0.02\mu S$ and all other parameters from neuron and synapse models were fixed to produce bursting behaviour. Burst duration for the models was set at one second per burst. The computer used for these tests was the one referred as Computer 1 in Table 2.1, with both Preempt-RT and Xenomai 3 and the same DAQ device and board described in section 2.2.2.

The task performed by the second child thread, or writer thread, is to write down into a file all the data recorded during the experiment. Due to the technical limitations of a hard real-time operating system, there are several actions that can not be carried out inside the real-time periodic task without causing a system failure. One of such operations is printing data, either to disk or text console, so a secondary thread is required. This second child thread has not the maximum priority since it does not need to run in real-time, so it will print the data that it receives through the message queue just when the scheduler gives it enough resources.

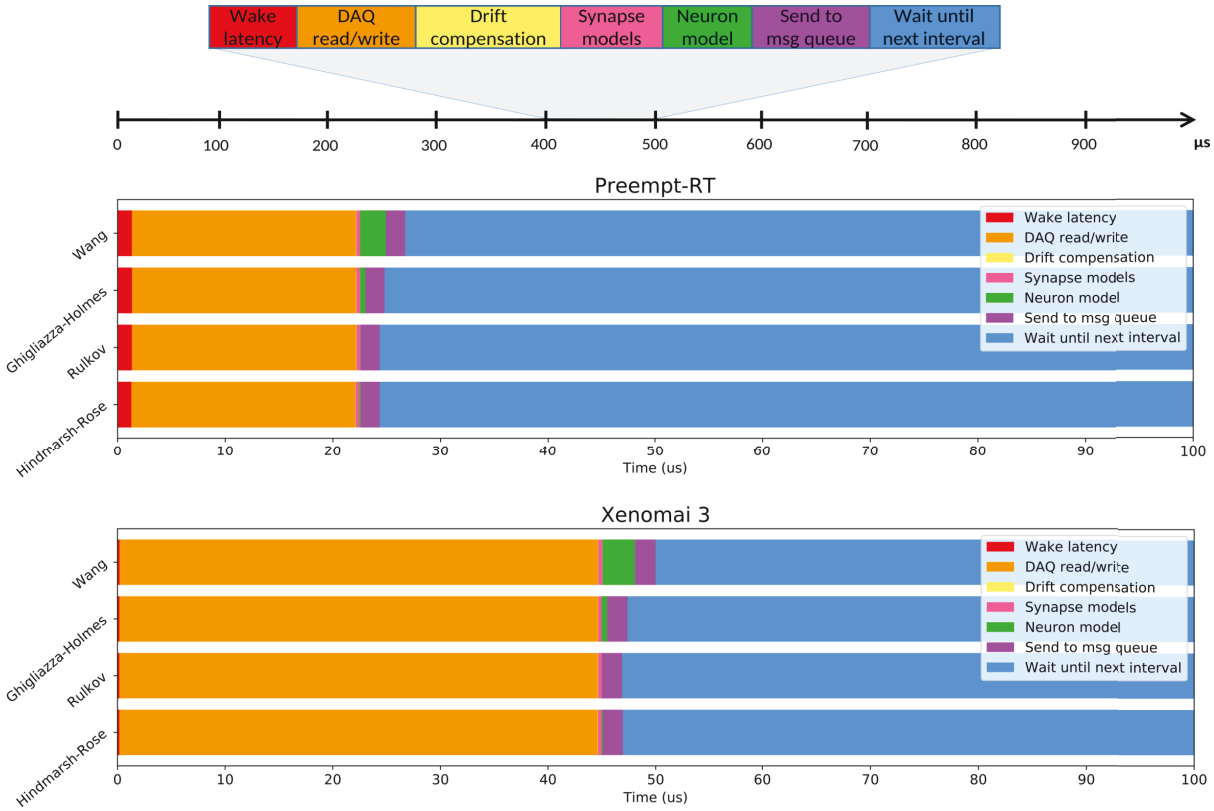

Figure 2.11: Average time usage for each operation over RTHybrid real-time intervals when different neuron models are computed. Top panel illustrates the operations that are performed inside each iteration of the loop executed at the real-time thread of RTHybrid: process wakes up, interacts with the DAQ device, performs (if activated) the drift compensation, computes the synapse models, calculates a new point (or points) of the neuron model, sends the message with data for the writer thread to the queue and sleeps until the expected beginning of the next interval. Middle and bottom panels show the time consumed, on average, by each of the previously described operations within a $100\mu s$ interval when different neuron models are computed, on Preempt-RT and Xenomai 3, respectively.

2.3.3 Model library

RTHybrid contains a model library that includes by default various neuron and synapse models (see Appendix B). User can also incorporate their own models into it. Several numerical integration methods for differential-equations systems are also part of the library, including Euler, Heun, Order 4 Runge-Kutta [Press et al., 1988] and (6)5 Runge-Kutta [Hull et al., 1972].

Included neuron models have been selected for their rich intrinsic dynamics, their suitability to build hybrid circuits and their capacity to be computed in a hard real-time scenario. They cover a wide range of types, from simple models with a low number of equations and also low computational cost [Izhikevich, 2003, Rulkov, 2002, Hindmarsh and Rose, 1984] to conductance-based ones which are more complex and realistic [Ghigliazza and Holmes, 2004, Wang, 1993, Komendantov and Kononenko, 1996, Nowotny et al., 2008]. In order to assemble a hybrid circuit, it is necessary that neuron models work in similar spatial and temporal scales than the living neurons.

This constitutes a critical feature to implement effective interactions between living and model neurons and, therefore, automatic calibration algorithms were incorporated into RTHy-

	Preempt-RT		Xenomai 3.0	
	Mean ± Std(μs)	**Max (μs)**	**Mean ± Std(μs)**	**Max (μs)**
Wake latency	1.3 ± 0.1	64.649	0.18 ± 0.05	19.844
DAQ read/write	20.8 ± 0.3	55.807	44.5 ± 0.3	49.973
Drift compensation	0.022 ± 0.006	8.654	0.022 ± 0.007	0.711
Synapse models	0.35 ± 0.04	12.486	0.4± 0.1	12.740
Send to queue	1.77 ± 0.09	37.449	1.88 ± 0.08	5.599
Neuron models				
Hindmarsh-Rose	0.10 ± 0.01	5.810	0.099 ± 0.005	0.832
Rulkov	0.03 ± 0.01	9.319	0.021 ± 0.008	0.709
Ghigliazza-Holmes	0.49 ± 0.04	4.937	0.51 ± 0.05	6.159
Wang	2.4 ± 0.2	136.114	3 ± 1	122.707

Table 2.4: Duration analysis for each operation performed inside the real-time cycle described in Figure 2.11 on every iteration. Mean and maximum times are shown in microseconds. The use of RTOS constrains the variability of these times, with Xenomai 3 being more efficient at this than Preempt-RT. Duration of each model computation differs greatly from the others due to their distinct mathematical descriptions.

brid in order to perform calibrations during the experiments [Reyes-Sanchez et al., 2020]. The adjustment in the spatial range is carried out by means of an escalation in the presynaptic membrane potential amplitude received by the synapses. This way, synapse models always calculate a current value in the working scale of the postsynaptic neuron so it has the expected effect on its behaviour without modifying the activity of either the neuron model or the living neuron.

The adjustment in the temporal range consists in tuning the neuron model functioning so its bursts have an specific duration, usually one that matches the biological neuron ones. For example, if the living neuron generates one-second long bursts and the experiment is being performed using a 10kHz sampling frequency then, in computational terms, the model's bursts should have 10000 points. Some neuron models, such as Rulkov's one, generate their dynamics using very few points (see Equations B.6 and B.7), so it is necessary to interpolate until the proper number is reached. On the other hand, most models are defined by differential equations so the number of points that they use to produce an event is determined by their integration step. Therefore, in this case, the calibration consists in choosing the highest integration step possible that produces a high enough-resolution membrane potential activity and then downsampling it if still has too many points, until reaching the appropriate amount (Figure 2.12). RTHybrid performs this calibration automatically and can fix the bursting period to a value determined by the user or adapt it to follow the living neuron.

All neuron models follow a similar architecture in their implementation, specially those based on the differential equations model by Hodgkin and Huxley. Although writing the code for new models is a straightforward and mechanical task, it is also difficult to track mistakes in the process, specially when there are several equations and parameters involved. Thus, an auxiliary Python application was also developed to automatically generate all required C code for a neuron model from a plain text file containing the equations[8].

This model library also include various synapse models, both electrical and chemical (see Appendix B). Since synapse models represent dynamics which can be very different from one

[8] https://github.com/GNB-UAM/rthybrid-neuron-model-generator

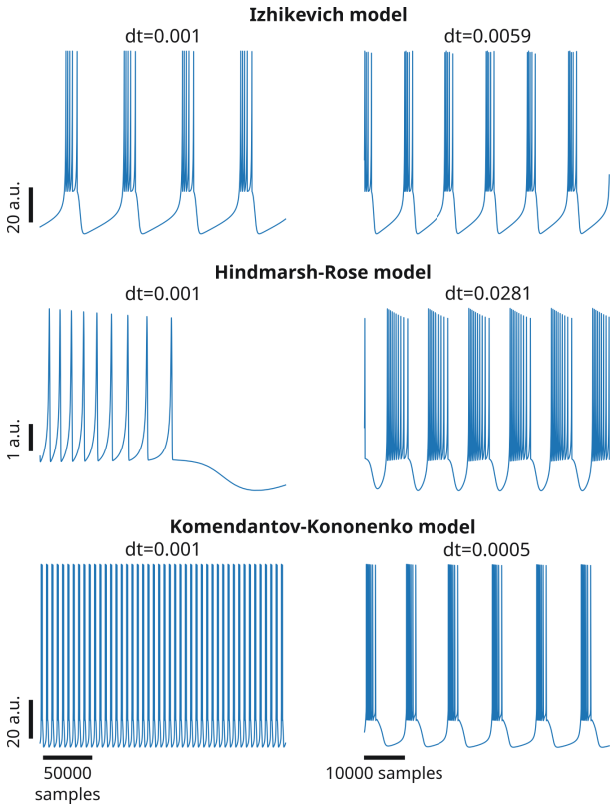

Figure 2.12: Left column: bursting behaviour for three different neuron models, [Izhikevich, 2003], [Hindmarsh and Rose, 1984] and [Komendantov and Kononenko, 1996], using the same integration step (dt=0.001) and with no downsampling. Each model produces bursts with a completely different temporal resolution. **Right column:** same neuron models calibrated to generate bursting activity with a 1 second period at a 10kHz sampling rate (i.e., 10000 samples per cycle). This was achieved by adapting the integration step to values that produced around 10000 samples per burst and no downsampling was necessary in this case.

another, their models implementations are also much more heterogeneous than neuron ones. Users can also implement and add their own synapse models to RTHybrid.

2.3.4 Validation in a experimental environment

Proper performance of RTHybrid neuron and synapse models was tested building hybrid circuits as detailed in section 2.2.2. LP living neurons from the pyloric CPG were bidirectionally connected through chemical graded synapse models with neuron models. Four trials per model were conducted, each of them five minutes long, with one a minute long control period before and after the hybrid circuit interaction. The first two trials were performed with a sampling frequency of 10kHz, thus cycle interval duration was $100\mu s$. The remaining two trials were run at 20kHz, and the interval was $50\mu s$ long. Any latency value exceeding that limit was considered a real-time failure. All four trials per model were repeated both in Preempt-RT and Xenomai 3, without core isolation.

Connections in these hybrid circuits mimicked graded chemical synapses with fast and slow dynamics [Golowasch et al., 1999]. The connection from the model to the living neuron was built with a slow synapse. A fast graded synapse was used for the connection from the living neuron to the model. Our target was to achieve rhythmic antiphase behaviour between the neurons so we set inhibitory synapses in both directions. The slow synapse had a conductance of $g = 0.2\mu S$, $V_{th} = 15\%$ and $s = 1\%$ of the maximum amplitude range and kinetic parameters $k_1 = 14.0$ and $k_2 = 4.0$, except when using Hindmarsh-Rose model that the conductance was $g = 0.1\mu S$. The fast synapse had $V_{th} = 50\%$ and $s = 5\%$ of the maximum amplitude range and different conductances depending on the neuron model at use: for Izhikevich and Ghigliazza-Holmes models the

conductance was $g = 0.8\mu S$, for Hindmarsh-Rose model it was $g = 1.0\mu S$ and for Rulkov was $g = 0.2\mu S$. Input voltage scaling factor was set to 100 and output current/voltage conversion factor to $10nA/V$. Neuron model parameters were set to produce bursting behaviour and the duration of each burst was set to automatically match the living neuron activity. Signal amplitude and temporal scaling was automatically performed by RTHybrid calibration algorithms [Reyes-Sanchez et al., 2020]. Validation tests are shown in Figures 2.13 and 2.14.

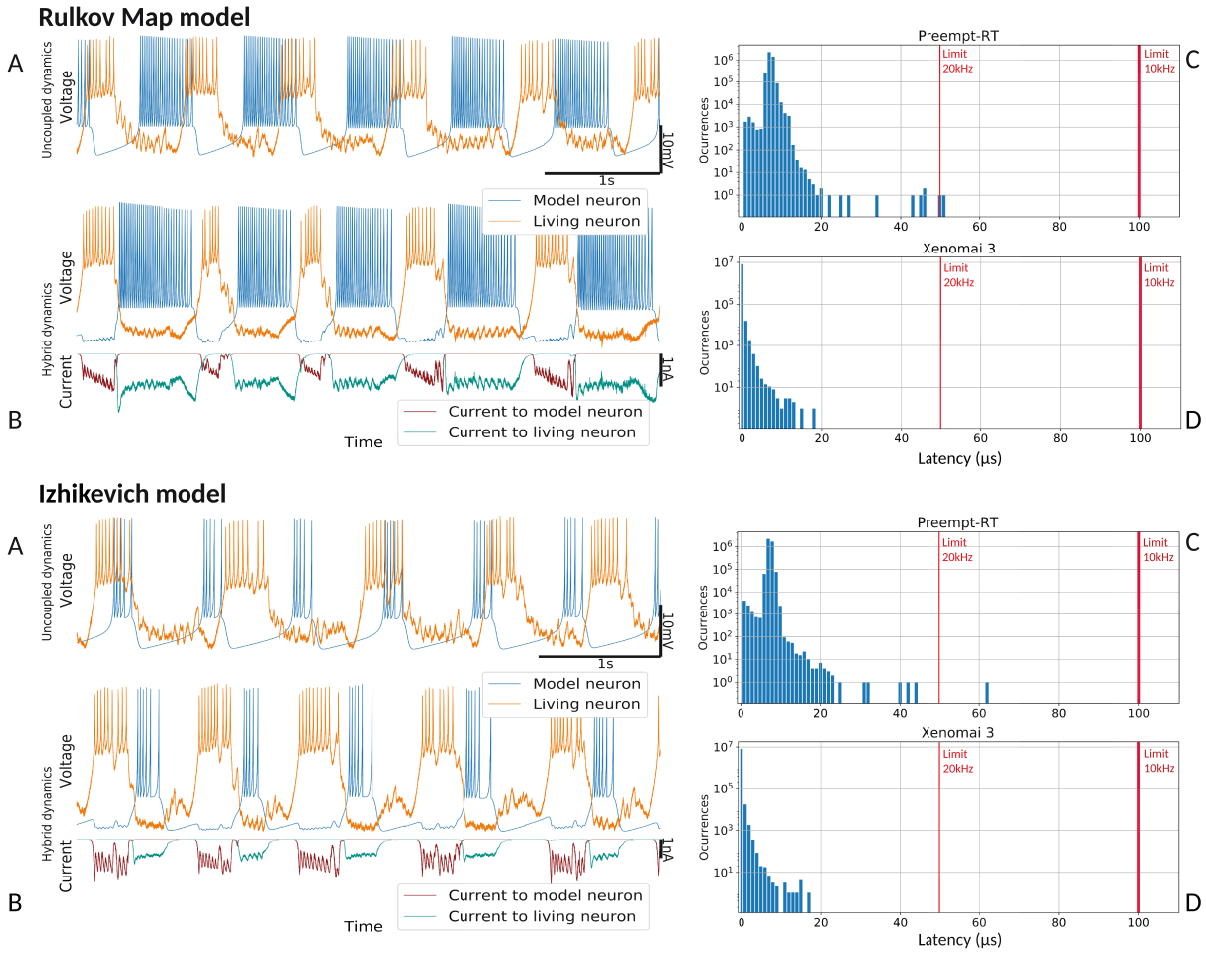

Figure 2.13: Hybrid circuit built with an LP pyloric CPG neuron and a simulated neuron using models from [Rulkov, 2002] and [Izhikevich, 2003]. For both models, each of the four panels represent: A) membrane potential of the uncoupled neurons before any interaction (adapted to the living neuron range), showing their independent dynamics B) membrane potential and synaptic currents (adapted to the living neuron range) during the hybrid interaction, showing their coupled antiphase dynamics C) Preempt-RT latency values D) Xenomai 3 latency values. On the latency figures, left-most red line represents the 20kHz limit and the right-most, the 10kHz limit.

Figure 2.13 shows the results of the validation tests for the Rulkov and Izhikevich models, and Figure 2.14 displays the results with the Hindmarsh-Rose and Ghigliazza-Holmes models. For each model, sorted by increasing complexity, there are four panels displaying different information about the hybrid interaction. The worst performance trials for each model, understood as those with higher latency values, were selected for the analysis. Recorded membrane potential when both living and model neurons are uncoupled is displayed in panel A. Voltage is scaled to the living neuron range, showing the independent behaviour of each neuron's bursting activity

when there is no interaction. Membrane potential during the hybrid interaction along with the synaptic current injected in both directions is shown in panel B, portraying the robust antiphase rhythm achieved due to the bidirectional inhibitory graded chemical connectivity. Panel C represents Preempt-RT test latency values, showing that in all cases the $100\mu s$ limit established for 10kHz trials, represented by the right-most red vertical line, was fulfilled. However, this RTOS was not able to keep 20kHz constrains, indicated by the left-most red vertical line, during these experiments. This was not the case for Xenomai 3, portrayed at panel D, whose latency values were far under the $50\mu s$ barrier set during 20kHz trials. Due to this outcome, the results displayed correspond to Preempt-RT 10kHz and Xenomai 20kHz trials.

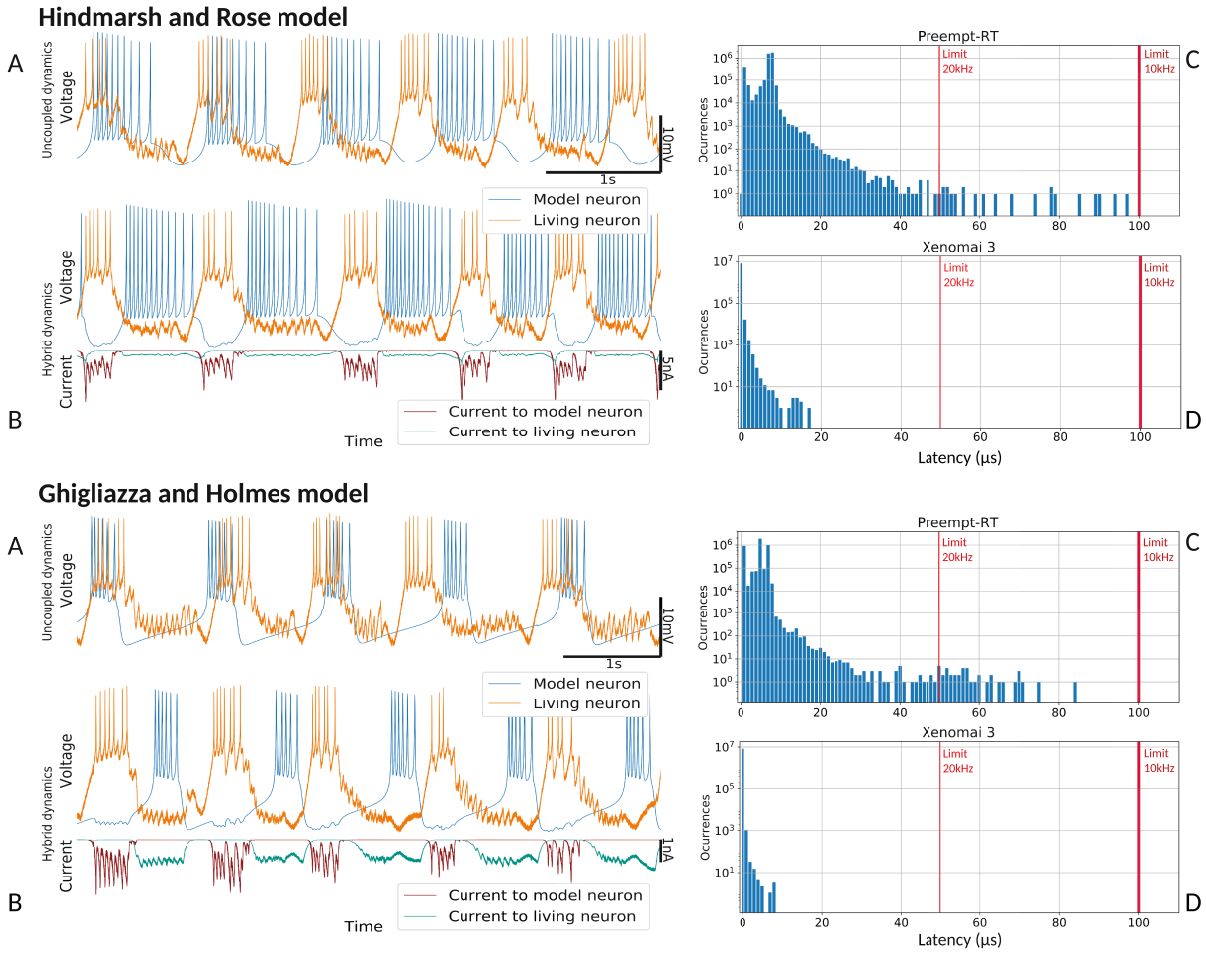

Figure 2.14: Hybrid circuit built with an LP pyloric CPG neuron and a simulated neuron using models from [Hindmarsh and Rose, 1984] and [Ghigliazza and Holmes, 2004]. For both models, each of the four panels represent: A) membrane potential of the uncoupled neurons before any interaction (adapted to the living neuron range), showing their independent dynamics B) membrane potential and synaptic currents (adapted to the living neuron range) during the hybrid interaction, showing their coupled antiphase dynamics C) Preempt-RT latency values D) Xenomai 3 latency values. On the latency figures, left-most red line represents the 20kHz limit and the right-most, the 10kHz limit.

2.3.5 RTHybrid for RTXI

Additionally to its implementation as a stand-alone program, another version of RTHybrid was developed. RTXI is an open-source hard real-time data acquisition and control application that works over Xenomai [Patel et al., 2017]. Its design as a block-based workspace where different plugins can be added allows the user to create diverse experimental configurations in a flexible way. For this reason RTHybrid, has also been developed as a set of plugins compatible with the RTXI platform to promote the standardization and dissemination of hybrid circuits and real-time closed-loop technologies in general[9]. These plugins include modules to handle temporal and spatial calibration as well as various neuron and synapse models (Figure 2.15). This variant of the software offers a higher degree of flexibility when it comes to designing hybrid circuits, but also requires of more time and knowledge to set them up.

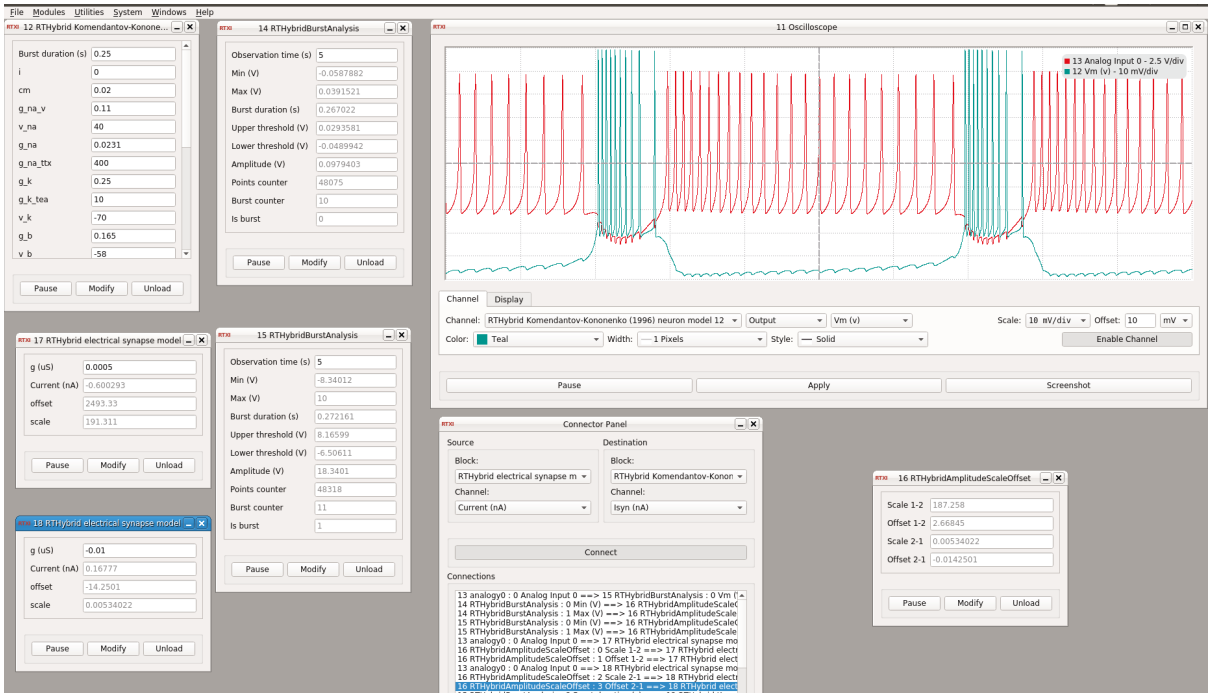

Figure 2.15: Example of a RTXI workspace with RTHybrid plugins to build a real-time hybrid circuit. In this case, a [Komendantov and Kononenko, 1996] model (green trace) is connected to an external electronic neuron (red trace) using a bidirectional electrical synapse. Note the difference in the membrane potential amplitude scales for both neurons. RTHybrid provides modules to simulate various neuron and synapse models, as well as to perform all the necessary calibration operations, both temporal and spatial, to create effective hybrid circuits and closed-loop interactions.

2.4 Discussion

Characterization and control of neural systems dynamics, as well as experimental protocol automation, can largely benefit from the use of closed-loop techniques and specifically from hybrid circuits built by connecting model neurons and synapses to living cells. RTHybrid provides a neuron and synapse model library aimed to build hybrid circuits in an easy and user-friendly manner with large flexibility provided by a real-time software approach that takes advantage of

[9] https://github.com/GNB-UAM/rthybrid-for-rtxi

the computational and memory capacity of modern computers. Developed for Linux, this program is open-source and can be downloaded for free from www.github.com/GNB-UAM/RTHybrid, where installation, user manuals and examples can also be found. RTHybrid has a simple GUI to design and configure the hybrid experiments. This tool also incorporates a command-line mode where configuration XML files can be loaded, a useful feature for experiment automation and scripting for activity-dependent exploration of neural dynamics. Beyond a standard Comedi compatible DAQ board, no additional hardware requirements are needed. Thus, standard computers used for data acquisition in any lab can run this software, for example in a dual-boot configuration with another hard-drive with a GPOS setup, e.g., Windows, to keep existing protocol configurations.

Temporal precision requirements for closed-loop interactions in RTHybrid are fulfilled by using hard real-time software technology. An extensive analysis of available RTOS was conducted to select the most suitable platforms to implement RTHybrid. The software has been developed to run over Preempt-RT and Xenomai 3 frameworks for Linux due to their balance between performance and accessibility. RTHybrid model library includes a wide variety of neuron models: from computationally-inexpensive paradigms, such as [Rulkov, 2002, Izhikevich, 2003, Hindmarsh and Rose, 1984], to realistic conductance-based Hodgkin-Huxley type [Komendantov and Kononenko, 1996, Nowotny et al., 2008]. The library also contains several synapse models: from simple gap junctions implementations to configurable chemical synapses [Destexhe et al., 1994, Golowasch et al., 1999]. All of them are adapted to work under hard real-time restrictions and additional neuron, network and synapse models can be easily added using C language. Moreover, calibration algorithms are integrated in the library to automate the adaptation of model amplitude and time scales to the living neuron behavioural range [Reyes-Sanchez et al., 2020]. Core isolation can also be employed as validated in the tests reported in this chapter.

Many researchers and laboratories overlook closed-loop techniques despite their advantages due to the difficulties in the installation and use of the required technology. With RTHybrid, we aim to encourage the use of open-source, standardized and user-friendly real-time software tools, available in different platforms, to facilitate the implementation of closed-loop experimentation in neuroscience research.

2.5 Future work

RTHybrid current version includes a wide collection of neuron and synapse models, but this can be of course expanded. The automatic model generator is a very useful tool for this purpose, providing both developers and users an easy way of extending the library. This application can be improved by including more mathematical expressions into its parsing algorithm, such as trigonometric ones, enabling its use with a wider range of models. Additionally, it can also be expanded to also handle synapse models.

The stand-alone implementation of RTHybrid builds circuits by connecting one living neuron with one neuron model, but the number of both kinds of neurons involved in a hybrid circuit is only limited by the electrophysiological setup, on the biological side, and the computational power of the computer, for the models. Thus, the program could be extended to allow the constructions of hybrid circuits with any number of living and model neurons. The graphical user interface can be also improved to provide a simpler and more intuitive usability, including a graphical network design system with click-and-drag elements to easily allow the implementation of greater hybrid circuits. An online signal visualization system would also be an appreciated incorporation. Future development in parallelization and GPU computing will be considered to implement large scale network or highly-realistic biophysical models.

Finally, a similar approach as the one followed with RTXI could be implemented to adapt

RTHybrid to other recording and acquisition frameworks in order to make it as standardized and spread in the experimental neuroscience community as possible.

Models currently included in the RTHybrid library are suitable for a wide variety of hybrid circuit experiments implemented using dynamic clamp. Beyond electrophysiological protocols, RTHybrid can also be easily generalized to drive open- and closed-loop interactions in optogenetics and drug microinjection paradigms.

3

FLC-Hybrot: functional living circuit dynamics driving the locomotion of a hybrid robot with closed-loop sensory feedback

3.1 Introduction

This second chapter details the implementation of a functional living circuit hybrot (FLC-Hybrot), a hybrid robot formed by a mechanical body and a living CPG that fully controls its locomotion using the dynamical coordination present in the neural activity. At the same time, the robot's sensors send feedback to the living neurons, modifying the activity of the circuit while preserving the coordination of the CPG neural dynamics. This closed-loop interaction is conducted in soft real-time employing some of the tools described in the previous chapter. We performed experiments with central pattern generators of invertebrates to validate the correct locomotion of the hybrot when modulated by cycle-by-cycle dynamical invariants from the CPG rhythm. Dynamical invariants are robust relationships between some of the cycle-by-cycle intervals that build a sequence, which can underlie motor coordination rules in CPGs. We propose FLC-Hybrots as a useful tool to validate dynamical principles in neuroscience. The approach described in this chapter can be easily generalized for a wide variety of hybrots in different animal models. All the developed tools are available online at https://github.com/GNB-UAM/FLC-Hybrot.

Following the scheme of the previous section, this chapter is divided in an introduction, were the motivation and state of the art are explained, a materials and methods section, with the detailed description of the implementation of the project, and a results section where the outcome of the experiments is analyzed. Lastly, the results and future impact of the project are discussed.

3.1.1 Motivation and state of the art

There exists a close relationship between the fields of biology and neuroscience with robotics. For starters, biological mechanisms have been, and still are nowadays, an inexhaustible source of inspiration for the design of physical components, behavioural strategies and even learning and

artificial intelligence algorithms to be employed in the construction of mechanical automatons. This biomimetic approach has resulted, for example, in a myriad of animal-shaped robotic limbs or even full bodies [Crespi et al., 2005, Ijspeert et al., 2007, Saranli et al., 2001, Keennon et al., 2012, Kim et al., 2008, Chen et al., 2012, Roderick et al., 2021], as well as in the development of problem-solving algorithms based on biological behavioural patterns, like those implemented in robots that perform search tasks following collaborative strategies similar to the ones used by some insects and other organisms [Garcia-Saura et al., 2014, Wasilewski et al., 2021].

Moreover, machine learning algorithms that utilize artificial neural networks are of course inspired by their biological counterparts [McCulloch and Pitts, 1943, Hebb, 1949, Sejnowski, 2020, George et al., 2020] and currently constitute some of the most employed and widespread tools in the world. This includes their implementation in several complex intelligence and sensory perception systems for robots [Pomerleau, 1997, Floreano and Mondada, 2014, Lindsay, 2021, Montero et al., 2015]. Many robotic locomotion models have been drawn upon biological neural circuits, being CPGs some of the most extended ones due to their role as generators of robust rhythmic patterns, which animals use to coordinate motor functions such as walking, breathing or heart-beating [Garcia-Saura, 2015, Herrero-Carrón et al., 2011, Crespi et al., 2013, Dzeladini et al., 2017, Ayers and Witting, 2006, Ijspeert, 2008].

On the other hand, electronic and artificial elements can be used to alter biological systems, creating entities usually known as cyborgs (cybernetic organisms). The most apparent example of this practice is bionics, that consists in the use of robotic implants and prostheses to enhance the capabilities of a biological body, such as cochlear implants utilized in people with hearing impairment [Svirsky, 2017] or prostheses aimed to replace missing limbs [Kyberd et al., 2001, Grimmer and Seyfarth, 2014, Wang et al., 2022b]. This area of work also includes neural implants which are introduced into the subject brain and can be employed for numerous applications, from moving external objects like a wheelchair or a prosthesis just by thought [Warwick et al., 2003] to deep brain stimulation, which has therapeutic purposes for patients with Parkinson and other conditions [Benabid et al., 2009, Frizon et al., 2020].

But this exchange between biology and robotics can be taken even further than biomimetics and cyborgs. An example of this would be biohybrid robotics, which utilizes a combination of biological and artificial materials to create structures for robots, which are then controlled by computational means [Webster-Wood et al., 2017]. This enables the development of a wide range of innovations, particularly with the use of nanomaterials, from resource transportation systems built from unicellular organisms and even individual cells [Yasa et al., 2018, Striggow et al., 2020] to robots that employ animal muscle tissues to perform more complex tasks [Kim et al., 2007, Holley et al., 2016, Tanaka et al., 2019, Morimoto et al., 2018, Guix et al., 2021].

Following a similar concept but from the opposite point of view, hybrots are synthetic robots operated by real living neurons instead of a circuit model, algorithm or biological program [DeMarse et al., 2001, Potter et al., 2004]. Most of the existing hybrots are built using dissociated neurons grown on a cellular culture and whose activity is registered using multielectrode arrays [Potter et al., 2006, Novellino et al., 2007, Pizzi et al., 2009, Warwick et al., 2010, Li et al., 2016, Bisio et al., 2019, Sawada et al., 2022]. Only a few projects have utilized so far intact or semi-intact neural systems as the hybrot "brain" [Ando and Kanzaki, 2020], such as [Minegishi et al., 2012], which uses a visual interneuron of a living fly mounted on a robot to control its movement, with the feedback system being the animal's own sensory organs; or [Huang et al., 2019], which mounts a living silkworm moth into a wheeled robot while recording the activity from its neck motor neurons to coordinate the hybrot trajectory, again using the insect's own sensory system to introduce feedback.

A closed-loop interaction between the robotic components and the neurons is typically required when constructing a hybrot: neurons modulate the physical body functioning and, at the

same time, receive information from the surrounding world according to the robot situation in it. In most hybrot cases, the feedback received by the neurons comes from the animal's own sensory systems (visual, olfactory, etc.), but stimuli can also be artificially generated and directed to specific neural elements to produce more precise responses. Hybrots are an useful tool to study specific neural circuits dynamics and how they can be applied in task automation, as well as to formulate new robotic paradigms.

The aim of this project is to design and implement FLC-Hybrot, a hybrot controlled by the neural activity of a living and functional pyloric CPG, to validate the importance of the balance between robustness and flexibility in this circuit, as provided by cycle-by-cycle dynamical invariants present in its dynamics. With this idea in mind we built an hexapod robot whose legs performed oscillatory motions to move forward. The period and amplitude of these oscillations was determined by the sequential cycle-by-cycle activity of the CPG neurons. At the same time, the hybrot included a light-sensor and sent feedback information regarding the light conditions around it back to the neural circuit in the form of electrical current. These stimuli modified the behaviour of the cells, resulting in a change of the hybrot locomotion, preserving always the required motor coordination, and therefore forming a real-time closed-loop interaction among living and electronic components. The goal of the FLC-Hybrot is to demonstrate that a dynamical principle of the functional living circuit can be used to coordinate the locomotion with sensory feedback from the robot. In our case, we use the presence of dynamical invariants in the form of robust relationships between the time intervals that build the cycle-by-cycle activity of the CPG, which are sustained under any circumstance even when there are external inputs to the circuit, to modulate the behavior of the robot.

3.1.2 Dynamical invariants

As we mentioned in the previous chapter of this thesis, central pattern generators are neural circuits that produce very robust rhythmic sequences which are utilized by animals to coordinate motor functions (see Section 2.1.2). In the specific case of the *Carcinus maenas* pyloric CPG, this circuit presents a triphasic sequence formed by the bursting activity of three types of neurons: LP, PY and PD (Figure 2.3). Each cycle of this rhythm can be characterized in terms of the duration of the temporal intervals that take place between the beginning and end of each neuron bursts and resting periods (or hyperpolarizations). To define the start of each cycle we can choose the beginning of one of the neurons' burst, hence its instantaneous cycle period is equal to the time elapsed between the first spikes of each of that neuron's bursts (Figure 3.1).

The duration of these intervals on each cycle of a living CPG activity can be highly variable, both during the same experimental session and among different subjects. This large variability makes the CPG rhythm very flexible and capable of adapting to changes while maintaining the robust triphasic sequence at all times (see Figure 3.2). The linear relation between the duration of specific intervals is preserved cycle-by-cycle, regardless of how irregular these intervals may be. These relations are known as dynamical invariants [Elices et al., 2019]. The circuit effective coordination arises from the balance between the robustness of the sequence and the flexibility of these intervals.

In particular, of all the possible combinations of these cycle-by-cycle time intervals, robust dynamical invariants only emerge in two cases. The first of them is between the cycle period and the LPPD interval, which is the distance from the start of LP burst and the start of the next PD burst. The second is among the cycle period and the LPPD delay, defined as the time between the end of one LP burst and the start of the next PD burst. This kind of linear relation is not robustly sustained for any other combination of two intervals along the experiments [Elices et al., 2019]. Those intervals that are not part of invariants do

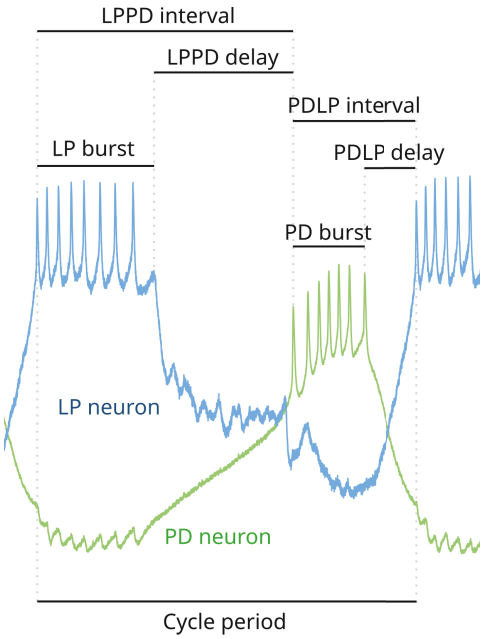

Figure 3.1: Example of the different time intervals in which a cycle of the *Carcinus maenas* pyloric CPG robust rhythm can be described using the LP and PD neurons as reference. In this case, each cycle starts with the first spike of the LP neuron.

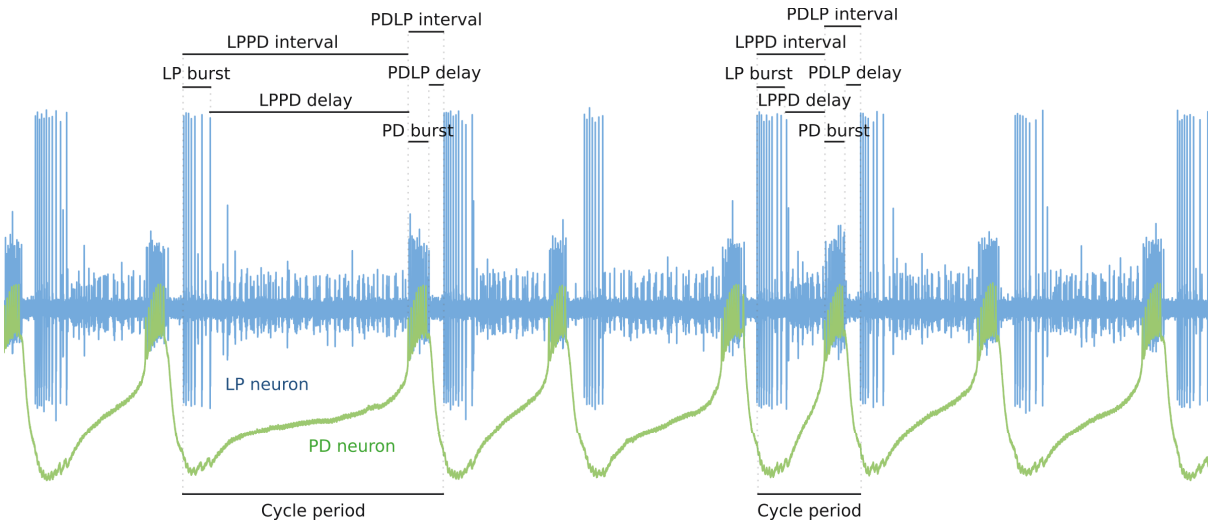

Figure 3.2: Example of the different time intervals in a sequence of cycles from a *Carcinus maenas* pyloric CPG rhythm. Cycles evolve and may have different duration. The sequence is always preserved and so do the dynamical invariants. Note how the LPPD interval and LPPD delay duration maintain a relation with the cycle period, changing when the later does. Intervals that are not part of invariants evolve independently, like the LP or PD bursts.

not have any apparent restriction in their variability. Thus, dynamical invariants can be interpreted as cycle-by-cycle rules for sequence coordination by linking the variability of only specific sequence intervals. The concept of dynamical invariants can be better understood by watching the video available at https://static-content.springer.com/esm/art%3A10.1038%2Fs41598-019-44953-2/MediaObjects/41598_2019_44953_MOESM2_ESM.mp4.

Dynamical invariants have been observed in control recordings with both regular and irregular activities, as well in experiments where an irregular rhythm was forced by the injection of chemical substances as ethanol (Figure 3.3). Nevertheless, and despite the great amount of CPG models developed for robotic control and other purposes, these relations do not emerge easily in

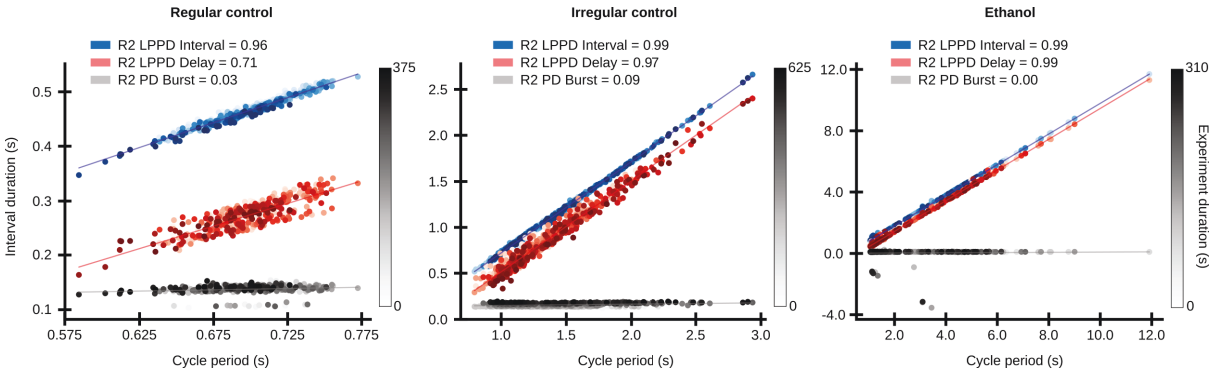

Figure 3.3: Example of dynamical invariants present in three different subjects with distinct pyloric CPG activities: regular control, irregular control and extreme irregularity caused by the injection of ethanol. Despite the diverse conditions of each experiment, two of the intervals, LPPD interval and LPPD delay, maintain a strong linear relation with the cycle period, while the PD burst never shows this relation. R^2 statistics for each linear regression is displayed at the top of each panel. Note the great differences in the cycle period duration across sessions, as well as in the duration of every interval. Data kindly provided by Irene Elices and Pablo Sánchez-Martín.

their computationally recreated activity since they lack the intrinsic variability present in their biological counterparts. The origin of the variability in the pyloric CPG intervals has not been fully assessed.

3.2 Materials and methods

3.2.1 Experimental design

The experimental paradigm designed in order to test the correct behaviour and locomotion of the FLC-Hybrot established a real-time closed-loop interaction between the robot and a living pyloric CPG (see Section 2.1.2). During the experiment, the FLC-Hybrot had to walk through a straight and flat lane, with no obstacles, along which there were interspersed sections with lights and shadows. Neural activity was recorded online from the living circuit and used to modulate the robot movement. At the same time, feedback current was injected into the living circuit when the robot's light-sensor detected a shadow (Figure 3.4).

Top panel of Figure 3.5 shows an illustration of the experimental setup for the FLC-Hybrot implementation with all its main components. A computer was employed to serve as nexus between both the robot and the biological elements and was in charge of executing a periodic real-time loop (Figure 3.5A), inside of which the following tasks were performed:

1. First, it read the CPG neurons electrophysiological signals from the DAQ device, that at the same came through the amplifiers (Figure 3.5B) after being recorded online from the living preparation (Figure 3.5C).

2. From this acquired data it extracted, online, the duration of the temporal intervals of the cycle-by-cycle CPG activity, which were then sent to the robot's controller via Bluetooth.

3. In the opposite direction, it received the robot's light-sensor data through the same Bluetooth connection.

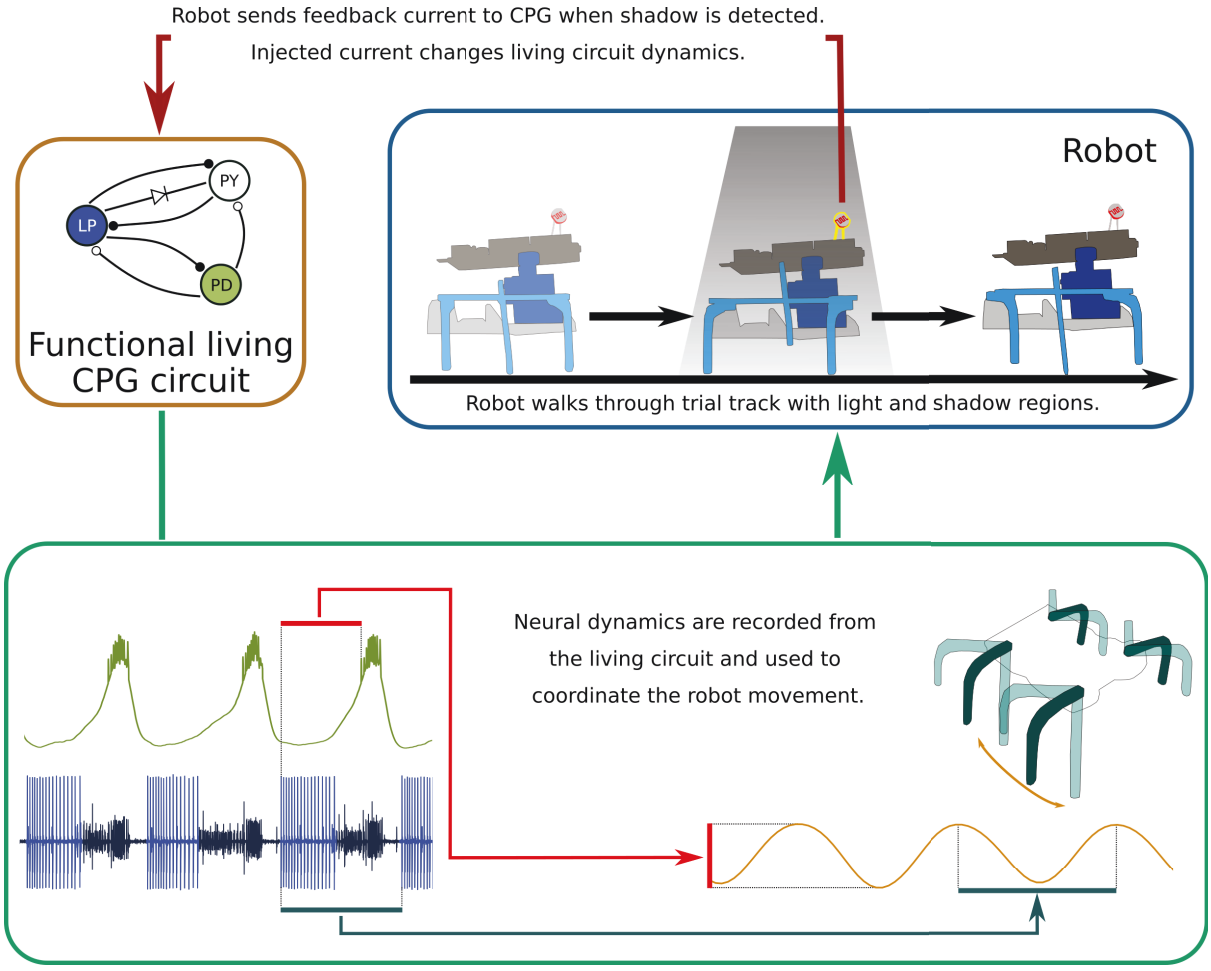

Figure 3.4: Representation of the FLC-Hybrot paradigm design. Neural dynamics from a functional living CPG circuit are recorded online and used to coordinate the robot movement in real-time. Activity from the PD neuron is recorded intracellularly (green trace), while the LP bursts are extracted from the nerve's extracellular signal (blue trace). This robot walks through a trial track with interspersed light and shadow regions. When the robot's light sensor detects that it is located under a shadow, it sends feedback current to the living circuit. CPG dynamics change as a reaction to the injected current, thus modifying the robotic locomotion. The following video shows an example of this experiment https://youtu.be/ny2dJGbG8lo.

4. If this data indicated that the robot was located under a shadow at that moment, then it sent feedback to the DAQ device in the form of electric current. This current was injected into the neurons, delivered by the amplifier, altering their behaviour.

All these operations were repeated while the robot traversed the trial track (Figure 3.5D). Robot's locomotion was determined by its legs oscillation, so we utilized video tracking tools to capture their movement during the experiment (Figure 3.5E). Two electrodes were used in order to measure the pyloric CPG activity. First, an extracellular electrode picked up all the surrounding neurons' activity. In this signal, the spikes and bursts of the LP neuron were clearly recognizable due to their larger amplitude. One of the PD neurons' membrane potential was recorded using an intracellular electrode. A second intracellular electrode was used to inject the feedback current into a PD neuron to implement the robotic sensory feedback (Figure 3.5F).

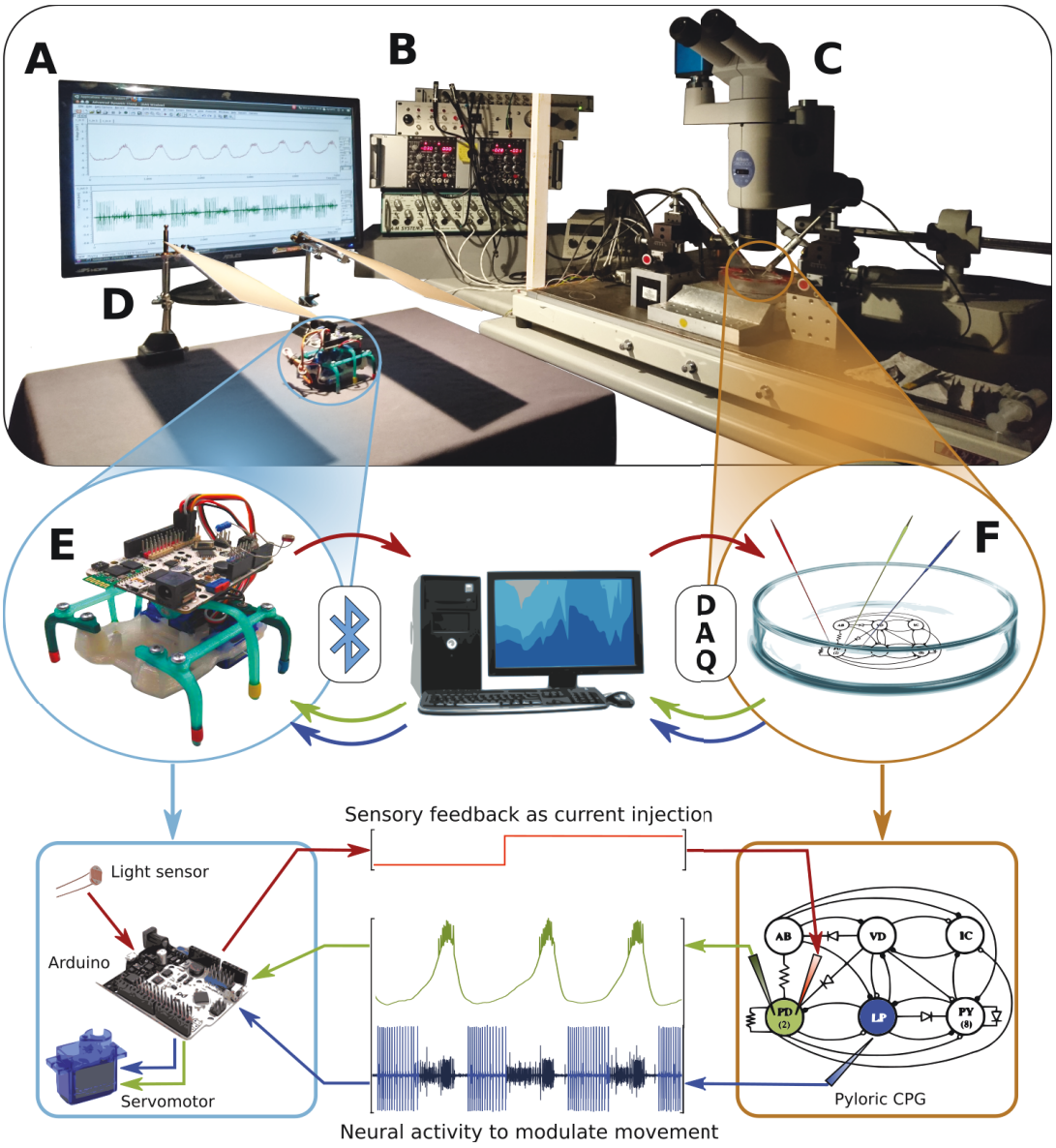

Figure 3.5: Top panel: illustration of the experimental setup for the FLC-Hybrot implementation: A) Computer that controls the closed-loop interaction between the living CPG and the robot. B) DAQ and signal amplifiers; C) Microscope; D) Testing track with light and shadows regions. **Middle panel:** detail of the elements of the setup: E) Hexapod robot, receives the neural information from the computer and sends back the sensory feedback through a Bluetooth connection; F) *In vitro* pyloric CPG preparation with three electrodes, one to record extracellular activity from the nerve which includes the activity from the LP neuron (blue), one to record intracellular activity from the PD neuron (green) and another to introduce the feedback current into a PD neuron (red). Electrodes and computer are connected through the DAQ device. **Bottom panel:** detail of the elements of the robot and the preparation. PD neuron (intracellular recording, green trace) and LP neuron (extracted from the extracellular recording, blue trace) behaviour is analyzed online by the computer and used to modulate the movement of the robot. Meanwhile, robot's light sensor detects the presence of light or shadow over it and the Arduino board sends that information to the computer, which injects feedback current into the PD neuron (red trace).

3.2.2 Experimental setup

The experimental preparation for this work was similar to the one described in 2.2.2. Living neurons were taken from the pyloric CPG of an adult *Carcinus maenas*. Before the dissection, the crab was anesthetized by introducing it in the freezer for 20/30 minutes. The stomatogastric ganglion, dissected following the standard procedure, was attached to a Petri with Sylgard cold saline dissolution (13-15^{o}C kept by a microcontroller) using pins. The saline had the following composition (in mM): 433 $NaCl$, 12 KCl, 12 $CaCl_2 \cdot 2\ H_2O$, 20 $MgCl_2 \cdot 6\ H_2O$, 10 $HEPES$, adjusted to pH 7.60 with 4M $NaOH$. Neurons were identified after desheathing the ganglion by their membrane potential waveforms and their corresponding spike times in nerves. Intracellular recordings were performed using 3 M KCl filled microelectrodes (50 $M\Omega$). For details on the preparation see [Elices et al., 2019].

Intracellular and extracellular electrodes were connected to NPI Electronic's ELC-03M DC and A-M Systems Model 1700 amplifiers, respectively, and then recorded by a National Instruments PCI-6251 board with a NI BNC-2090A DAQ device. This same components were used for current injection. Video recordings of the robotic locomotion were conducted with a Sony HDR-XR520 videocamera, with 25 frames-per-second video speed at 1080p resolution.

3.2.3 Robot design and construction

Our FLC-Hybrot was built using hexapod robot parts whose design were downloaded from an open repository[1] and all its elements were printed using a Prusa i3 3D printer. It had six legs, three on both sides of the robot (see Figure 3.5E). On each side, the front leg was attached to its corresponding back one by a rigid piece of plastic. Hence, each pair of front and rear legs moved together in an oscillatory manner from front-to-back and in reverse. Central legs from both sides were also tied together and oscillated up-and-down, alternating one side and the other. Phase for each front-rear pair was inverse, meaning that when one side was moving to the front the other was going back and vice versa. When one's side pair moved from back-to-front its corresponding central leg moved down, lifting the robot on that flank, thus suspending the front-rear pair over the ground during that swing. Simultaneously, the opposite side legs were moving from front-to-back and touching the floor while doing so, because their matching central leg was up at that instant, hence pushing the robot forward. Alternation of these two movements by the legs on each side of the robot enabled it to advance in a straight line. Therefore, the FLC-Hybrot locomotion had a biphasic behaviour.

Each rear leg was connected to a TG9d servomotor that moved it, and by association, it moved its linked front leg too. Both central legs shared a common axis, which was attached to a third servomotor. The robot included a photoresistor, which is an electronic component whose internal resistance varies according to the amount of light that it receives. This element worked as a light sensor that indicated when the robot was walking through an area with light or shadow. Additionally, LEDs where placed in the ends of the robot legs to facilitate video tracking. All these electronic components were governed by an Arduino-type microcontroller board.

In particular, an Arduino BQ Zum Core microcontroller board was used, which had a basic Atmel ATMEGA328P chip with a processing speed of 16MIPS. This board was chosen firstly because it was specially designed for connecting servomotors and included a large number of pins for this purpose. The second reason for this choice was that it had an integrated Bluetooth antenna for wireless communications. The robot was powered by one 9V rechargeable battery.

[1] https://www.thingiverse.com/thing:5156

The microcontroller was programmed using Arduino language and Arduino IDE. This code was in charge of controlling the servomotors, reading the data captured by the light sensor and establishing a bidirectional connection with the computer. In order to manage the servomotor's oscillatory activity more efficiently we utilized ArduSnake's Oscillator library[2]. This library contains functions to easily set and handle every parameter of the servomotors oscillation, including its period, amplitude, phase and initial position. Communication with the computer was carried out via Bluetooth and through a Serial port. Baud rate, or transmission rate, was set to 19200 since the Bluetooth module could only operate at such frequency. The robot sent to the computer the information registered by the light-dependent resistor. If its resistance value surpassed a specific threshold then it was receiving light and, in that case, the data sent to the computer was an integer equal to 1. Otherwise the robot was located under a shadow, and a 0 was sent. On the other hand, the board read all available values that came through the Serial port from the computer. These data consisted in 8-bits characters, which were stored sequentially in a string. When a newline character arrived, '\n', it indicated that the current string was complete and it was then split using the tabulator character, '\t', as delimiter. The resulting two substrings where then parsed into integer numbers and represented the amplitude and period, respectively, to be used as parameters for the legs oscillation. These movement settings were maintained until new values were received. In this manner, the FLC-Hybrot modified its locomotion according to the commands sent by the computer, which were at the same time determined by the living CPG behaviour.

3.2.4 Computer setup

The computer that served as linking element between the living circuit, via DAQ device, and the robot, via Bluetooth, had a 4-core Intel Core i7-6700 3.40 GHz processor and 16 GB RAM memory. The operating system was Debian 9 with kernel 4.9.0-4 and Preempt-RT real-time patch, to ensure the most precise soft real-time interaction possible. Bluetooth connection was established using an external USB Nano Stick v4.0 antenna. This connection was managed using BlueZ 5.43, the open-source official Linux Bluetooth protocol stack.

The software that controlled the closed-loop interaction between the robot and the CPG, which was executed in the computer, was written in C++ and compiled with G++ 6.3. It used Comedi 0.7.76 drivers to communicate with the National Instruments board and the open-source LibSerial 1.0.0 library to handle data transfer to and from the robot.

All data analyses were performed in Python 3.7.9 using libraries Pandas 1.1.4, Matplotlib 3.3.3, Numpy 1.18.5 and Scipy 1.5.4. Coefficient of determination, also known as R-squared or R^2, was used to measure the dynamical invariant existing between two temporal intervals in the signals. This metric calculates the proportion of the dependent variable variance that is predicted by the independent variable [Steel and Torrie, 1960]. This statistic is calculated subtracting the division of the residual sum of squares by the total sum of squares to 1:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

where each element is defined by the formulas

$$SS_{res} = \sum_i (y_i - f_i)^2 \quad \text{and} \quad SS_{tot} = \sum_i (y_i - \bar{y})^2$$

[2] https://github.com/Obijuan/ArduSnake/tree/master/ArduSnake

and y_i represents each sample of the data, f_i is each one of the fitted values and $\bar{y}$ is the mean of the data defined as $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. The function scipy.linregress was used to compute this value.

3.2.5 Software implementation

The nexus computer was tasked with ensuring the correct interaction between the electronic and biological elements required for the FLC-Hybrot to work properly. This was achieved by, on one hand, reading the data registered from the CPG neurons as well as injecting feedback current back into the living circuit, both things carried out utilizing an intermediate DAQ device. On the other it sent, via Bluetooth, instructions based on the registered neural activity to the robot, to modulate its locomotion while also receiving, using the same channel, the information obtained by the light sensor, which determined the value of the feedback current. With this purpose in mind we developed a C++ program that executed all the aforementioned operations. Communication with the DAQ board was established employing Comedi drivers. Bluetooth transmission with the robot was performed through a serial port and managed by the LibSerial open-source library[3]. Once all connections were set up, a real-time loop was started in order to handle the interaction. Soft real-time constrains were enough in this case since precision under the millisecond scale was not required and the system had fault tolerance. Moreover, the serial connection was not compatible with hard real-time. The code is available at https://github.com/GNB-UAM/FLC-Hybrot.

At the beginning of the loop there was a calibration time during which the neural signals were read and their amplitudes saved but the robot worked autonomously. These values, which were later updated periodically, were utilized to establish the detection thresholds for both neurons and to compensate for possible drifts in the recordings. Afterwards came the interaction phase where the following actions were performed:

1. Sleeping until it was time for the next interval of the real-time task.

2. Reading the light sensor data that was coming through the serial port.

3. Reading both intra and extracellular signals from the DAQ device.

4. Assessing whether a burst of the PD neuron has started or finished. Detection of PD bursts was carried out using two thresholds, a higher and a lower one, and a flag variable indicating if a burst was already going on. When there was no active event and the intracellular membrane potential surpassed the higher threshold a new burst was starting, and the flag was changed accordingly. If during the ongoing event the voltage went under the lower threshold it meant that the burst ended, and the flag went back to its waiting state.

5. Assessing whether a burst of the LP neuron has started or finished. Detection of LP bursts was carried out over an extracellular signal using one threshold and a flag variable indicating if a burst was already going on. When there was no active event and the extracellular voltage surpassed the threshold a new burst was starting, and the flag was changed accordingly. This flag was changed back to its waiting state just when a PD neuron burst ended. The reason behind this decision was that PD and LP bursts can not overlap due to the characteristics of the CPG triphasic rhythm, hence a LP neuron burst could never begin until the previous PD bursts finished.

[3] https://libserial.readthedocs.io/en/latest/index.html

6. Every time that the onset of an LP burst was detected, the period of this neuron's last cycle was calculated. At this moment, the duration of all the other intervals contained in this last cycle was also computed.

7. When both these intervals were obtained for a cycle in the CPG activity, this information was sent to the robot. Specifically, these values were used to modulate the amplitude, in degrees, and the period, in milliseconds, of the robotic legs' oscillation. Amplitude was computed as the duration of the LPPD interval (in milliseconds) divided by a constant value to obtain a number that represented an amount of degrees valid for the servomotors rotation range. Oscillation period was equal to the cycle period (Figure 3.6).

8. A string was composed with these two measures, separated by a tabulator character '\t' and ended by a newline character '\n'. For example, if there was a 30 degrees amplitude and a 1000ms period, the resulting string would be "30\t1000 \n". This string was sent to the Arduino board via Bluetooth.

9. If the information captured by the light sensor indicated that the robot was located under a shadow during that instant, then feedback current was injected into the PD neuron through the DAQ device. This current increased or decreased gradually until the target value was reached, or until zero when the injection was stopping. Its variation rate was defined by the formula $(target\_current/sampling\_frequency) * ramp\_duration$, with the duration being adjusted to fit the properties of each experimental preparation.

Once the interaction was over, all the data regarding the experiment was stored in a file, including the recorded neural signals, both temporal intervals duration along the experiment and the injected current. This operation was performed at the end of the procedure to avoid interfering with the closed-loop protocol.

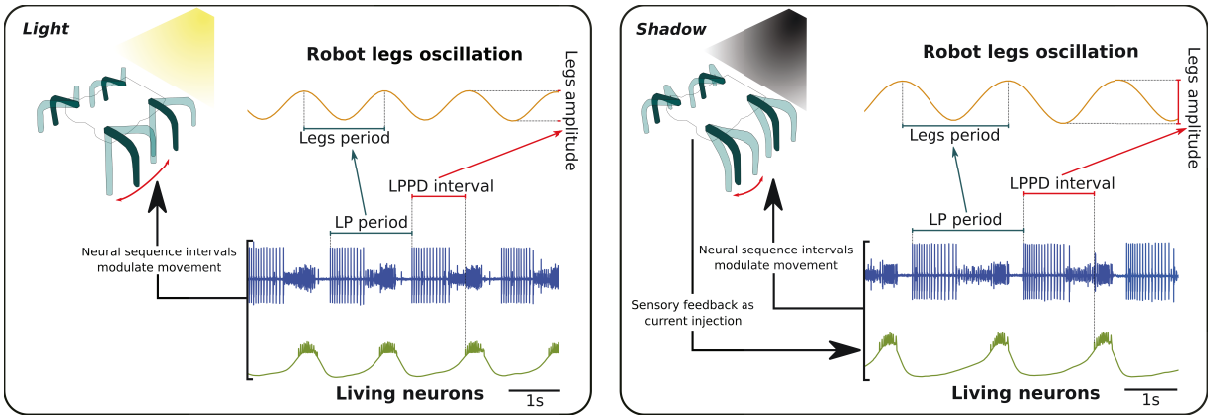

Figure 3.6: Functional living CPG neural activity modulates the robotic locomotion and the sensory feedback affects the circuit behaviour. The servomotors of the robot oscillate with a specific period, related to the cycle period, and amplitude, modulated by the time interval between the LP and PD bursts. When the robot's light sensor detects a shadow, feedback current is injected into the PD neuron, affecting the behaviour of the whole circuit. If this current is positive, the rhythm becomes faster, while if the current is negative it slows down (as in this figure).

3.2.6 Video tracking

Video tracking of the FLC-Hybrot movement was performed in Python with OpenCV 4.5.2.54 and imutils 0.5.4 libraries. The elements tracked were the LEDs attached to the robot legs, since

they are the most visible components, especially during shadow sections of the experiment. Every frame of the video was processed sequentially and the position of the LEDs on each of them was saved. For each frame, first a Gaussian blur (cv2.GaussianBlur) was applied to "clean" it and then the image was converted from BGR color scheme to HSV (cv2.cvtColor). A color mask was applied later to turn to black everything but the target element (cv2.inRange), and then two more filters were used to smooth the image even more (cv2.erode, cv2.dilate). The color boundaries of the mask were set manually by trial-and-error for each video. Afterwards, the contours of the remaining objects in the image were found (cv2.findContours, imutils.grab_contours), and the center of the one with the largest area was stored (cv2.minEnclosingCircle, cv2.moments). After going through all video frames the result was a signal representing the position, in axes X and Y, for the tracked objects throughout the experiment (Figure 3.7).

Once we had the signal for the movement of the robot leg, we applied a detrend function (scipy.signal.detrend) and a third-order Butterworth digital low-pass filter (scipy.signal.butter, scipy.signal.filtfilt) to smooth the signal and remove artifacts from the tracking. A second third-order Butterworth digital high-pass filter was then applied to remove the slow oscillation and flatten the signal. The resulting signal was a representation of the oscillatory movement of the robot legs during the experiment. From this oscillation, the period and amplitude for each cycle could be extracted and used to compute the R-squared statistic.

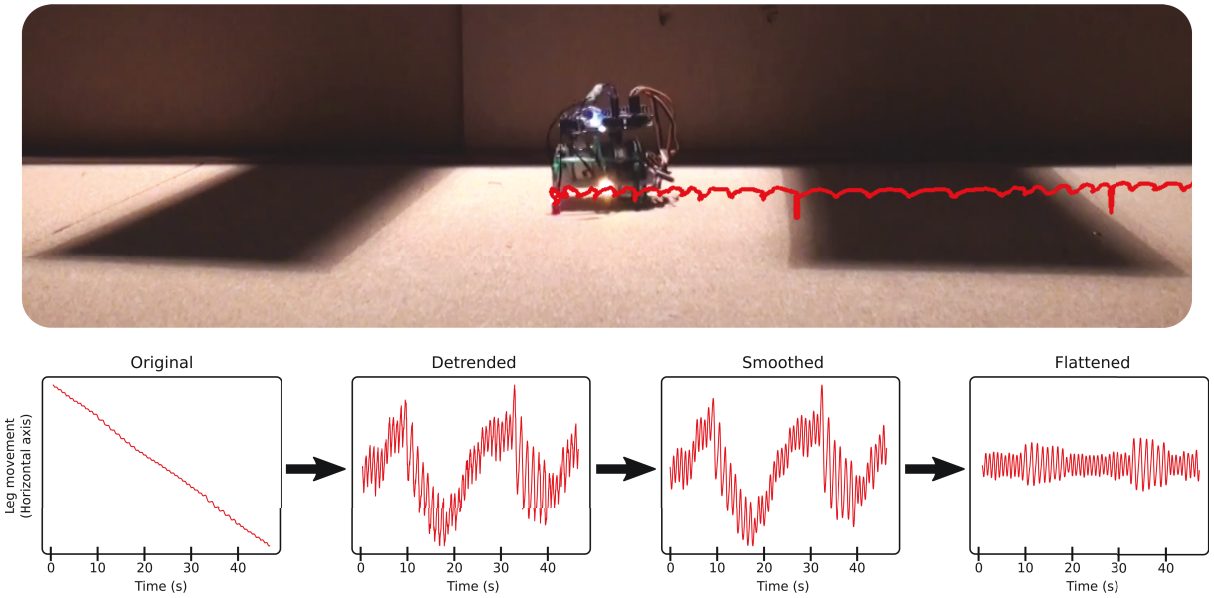

Figure 3.7: Illustration of the video tracking procedure. Red line represents the movement of the LED attached to the front leg of the robot. This signal is then detrended, smoothed and flattened to obtain the robotic leg oscillation behaviour during the experiment.

3.3 Results

3.3.1 Invariant-driven coordinated locomotion under different sensory cues

In order to validate the adequate locomotion of the FLC-Hybrot when modulated by the pyloric CPG online behaviour, as well as the living circuit real-time adaptation to the injected feedback, we performed a set of experiments employing a 1.5 meters long trial track. Along this surface there were interspersed segments of lights and shadows. This configuration caused the FLC-Hybrot to alter its behaviour several times during the experiment due to the injection of current

into the PD neuron when it walked under a shadow. Current injected in a shadow section varied from one experiment to another, according to the neurons response to the stimuli. When the robot was located on a luminous area the injected current amount was 0 nA. The following video illustrates one of such experiments https://youtu.be/Dltec7TeGso.

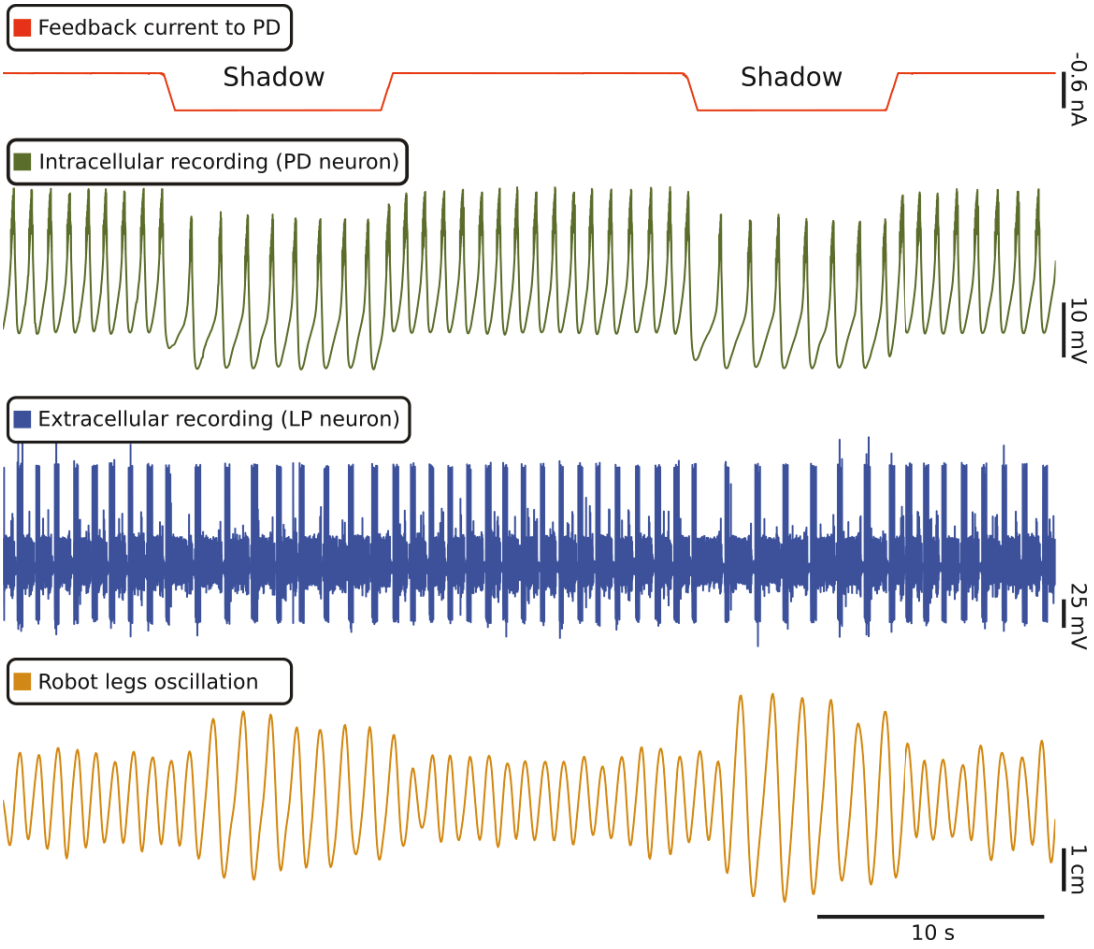

Figure 3.8: Flexible hybrot adaptation to environmental changes when noticed by the robot sensors and associated coordinated locomotion. When the robot entered a shadow section, it sent sensory feedback to the living CPG in the form of positive electrical current (red trace). Blue and green traces are recordings of the extracellular and intracellular (PD neuron) activity of the circuit and display the change caused by the feedback current injection in the PD neuron, slowing down the CPG rhythm. The oscillatory movement of the robot legs is represented in the brown trace and it can observed that a variation in the CPG rhythm leads to a change in the robotic locomotion, modifying both the period and the amplitude of the oscillation.

Figure 3.8 shows the results for one of these tests. An alteration in the neural activity is clearly appreciable when -0.6nA current is inserted into the circuit, causing its rhythm to slow down while also modifying the PD neuron membrane's potential amplitude. This change is immediately reversed as soon as the current goes back to zero, restoring its previous behaviour. Concerning the robot's locomotion, a variation in the legs' oscillation is observed just after the neurons alter their functioning. Despite the successive changes in the amplitude and period of its legs' oscillation, the FLC-Hybrot maintained a coordinated and effective locomotion during the whole experiment.

The dynamical invariant present in the pyloric CPG activity, in the form of a linear relation between the LP neuron period and the LPPD interval, was effectively transferred to the robot

locomotion. It displayed the same correlation between its legs oscillation period and amplitude, which were modulated by the two previous temporal intervals. Figure 3.9 shows a comparison between the living CPG and the robot dynamical invariant, with the later reaching an R^2 correlation of 0.87 despite the lack of precision of its servomotors.

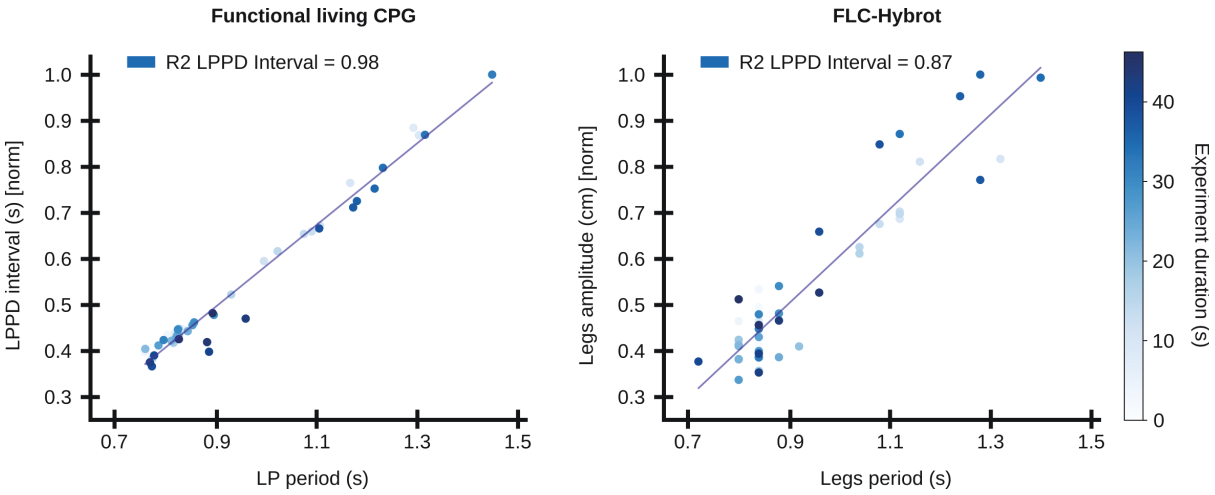

Figure 3.9: Left: dynamical invariant present in the living CPG activity during the validation test, represented as a linear relation between the LP neuron bursting period and the LPPD interval cycle-by-cycle. **Right:** dynamical invariant present in the FLC-Hybrot locomotion during the validation test, represented as a linear relation between its legs oscillation period and amplitude cycle-by-cycle. The dynamical invariant property is effectively translated from the living CPG to the robot locomotion, codified as: Robot period = LP period, Robot amplitude = LPPD interval * factor.

3.3.2 Assessment of the locomotion using intervals that build dynamical invariants

Having proved that the dynamical invariants present in the pyloric CPG activity can be effectively transferred into the hybrot legs' oscillation and that this results in a coordinated locomotion, we wanted to assess the differences in the robot's movement when controlled using temporal intervals that maintained a dynamical invariant relation and those that did not have such relationship. Since we employed the cycle period to modulate the oscillation period (time) and the second interval to modulate its amplitude (distance), the relationship between them was rendered as the hybrot overall moving speed (distance over time). Given that dynamical invariants keep a linear relationship, they should be translated into a constant hybrot velocity, while using intervals that do not maintain such relation should result in a more unstable behaviour in terms of speed, with sudden accelerations. Keeping a stable speed, despite changes in the environment, can be a functional requirement in many robotic locomotion applications (e.g., when transporting fragile or hazardous materials).

With this goal in mind we decided to perform a series of offline trials utilizing recordings from five different sessions. This way, we could compare the hybrot's behaviour when controlled by temporal intervals that maintained a linear relationship with the cycle period and by others that did not, all of them extracted from the same neural sequences, even the same cycle period. Specifically, for each session we carried out two trials, one of them modulating the legs' oscillation amplitude as a function of the LPPD interval and the other using the PD burst. These intervals were chosen since the duration variability for the first one displays a very strong correlation with

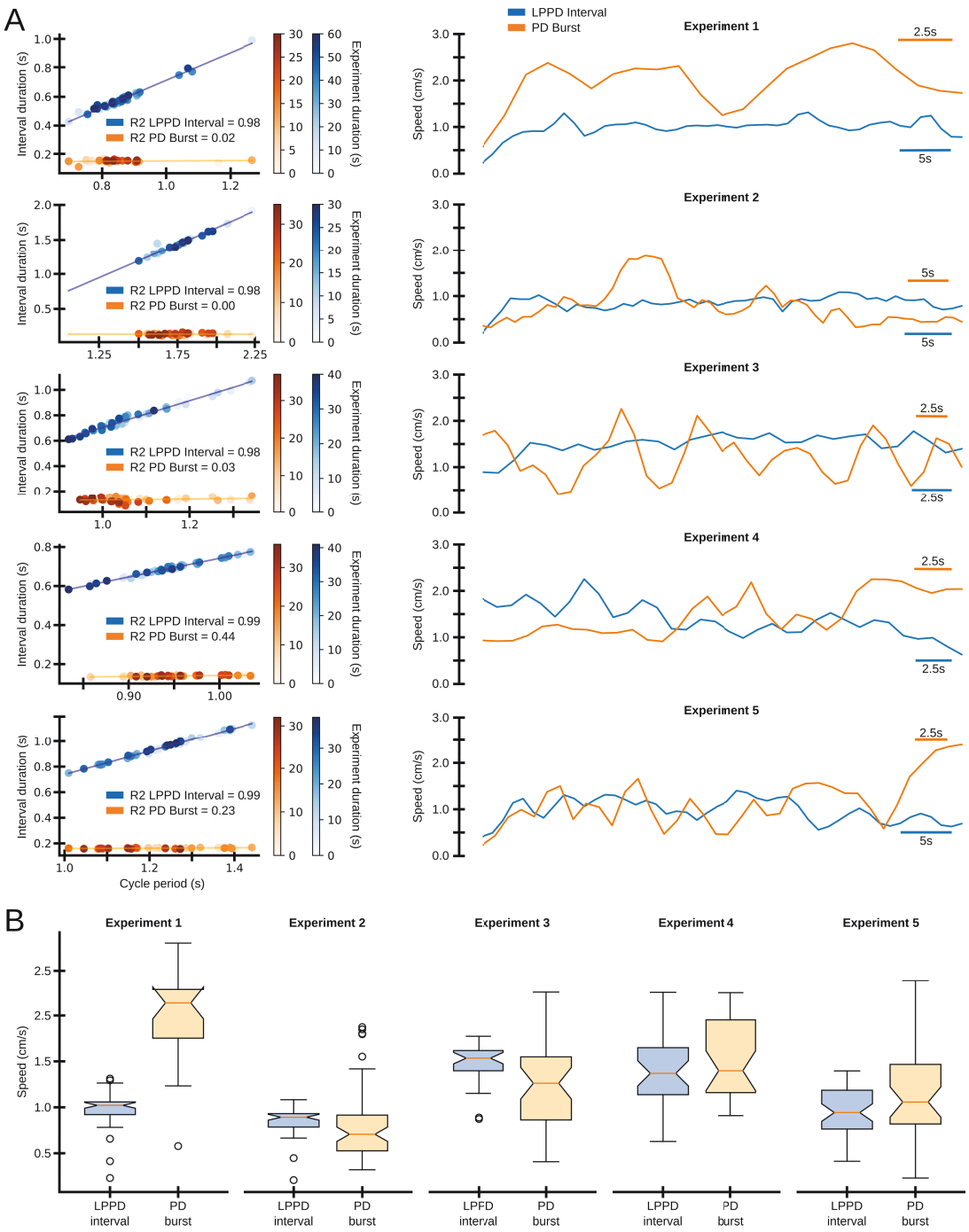

Figure 3.10: Assessment of the hybrot locomotion when using different intervals that build dynamical invariants to modulate its legs' oscillation. **A:** for each experiment, **left panel** represents the relation between the cycle period and the LPPD interval and PD burst duration during the tests. **Right panel** shows the instantaneous speed of the hybrot during the trials, calculated every one second, when using the LPPD interval or the PD burst to modulate the legs' oscillation amplitude. **B:** comparison of the hybrot speed for each experiment when using the LPPD interval or the PD burst to modulate the legs oscillation amplitude, as quantified in the boxplots.

the cycle period while the second, generally, is very independent. During each trial, the hybrot had to traverse a given distance, equal in all of them. These tests were video recorded and the

position of the robot's body was obtained by video tracking, similarly as done earlier with the leg.

Since these were offline tests, and therefore we knew the intervals duration beforehand, we decided to change the oscillation's amplitude codification. Instead of utilizing the formula $interval\_duration * factor$ as we previously did, we mapped each interval's minimum and maximum duration to the minimum and maximum amplitudes tolerated by the robot's legs, respectively. In such manner, in every trial the maximum possible amplitude of the leg oscillation was utilized.

Figure 3.10 shows the results for these tests. For each of the two trials with each session we computed the relation between the cycle period and the duration of the cycle-by-cycle intervals (Figure 3.10A left column). Likewise, we calculated the hybrot's instantaneous speed during the trial, computed as the distance covered (in cm) per second (Figure 3.10A right column). We can observe that the LPPD interval duration maintains a linear relation with the cycle period despite their variability, and that this relationship is much stronger than the one kept by the PD burst duration. This is translated into a more stable speed when using the first interval, with less sudden accelerations and large variations. Note that in sessions 4 and 5, R^2 values for the second interval are significantly high (i.e., this measure is usually close to 0 for this interval), which resulted in a much more constant speed for the robot, but still less stable than the one achieved with the LPPD interval. When comparing the speeds obtained during each trial for each session, it can be observed that the use of temporal intervals that maintain strong dynamical invariants with the cycle period always involve a more stable overall movement velocity (Figure 3.10B).

3.4 Discussion

FLC-Hybrot, a hybrot composed of an hexapod robot and a functional and living CPG acting as its "brain", has been designed and constructed in this project. To the best of our knowledge this is the first hybrot built with a functional living CPG, and the first time that a dynamical principle arising in a functional living circuit is validated to control a robotic device with sensory feedback. The robot's legs oscillation was modulated by the CPG activity on each cycle of its rhythm, specifically by those temporal intervals that are known to maintain linear relations in the form of dynamical invariants. Oscillation period was determined by the period of each cycle of the neural circuit activity while its amplitude was defined by the LPPD interval. Likewise, the robot included a sensor that captured information regarding the amount of light in its surroundings. When the FLC-Hybrot was located in a shadowed area it sent sensory feedback in the form of electrical current to the living neurons. This stimulus changed the behavior of the circuit, hence also modifying the robotic locomotion. In this manner, a real-time closed-loop interaction was established between the biological and electronic components of the hybrot.

Hybrots constitute an useful resource to study neural circuits dynamics and their relationship to behavior since they provide an artificial powerplant to implement experiments that otherwise would lack one, like those performed *in vitro* or *in vivo* with immobilized animals. In our particular case, we designed FLC-Hybrot to analyze how the dynamical invariants present in the *Carcinus maenas* pyloric CPG could be applied for the coordination of robotic locomotion. We validated that when modulated by these temporal intervals, the oscillatory motion of the robot legs remained coordinated at all times, despite all the variations in amplitude and period that took place during the experiments. Moreover, video tracking of the hybrot movement revealed that dynamical invariants were effectively translated into the relation between the oscillation amplitude and period. This resulted in a more stable and constant movement speed for the hybrot than when coordinated by intervals that did not maintain a dynamical invariant relationship.

The temporal constrains required for the online interaction of biological and synthetic elements, which have very different characteristics and functioning dynamical mechanisms, make the implementation of hybrots a challenging task. These difficulties also include the need of recording the neural signals and detecting in them the desired events, performing the same process with the robot's sensory information, as well as stimulating and controlling all the involved components in real-time. This work was carried out thanks to the code and knowledge previously developed for RTHybrid. With the purpose of facilitating and promoting the use of this kind of technologies and protocols for future works, FLC-Hybrot was built employing open 3D designs and all the code developed is available in an online repository https://github.com/GNB-UAM/FLC-Hybrot.

3.5 Future work

Our study focuses on a triphasic CPG rhythm defined by the sequential and ordered bursting of the LP, PY and PD neurons. However, the constructed robot movement only comprises two phases: lateral legs move forward or backwards simultaneously and alternating, while the central legs oscillates to one side and the other synchronized with the lateral ones motion. Therefore a new robotic body model, with a wider range of movements that can take advantage of the whole triphasic rhythm properties, can be designed in order to conduct a more detailed analysis of the effect of dynamical invariants in the hybrot locomotion.

The approach proposed in this work can be utilized not only for further exploration of CPGs' properties and dynamics with the goal of understanding the mechanisms that produce their robust motor control, but can also be applied in novel studies to comprehend different neural dynamics, employing other types of robots, tests and environments.

Functional living circuit hybrots can also be a first step to validate existing neural dynamics for the control exoskeletons and other prosthetic devices relying in sequential neural activity for motor coordination. These kind of applications can heavily depend on robotic feedback while using dynamical invariants to keep coordination cues.

4

CNN-ripple: an offline and online deep learning based detection method for real-time closed-loop intervention of sharp-wave ripples

4.1 Introduction

This chapter describes the implementation of CNN-ripple, a method powered by an artificial neural-network to detect sharp-wave ripples both in offline and online contexts. For this purpose we employed several software and hardware technologies in the development of a system capable of real-time signal recording, data processing and electrophysiological intervention while using machine learning algorithms as a detection method. Numerous experiments with rodents were conducted in order to validate CNN-ripple performance in both offline and online configurations. This project was fulfilled in collaboration with Dr. Liset Menéndez de la Prida's laboratory at Instituto Cajal (CSIC).

Similarly to previous chapters, here we first introduce the motivation and state of the art of the project, followed by a description of the materials and methods utilized and then an analysis of the obtained results. The chapter is concluded by a discussion of the developed work.

4.1.1 Motivation and state of the art

Of all the different dynamics that take place in the neural system, brain oscillatory patterns are among the most studied ones. These oscillations are detected in the collective activity of the neurons and are tightly related to specific cognitive functions [Buzsáki et al., 2012, Friston et al., 2015]. Due to its own oscillatory nature, the study of such events is typically carried out using spectral analysis methods [Schomer and da Silva, 2005]. However, some other features of these signals waveforms, either when they are recorded as extracranial electroencephalogram waves (EEG) or intracranial local field potentials (LFP), sometimes comprise important information to comprehend the role played by these phenomena in neural dynamics, like their amplitude or their entropy [Modi and Sahin, 2017].

Sharp-wave ripples (SWR) are found among these oscillations, and they are high frequency events (ripples) matched with a sinking of the LFP (sharp-wave) [Buzsáki, 2015]. These SWR

have been found in the hippocampal activity of every mammal and are related to memory consolidation and retrieval tasks [Joo and Frank, 2018]. There are numerous works that attempt to detect SWR and interact with them in real-time to study the role of these oscillations in memory formation, decision making and spatial coding. The disruption and extension of these events through electrophysiological stimulation has been proven to alter the aforementioned neural processes [Dutta et al., 2019, Fernández-Ruiz et al., 2019, Girardeau et al., 2009, Jadhav et al., 2012, Oliva et al., 2020, Aleman-Zapata et al., 2021]. Most of these approaches are based on spectral methods which fall short to capture the underlying variability of many of these events since they can only look for frequency-related characteristics.

On the other hand, deep learning techniques, which are based on artificial neural networks, have become the state-of-the-art and the most extended tools for detection and classification tasks in a wide range of different types of data, such as images [Srivastava et al., 2021], videos [Redmon et al., 2015, Bochkovskiy et al., 2020], natural language [Brown et al., 2020] and audio signals [Purwins et al., 2019]. Deep learning models can be taught to look for specific patterns by training them with data previously labelled by an expert and then used on unlabelled data (supervised learning). Alternatively, the training data can be unlabelled so the model must look for patterns and clusters of information on its own (unsupervised learning). Since artificial neural networks are such a great tool to detect objects and patterns in so many different fields, even in real-time, it was only natural that they also got applied to biological and electrophysiological data [Özal Yıldırım et al., 2018, Xu et al., 2020, Celik et al., 2020, Li et al., 2019].

In particular, there are already works that attempt to identify SWR using these kind of techniques, focusing mainly in the use of recurrent neural networks, which are specially designed to handle sequential information. [Hagen et al., 2021] followed a supervised learning process to train a detection model, using one-channel raw LFP recordings as input. SWR were previously labelled using a standard filter-based method to create the ground truth. The network architecture consisted in several recurrent Long-Short-Term-Memory (LSTM) and convolutional layers with a dense final layer. This method achieves a good performance in some scenarios, but these values get reduced when applied over continuous data such as a typical online recording, even when the sections where the animal moves are removed to avoid muscle noise. Moreover, since the network was trained using data labelled by a spectral-method it only looks for SWR with the same standard frequency features. This algorithm was designed for offline detection and therefore is not well suited for real-time closed-loop experiments.

The goal of this project was to design and develop a real-time sharp-wave ripple detection method based on artificial neural networks with further functionality than the one described above. Specifically, convolutional neural networks (CNN) were used, a kind of network which is currently being extensively utilized for image recognition problems but can also be employed for pattern detection in signals. An advantage of this type of architecture is its huge processing speed and little need for data pre-processing, thus allowing its use in an online scenario [Bai et al., 2018]. Appropriate functioning of the implemented solution was validated, both offline and online, over several datasets acquired from different experimental contexts. At the same time, its performance was compared with a traditional spectral-based detection method. In this chapter we also detail the various tools and components, both software and hardware, that were employed in the making of a closed-loop system designed to detect SWRs and disrupt them in real-time during an *in vivo* experiment. This closed-loop implementation was build around the developed deep learning model.

4.1.2 Brain oscillations, hippocampus and sharp-wave ripples

As mentioned in previous chapters of this work (see Section 2.1.2), neurons' membrane potential activity displays oscillatory patterns in the form of spikes or bursts, generated by their own

internal mechanisms and the interaction with other neurons. When the activity of a large group of neurons is synchronized, it gives rise to macroscopic oscillations visible in EEG or LFP recordings [Steriade et al., 1990]. These macroscopic oscillations are present in all mammal brains, including humans, and there are various of them that can be identified by their signature frequency band, such as alpha (8-12 Hz), beta (12-25 Hz), gamma (40-80 Hz), delta (1-4 Hz) or theta (4-12 Hz) (Figure 4.2A) [da Silva, 2013]. It is known that these oscillations are related to different cognitive and behavioural functions, so their study is considerably extended [Buzsáki et al., 2012, Friston et al., 2015].

Several of these oscillations can be found in the brain region known as hippocampus. This region is in charge of episodic memory, also referred as autobiographic memory, and spatial coding, meaning that it codifies the environment, creates mental maps of the world and consolidates them in the form of memories [Andersen et al., 2006]. The hippocampus has a curved shape resembling a C letter, and for this reason it is usually compared with a sea horse or a ram horn. Some of its regions are named after the Latin name of the ram horn, *Cornu ammonis*, and are known as CA1, CA2 and CA3 [Amaral and Lavenex, 2006]. Two symmetrical halves of the C structure are also differentiated, being the one closest to the skull called dorsal, and the most distant, ventral [Moser and Moser, 1999]. Additionally to this distribution in subregions, the hippocampus is also divided in layers that span across all the formation: Stratum Oriens (SO), Stratum Pyramidale (SP), Stratum Radiatum (SR) and Stratum Lacunosum-Moleculare (SLM). Due to the curved shape just mentioned, these layers have opposite orientations both in the different subregions (CA1-CA3) as well as in ventral and dorsal disposition (Figure 4.1).

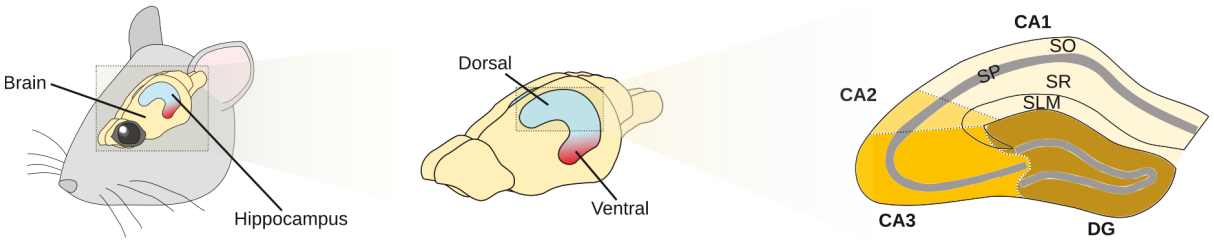

Figure 4.1: Illustration of a rodent's brain and hippocampus. Hippocampus displays a C-like curved shape and is separated in dorsal and ventral regions, according to their distance to the skull. Inside each of these parts, the hippocampal formation is separated in several subregions, the most prominent of them being CA1, CA2, CA3 and Dentate gyrus (DG). All these subregions are at the same time divided in different layers: Stratum Oriens (SO), Stratum Pyramidale (SP), Stratum Radiatum (SR) and Stratum Lacunosum-Moleculare (SLM). Due to the hippocampal regions distribution, these layers are orientated in opposite directions in CA1 and CA3, as well as in the dorsal and ventral areas regarding each other.

Of all the oscillations present in a rodent's hippocampus the strongest one is theta. It appears while the animal is awake and, generally, moving. It is a state of awareness related to exploration during which spatial codification takes place [Buzsáki, 2015]. Some hippocampal neurons, known as place cells, fire when the animal is situated in a specific location. During movement this firing is coordinated by theta, forming organized sequences of neural activity that encode spatial information [Olafsdottir et al., 2018]. However, when the animal is at rest, or performing tasks such as moving, eating, drinking or sleeping, some large amplitude reflections of the LFP emerge. These are known as sharp-waves and are usually coupled with a fast oscillation (100-250 Hz) known as ripple (Figure 4.2B) [Buzsáki, 2015]. Sharp-wave ripples (SWR) can be found, among other areas, in the CA1 subregion and are mainly located in the pyramidal layer (SP). During sharp-wave ripples there is a repetition of the same place cells firing sequences that occurred when there was movement. This replay process is believed to serve as a memory

retrieval and consolidation mechanism [Prida, 2020]. SWR replays that occur during the awake state may display a forward or a reverse order. Place cells fire in the same disposition than in the original sequence during forward replay, suggesting an involvement in information retrieval tasks related to spatial navigation decision-making. In other words, forward replay helps the animal recall past events in the process of choosing its next steps. On the other hand, reverse replay reproduces the sequences in the opposite direction from what the original experience generated and typically appears when a destination is reached, implying some kind of reward consolidation effect. As opposed to the awake state, most replay sequences during sleeping periods present a forward orientation but their role is a consolidation one, reliving past sequences the same way they took place in order to reinforce and integrate those memories [Roumis and Frank, 2015]. Electrophysiological interaction with SWR produces changes in these learning and navigation processes, disrupting memory consolidation and retrieval when these events are interrupted but also improving them when some brain areas are stimulated just after a SWR takes place [Joo and Frank, 2018].

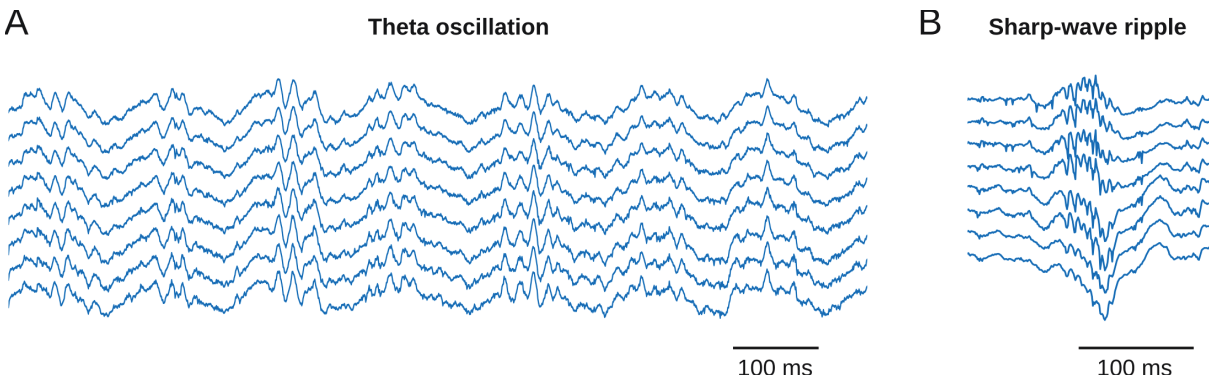

Figure 4.2: LFP traces with examples of different oscillations present in a mouse hippocampal activity. **A:** theta oscillation during a walking period. **B:** sharp-wave ripple during a resting period. LFP recordings kindly provided by Teresa Jurado-Parras, from De La Prida's lab at Instituto Cajal (CSIC).

4.1.3 Spectral-methods for sharp-wave ripples detection

Traditionally, SWRs detection has been carried out using frequency-based filters, since the most characteristic feature of these events is, a priori, their high frequency oscillation [Girardeau et al., 2009, Jadhav et al., 2012]. Normally when using this method, a single LFP channel is selected, usually located in the Stratum Pyramidale layer (SP), and a pass-band filter with a band around 100-300 Hz is applied to it since that is the typical oscillation frequency of a ripple. The filtered signal will therefore have a larger amplitude where the original one oscillated more in that spectral range, or in other words, where a ripple took place. An amplitude threshold is then established and a ripple is detected when the filtered signal crosses this line. In offline analyses, the envelope of the filtered signal is usually computed and utilized instead of the filtered data (Figure 4.3). However, this kind of techniques are only able to detect events with high power in a specific frequency band and therefore many other features, which provide great richness and variability to these phenomena, are not taken into account. On the other hand, noises, artifacts and other events that oscillate in the same frequency band are accidentally captured by these methods.

Computing the envelope of the signal and other additional processing steps that help reducing the number of false positives detected introduce several milliseconds latencies when implemented online, so they are typically avoided in that context [Dutta et al., 2019]. One modification

that significantly improves the precision of these techniques and can be employed in real-time scenarios consists in using a secondary electrode placed in a location where ripples are not present. Therefore, if an event is detected in both channels simultaneously it is considered as noise or an artifact and it is discarded [Oliva et al., 2020]. Similarly, if a second electrode is inserted deeper in the hippocampal formation, it can be high-pass filtered in order to detect sharp-wave slow oscillations. Thus with this protocol, a SWR is detected when both a sharp-wave deflection and a ripple are identified in their respective channels [Fernández-Ruiz et al., 2019] (Figure 4.3).

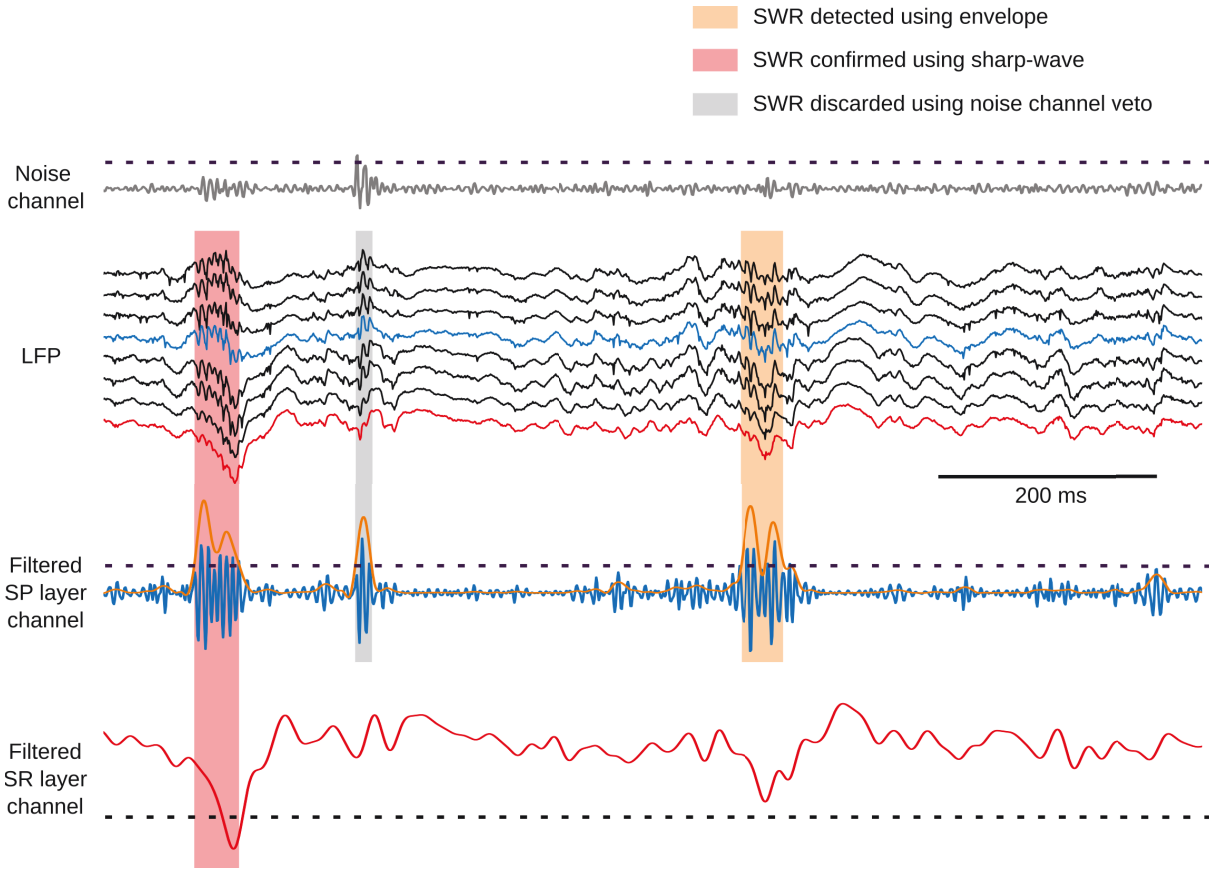

Figure 4.3: Example of SWR detection using different spectral-based strategies. Orange marked events have been detected when the envelope of a filtered pyramidal layer channel (between 100 and 300 Hz) crossed a determined threshold. Red marked events are those that after being detected by the filtered Stratum Pyramidale signal crossing a threshold (without the envelope) have been also confirmed by the presence of a close sharp-wave found in a low-pass filtered Stratum Radiautum channel. Another approach is to employ a channel from a different shank that has no ripples as a veto, filtering the signal in the ripple band and applying the same threshold than to the SP channel. When both channels cross the threshold simultaneously then the SWR is discarded (gray marked). LFP recordings kindly provided by Teresa Jurado-Parras, from De La Prida's lab at Instituto Cajal (CSIC).

4.1.4 Deep learning

Deep learning is a subset of techniques from the machine learning family that are characterized by the use of several non-linear levels of representation and abstraction to extract high level features from a specific input [Cun et al., 2015, Deng and Yu, 2013]. They are broadly utilized

in image pattern recognition and classification applications, natural language processing and design of artificial intelligence. They are based on artificial neural networks (ANN), whose basic structure is inspired by biological neural networks and are constituted by processing nodes called artificial neurons. These neurons can be connected between them as if they had synapses, using weight values to ponder the information received from each neighbour. Neural networks contain one input layer, that receives the original data to process, and an output layer, which produces the final outcome of the algorithm. Deep neural networks are those that, in between of those layers just mentioned, also incorporate any number (but at least one) of intermediate layers, known as hidden layers because their functioning is unknown to the end user for being in the middle (Figure 4.4A). Artificial neurons apply non-linear functions to the data they receive as input and produce an output that moves forward to the next layer's neurons, usually related to a specific task (Figure 4.4B). For example, in a facial recognition neural network one layer may be specialized in identifying only eyes, while other does the same just with mouths, and so on.

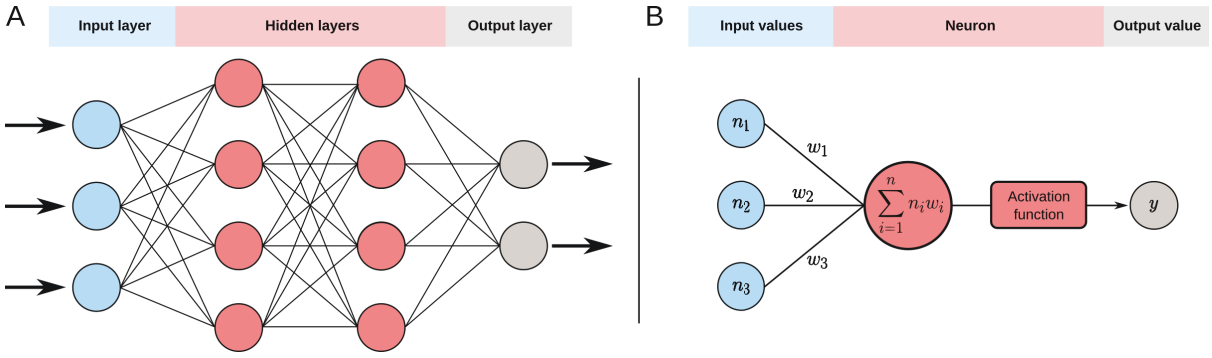

Figure 4.4: A: example of an artificial neural network architecture. In this case it is a fully-connected network, since each neuron is connected to every neuron in the next layer. Hidden layers neurons are the ones that actually process the data and the final layer generates the output. **B:** illustration of an artificial neuron functioning. The neuron receives and integrates the output from the neurons of the previous layer (or even next layers, if there is back-propagation) and then applies an activation function (e.g., sigmoid, ReLU, etc.) to produce an output. Inputs from other neurons are pondered according to some weights associated to the connections. Artificial neural networks learn by updating these weight values during the training process to change the produced output until it better fits the input data.

4.1.5 Supervised vs Unsupervised learning

Deep learning algorithms can follow supervised or unsupervised learning strategies to adjust the weights (the connections between neurons) of the network, updating their value on each step of the process in order to produce the most accurate output for each piece of input information [Mohri et al., 2018]. In supervised learning, the network is trained using pairs comprised by the input data and their corresponding expected output. The algorithm considers a loss function and the goal of the training is to minimize this loss. Unsupervised learning is used when the algorithm only receives input data to train, without their corresponding labels, so it must uncover the underlying patterns on its own. There is a third type or learning strategy, called semi-supervised, that uses a small amount of labeled data and a bigger one of unlabeled as training data.

In supervised learning, the resulting trained model should be able of generating the appropriate outputs for new data that was not used for training. One of the most common issues that can appear when tuning a machine learning algorithm is overfitting, which occurs when the

model works perfectly fine over the training data but then performs terribly when applied to novel inputs [Domingos, 2012]. When this occurs, the algorithm has "memorized" the training data too well and now is not capable of properly fitting data that it has not seen before (Figure 4.5). This problem is usually a consequence of a very extensive and complex model architecture or utilizing a small amount of data for training. Depending on its source, overfitting can be avoided by applying different solutions [Ying, 2019].

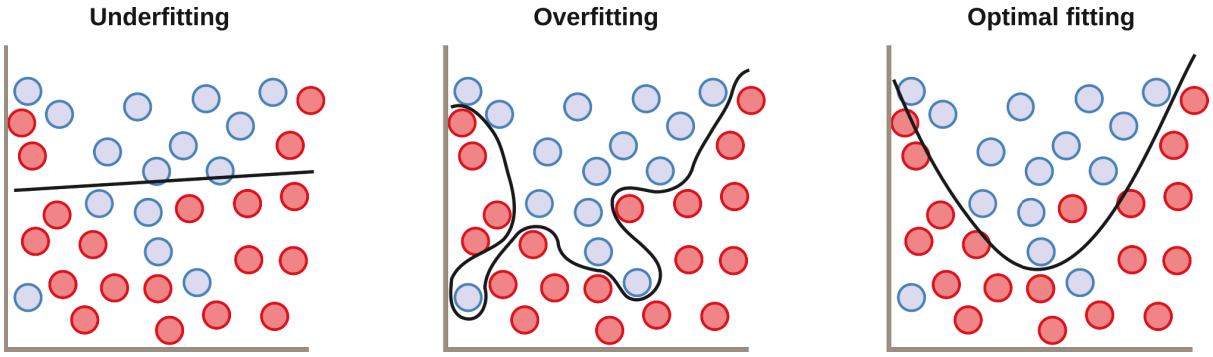

Figure 4.5: Graphical examples of underfitting, overfitting and optimal fitting for a machine learning algorithm that is trained to separate two types of elements (blue and red). Underfitting happens when the method is not even able of properly learn the features of the training data. On the other hand, if the algorithm adapts too much to the training data and is not capable to generalize when new data is presented, then overfitting occurs.

4.1.6 Dealing with overfitting

First of all, and in order to detect overfitting, it is essential to split the data into training and test sets, with the later only being used after the model has been completely trained and tuned: if performance is much worse on the test set then the model is probably overfitting. The training set should also be balanced, meaning that the number of samples belonging to each labelled class should be similar, to avoid the model learning that, since one the classes is more common, it is safer to predict that an element belongs to it. If the dataset is imbalanced and there is no way to fix it, then it can be compensated by assigning weights to each class to reduce the importance of the more heavily populated groups. Regularization techniques (which are addressed in the following paragraph) also help when dealing with imbalanced training sets.

Training dataset can also be separated into different train-development-test subsets to be utilized while tuning the hyperparameters. Typically, increasing the size of the training set results on a better generalization capacity, thus reducing overfitting, and the same effect can sometimes be achieved by simplifying the algorithm architecture and loss function, which decreases the number of parameters to fit. Supervised learning is an iterative procedure which is usually executed until the loss cost is stable. By looking at the error curve for the training and test sets for each iteration, sometimes it can be seen that after a certain point the model starts overfitting, so the process can be early stopped at that moment. Regularization techniques are also used to avoid overfitting, like those of type L1 and L2 that add an extra regularization term to the loss function on every iteration of the training process, hampering the chances of it getting trapped on a local minimum [Cortes et al., 2012]. Regularization by dropout is based on randomly ignoring some neurons during the learning process and pursues the same goal [Wan et al., 2013].

4.1.7 Architectures, layers and parameters

There exist many different sorts of artificial neural networks in terms of their topology, connections and type of elements. Based on the first neural model by McCulloch and Pitts [McCulloch and Pitts, 1943] and Hebbian learning plasticity rules [Hebb, 1949], Rosenblatt developed Perceptron in 1958, the first artificial neural network [Rosenblatt, 1958]. Since then there have been numerous innovations and improvements in this field [Rumelhart et al., 1986, Cun et al., 1998, Glorot et al., 2011, Ciregan et al., 2012, Schmidhuber, 2015] and these algorithms are now considered state-of-the-art for several regression and classification tasks. Some of the most relevant and utilized nowadays are Convolutional Neural Networks (CNN) and Long Short Term Memory networks (LSTM). CNNs are networks that contain at least one convolutional hidden layer whose task is, as its own name implies, to apply a convolution operation over the data. LSTMs are a type of Recurrent Neural Networks (RNN) that incorporate feedback connections between their layers. RNNs are specialized in time series processing, such as natural language processing. Despite being originally conceived to work with two-dimensional images, CNNs can actually achieve a similar performance over one-dimensional signals as RNNs [Bai et al., 2018]. Since CNNs have a lower computational cost, they are better suited to be employed in a real-time environment. One example of a state-of-the-art object recognition algorithm based on CNNs is You-Only-Look-Once (YOLO), which is able of detecting different elements on a video stream in real-time [Redmon et al., 2015, Redmon and Farhadi, 2016].

Artificial neural networks have numerous hyperparameters that must be properly tuned for an optimal training and performance. Some of them are common to all ANNs while others are particular to some kinds of architectures or layers. The most important hyperparameter for any supervised network is the learning rate, which defines how much does the model change on every iteration of the training process while the algorithm tries to minimize the loss function. A too big value may cause the algorithm to skip the absolute minimum, while a too small one can lead to it getting stuck in a local minimum or taking a very long time to converge on the correct solution. Another relevant hyperparameter is the batch size, that designates the number of samples from the dataset that are introduced into the network at once during training. An excessively small value usually implies a faster training and less memory usage, but it also produces larger fluctuations when trying to converge on a solution [Masters and Luschi, 2018]. Regarding CNNs unique parameters, the number of kernels of each convolutional layer is a very decisive one. Each of these kernels will get specialized in identifying a specific feature of the data, so a greater number of them will be able of capturing more characteristics but also will entail a higher computational cost and risk of overfitting during training.

The algorithm in charge of modifying the networks' weights and learning rate over the loss function minimization process is the optimizer. There exist several of these optimizers, generally based on gradient descent strategies, and each one of them include their own hyperparameters to tune. Differences between these algorithms are related to their efficiency and speed when finding the best solution, how easily can they get stuck on a local minimum or how difficult is to properly adjust their hyperparameters, among others [Buduma and Locascio, 2017]. In addition to the optimizers, a decay rate can be used to apply complementary changes to the learning rate during training.

4.1.8 Multi-site recordings and optogenetic stimulation

In prior chapters of this thesis, all recordings, both intracellular and extracellular, were performed using a single electrode. However, nowadays technology allows the acquisition of multiple signals by several electrodes simultaneously. Multi-site probes integrate numerous electrodes, which can be thousands, in a single device and are typically used for extracellular recordings. These

multi-site electrodes can have diverse sizes, shapes and configurations. Moreover, within the subcategory of implantable multi-site probes that can be used for *in vivo* experiments, they can be classified according to their building materials [Cheung, 2007].

Tunsgten or steel microwire multi-site electrodes constitute the first group of implantable probes and are typically used to triangulate the exact position of specific neurons. Tetrodes are a common example of such devices and contain, as their own name imply, four electrodes in a single wire, which usually has a diameter of 30 um [Guerrero et al., 2018]. Another one of these groups is conformed by silicon-based probes, which can be configured into Utah or Michigan arrays. Utah arrays dispose their electrodes over a square shaped surface, only separated some micrometers from one another, and they just acquire signals with their tips [Leber et al., 2019]. Michigan type arrays have microelectrodes along all the surface of their shanks, enabling a higher channel density with a great spatial resolution [Wise and Angell, 1975, Buzsáki et al., 2015]. The linear layout of the electrodes along each shank allows to capture neural activity across one or several brain regions, obtaining a sorted representation of it through the different channels. For this project, high-density Michigan arrays with several shanks were employed.

Additionally, we also conducted experiments utilizing Neuropixels probes, a recent innovation in electrophysiology acquisition devices that incorporates 960 microelectrodes, from which 384 can be simultaneously recorded. These are distributed along a 10mm long unique shank, enabling to pierce the whole hippocampus from side to side and register its activity through all its layers, both in the ventral and dorsal sections, concurrently. Microelectrodes are separated 20um in the vertical axis and 70um in the horizontal one, thus obtaining ultra-dense recordings [Jun et al., 2017].

Furthermore, in previous projects we always used single-electrode intracellular electrical stimulation to alter the neurons and circuits behaviour. However, in this last work we employed optogenetics as real-time stimulation mechanism. This technique utilizes genetic engineering in order to incorporate genes that codify light-sensitive proteins into specific neurons of the experimental subjects, which are neurons in our case. These genes can be inserted by means of the creation of transgenic animals, viral transduction or transfection. When these proteins receive light in a determined frequency spectrum, they cause changes in the neurons membrane potential, inhibiting or exciting them [Pastrana, 2010, Methods, 2010]. In our experiments, the LEDs that produced the triggering light were integrated in the acquisition probes.

4.2 Materials and methods

4.2.1 Experimental setup

The goal of this work was to perform real-time detection of SWRs using a deep learning based method. The developed tool was trained using data recorded from *in vivo* experiments and then tested in online trials, some of which also included closed-loop disruption of the events. For this reason, various electrophysiology and optogenetics experiments were performed using different mouse lines. These animals were kept at Instituto Cajal facilities in a 12-hours day-night cycle (7a.m. - 7p.m.) and were given water and food at will. All protocols and procedures were performed according to the Spanish legislation (R.D. 1201/2005 and L.32/2007) and the European Communities Council Directive 2003 (2003/65/CE). Experiments were approved by the Ethics Committee of the Instituto Cajal and the Spanish Research Council (CSIC). All animals used and experiments performed are detailed in Table 4.1.

The experiments were conducted under head-fixed conditions since this enabled the recording of individual neuron spikes when using high density probes, which were useful for further analyses. Fixation bars and ground screws were implanted in the animals following a surgery

procedure, under anesthesia. Mice from specific genetic lines, more prone to gene insertion, were selected for optogenetic experiments and injected with a AAV5-DIO-EF1a-hChR2-EYFP plasmid that made them sensitive to optogenetic light stimulation. Once the animals had recovered from the procedure, they were trained for around two weeks to get familiar with a head-fixed scenario. Since SWRs emerge in resting periods after movement, mice were placed on a running wheel or treadmill during the experiments, to which they had to become habituated. After this training, a craniotomy surgery was performed to enable the insertion of electrophysiology probes during the experiments.

High-density LFP signals were captured at a 30kHz sampling rate using 32-channels uLED optoelectrodes (4 shanks of 8-channels and 3 uLED each; Figure 4.8A) and a RHD2000 Intan USB Board running under Open Ephys in Windows 11 [Siegle et al., 2017]. Electrodes were placed in the dorsal CA1 hippocampal region where SWRs are usually located. Neuropixels 1.0 probes were also employed to capture LFP signals that covered several hippocampal regions, both in dorsal and ventral areas. A National Instruments PXIe board with a PXI-Express chassis was used as data acquisition device. These recordings were acquired utilizing SpikeGLX software at a 2500 Hz sampling rate with gain equal to 250 and in external reference mode. From their 966 electrodes, separated 20um in the vertical axis and 70um in the horizontal one forming a checkerboard pattern, just 384 could be used at once.

After Neuropixels experiments, a histological analysis was conducted on each subject animal in order to trace the probe track inside their brain. An open-source software, SHARP-Track[1] [Shamash et al., 2018], was used to match brain regions with the signal of each electrode. The names and acronyms of the regions that appear in this work are the following: Corpus Callosum (cc); Primary Visual Cortex (V1); Stratum Oriens (SO); Stratum Pyramidale (SP); Stratum Radiatum (SR); Stratum Lacunosum-Moleculare (SLM); Molecular Layer Dentate Gyrus (ML); Granular Layer Dentate Gyrus (GCL); Hilus (HIL); Hippocampal Fissure (fiss); Basolateral Amygdala (BLA); Amygdalopiriform transition area (APir); Lateral Posterior Medial Rostral Thalamus (LPMR); Posterior Thalamus (Po); Ventro Posterior Medial Thalamus (VPM); Ventro Posterior Lateral Thalamus (VPL); Lemniscus (Lemn); Ventro Medial Thalamus (VM); Zona Incerta, Dorsal Part (ZID); Zona Incerta, Ventral Part (ZIV).

More information regarding the experimental setup and electrophysiological recordings can be found in [Navas-Olive et al., 2022].

4.2.2 Ground truth and data preparation

To create the ground truth dataset with labelled SWR, an expert electrophysiologist visually checked out and manually annotated all SWR present in the data. The start and the end of each event were also marked in order to facilitate transition to ground truth detection. While there is not an exact definition for this time marks, we defined the SWR start at the first ripple or the sharp-wave onset and the SWR end at the last ripple or when the sharp-wave ended. To annotate the data and validate the events a MATLAB R2019b application was developed.

In order to build and evaluate the CNN we used three separated datasets: training, development and testing (Table 4.1). Training and developments sets were stored in two 3-dimensional matrices, called X and Y. Testing set was just a continuous stream of 8-channel LFP data. Training set was used to fit the CNN model to the data. Development set was employed to evaluate the trained model performance with data not used previously for training, and also while still changing the hyperparameters to optimize the model. Finally, the testing set was used to evaluate the final performance of the model over completely independent data that has not been used neither for training nor development.

[1] https://github.com/petersaj/AP_histology

Matrix X contained LFP signals with 8-channels separated in chunks. The recordings from every shank of the probe containing a SWR were loaded, unless specific shanks were selected manually. For shanks with more than 8 channels, only eight of them were randomly chosen but prioritizing those closer to CA1's Stratum Pyramidale. LFP data was then downsampled to 1250Hz, to reduce computing memory requirements while keeping the signals temporal properties, and normalized using z-score. These recordings were then split into 57.6 seconds chunks, since this is a number that can be divided by 0.032s and 0.0128s. This step was necessary to maintain uniform matrix dimensions even when sessions vary in duration. In order for the CNN to perform properly on continuous data streams it also required that the chunk size was large enough to keep the characteristics of a long duration signal. When a chunk did not contain any SWR event it was discarded, although this did not happen often. The final output of this whole process is a matrix X with shape (n, 72000, 8), where n is the total number of chunks, 72000 is the number of samples per chunk (57.6 seconds sampled at 1250 Hz) and 8 is the number of channels.

Matrix Y had shape (n, 1800, 1) for CNN32 or (n, 4500, 1) for CNN12, where n is again the total number of chunks. It contained the labels associated to each temporal window, of 32ms or 12.8ms in each case, which consisted in a number between 0 and 1 indicating the percentage of window occupied by a SWR. Each chunk was divided in windows of 32ms or 12.8ms, thus resulting in 1800 windows per chunk for CNN32 or 4500 windows per chunk for CNN12.

The training set consisted of 2 sessions from 2 different mice, and the whole dataset (both X and Y matrices) was separated into a train subset (70% of the samples) and test subset (with the remaining 30%) to evaluate the performance during training. Development set consisted of 15 sessions from 5 different animals that were not used for training. Test set was composed of 6 sessions from 3 different mice (Table 4.1). Four additional testing sessions from 4 animals were used in Neuropixels trials.

4.2.3 Artificial neural network implementation and specifications

All code related to the neural networks was programmed using Python 3.7.9 with libraries Numpy 1.18.5, Scipy 1.5.4, Pandas 1.1.4 and H5Py 2.10.0. To build, train and test the network, we used Tensorflow 2.3.1, an open-source library implemented by Google for machine learning and artificial intelligence purposes and compatible with several programming languages, like Python and C. Tensorflow is a very powerful tool, but at the same time it is often difficult to work with. For this reason we also utilized Keras 2.4.0, a Python API for deep learning backends such as Tensorflow, which provides a more user-friendly interface to handle the libraries.

Training and offline validation of the CNN was performed over the Artemisa high performance computing infrastructure[2]. This consisted in 23 machines equipped with 4 NVIDIA Volta V100 GPUs. Data was analyzed on personal computers (Intel Xeon E3 v5 processor with 64GB RAM and Ubuntu v.20.04).

Our CNN architecture basic building block consisted on a 1D-Convolutional Layer [Cun et al., 1990] followed by one Batch Normalization Layer [Ioffe and Szegedy, 2015] and one Leaky ReLU Activation Layer [Maas et al., 2013]. The final design consisted in seven of these blocks consecutively and a final Dense layer with a sigmoid activation function [Rosenblatt, 1958], thus having a total of 22 layers (Figure 4.6).

Data processing was performed by the one-dimensional Convolutional layers (tf.keras.layers.Conv1D), which would look for specific features in the data. The exact number of kernels for these layers was selected after a preliminary parametric search for the initial learning rate, number of kernels factor and batch size (Figure 4.9A). Convolutional kernels are weight matrices

[2] https://artemisa.ific.uv.es/web/content/nvidia-tesla-volta-v100-sxm2

Animal	Session	Duration (s)	#ripples	
Training				
Amigo2_1	hippo_2019-07-11_11-57-07_1150um	2398.86	1309	
Som_2	hippo_2019-07-24_12-01-49_1530um	1036.25	485	
Offline validation (Development)				
Thy7	2020-11-11_16-05-00	744.21	1064	
Thy7	2020-11-11_16-21-15	763.67	926	
Thy7	2020-11-11_16-35-43	701.99	656	
Thy1GCam1	2020-12-18_13-16-03	708.92	301	
Thy1GCam1	2020-12-18_13-32-27	669.15	412	
Thy1GCam1	2020-12-18_14-40-16	613.75	245	
Thy1GCam1	2020-12-18_14-56-54	725.4	237	
Thy1GCam1	2020-12-21_14-58-51	630.61	115	
Thy1GCam1	2020-12-21_15-11-32	651.16	159	
Thy1GCam1	2020-12-21_15-26-01	682.22	165	
Calb20	2021-01-22_13-08-20	1203.23	412	
Dlx1	2021-02-12_12-24-56	1200.67	254	
Dlx1	2021-02-12_12-46-54	1021.34	211	
Thy9	2021-03-16_12-10-32	1516.65	264	
Thy9	2021-03-16_14-31-51	1201.19	274	
Online validation (Test)				
PV6	2021-04-19_14-02-31	1051.75	422	
PV7xChR2	2021-05-18_13-08-23	958.53	80	
PV7xChR2	2021-05-18_13-24-33	855.42	121	
PV7xChR2	2021-05-18_13-08-23	958.53	88	
Thy10	2021-06-01_13-28-27	626.62	318	
Thy10	2021-06-15_15-28-56	976.90	566	
Neuropixels				
Calb	28Jul_g0_imec0	754.28	166	
Thy1	01Jul_g0_imec0	643.60	709	
Thy1	15Jul_g0_imec0	690.95	124	
Thy1	16Jun_g0_imec0	655.70	625	
External database				
			Original	**Validated**
Achilles	Achilles_10252013 (first 30 min chunk)	1800.00	223	423
Achilles	Achilles_10252013 (fifth 30 min chunk)	1800.00	470	821
Achilles	Achilles_11012013 (first 30 min chunk)	1800.00	569	283
Cicero	Cicero_09012014 (first 30 min chunk)	1800.00	264	133
Cicero	Cicero_09012014 (third 30 min chunk)	1800.00	515	743

Table 4.1: Datasets utilized for training, offline validation (development) and online validation (testing) of the developed deep learning model. External data [Grosmark and Buzsáki, 2016] and Neuropixels data were used for testing.

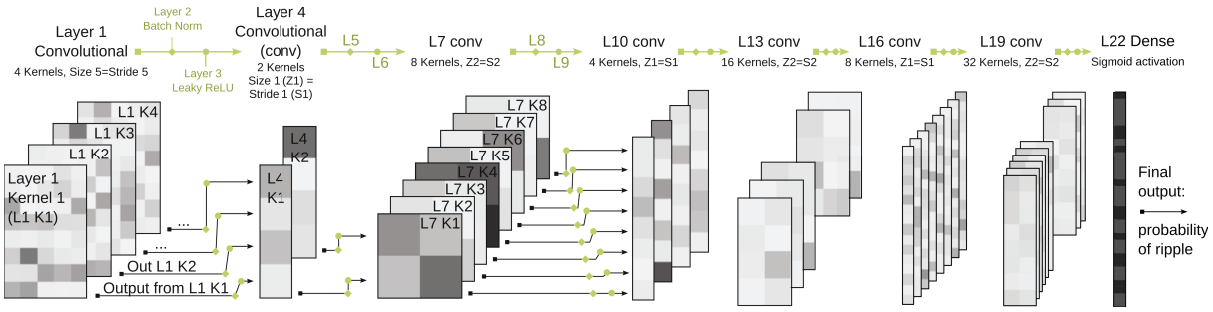

Figure 4.6: CNN-ripple architecture diagram. It consists of seven blocks formed by a 1D-CNN layer, that processes the data, followed by a BatchNorm layer and a LeakyReLU activation function. After all these blocks a Dense layer with a sigmoid activation function produces a probability value between 0 and 1. This probability represents the confidence of the model when predicting the existence of a SWR in the analyzed window, being 1 the highest. A probability threshold is then used to determine at which confidence value we consider the predictions as SWR. Figure adapted from [Navas-Olive et al., 2022].

which are used to perform a convolution operation over the layer input data (element-wise dot product), resulting in a kernel activation signal. Since each kernel generates its own activation signal, every convolutional layer will produce as many signals as its number of kernels. Batch Normalization layers (tf.keras.layers.BatchNormalization) normalized the output from Convolutional layers by fixing its variances and means. Leaky ReLU layers (tf.keras.layers.LeakyReLU) are an activation function that convert negative numbers coming from BatchNorm layers into values very close to 0. These two types of layers were included to provide stability and robustness to the whole network. Lastly, the final Dense layer (tf.keras.layers.Dense) fitted the output data to the desired output dimensions (i.e., probability values).

Parameters for Batch Normalization layers were kept as the default values specified in the Tensorflow 2.3.1 library. Alpha parameter for the Leaky ReLU activation function was set to 0.1. In order to have a neural network that operated equivalently in both offline and online scenarios, the Convolutional layers kernel size and stride were set at them same value. These two parameters shape the network's input window duration, so they were tuned to produce either a 32ms-input network (CNN32) or a 12.8ms-input one (CNN12). For CNN32 kernel size and stride for Convolutional layers 1, 4, 7, 10, 13, 16 and 19 were: 5, 1, 2, 1, 2, 1, 2, respectively. In the case of CNN12, for the same layers, the values were 2, 1, 2, 1, 2, 1 and 2. Convolutional layers with increased stride have the same effect than Max-pooling layers, so we did not include the later to avoid problems with the input window duration [Springenberg et al., 2014]. A sigmoid activation function was used in the Dense layer to create a probability value between 0 and 1 as output. All remaining parameters for Dense and Convolutional layers were kept to the default values.

Optimizer, regularization and decay were established after an extended parametric search (Figure 4.9C). Adam optimizer algorithm was selected, with an initial learning rate of 0.001, beta_1 = 0.9, beta_2 = 0.999 and epsilon = 1e-07 [Kingma and Ba, 2014]. Number of kernels for Convolutional layers 1, 4, 7, 10, 13, 16 and 19 was set at 4, 2, 8, 4, 16, 8 and 32, respectively. Batch size was established to 16. To avoid overfitting we used the L2 regularization method with a 0.0005 value. Learning decay rate was not employed.

4.2.4 CNN training, development and testing

CNN model was trained for 3000 epochs using Binary Cross-Entropy as loss function (see Equation 4.1), where N is the number of windows, y_i is the label of window i, and $p(y_i)$ is the probability predicted for window i. We chose this loss function since our network returned probability values between 0 and 1 and a Cross-Entropy equation compares this kind of outputs with their expected ground truth labels. Additionally, it includes a logarithmic term that penalizes the cost depending on how different is the predicted probability compared to the expected one [Murphy, 2012]. Since our model only has two classes, window with SWR or without SWR, we used the binary version of the function. Despite having a very imbalanced dataset, where only a tiny percentage of windows actually contain an event, the network design compensated this efficiently and we did not need to use weights for each class in the loss function.

$$H_p(q) = \frac{-1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{4.1}$$

Predictions made by the model for each window were considered as positive or negative using a probability threshold that was manually established: a predicted probability was considered positive if its value was above or equal to the threshold, or negative otherwise. With this in mind we could classify the predictions in four groups: True Positive (TP) when the prediction was positive and the ground truth window did contain a SWR event; False Positive (FP) when the prediction was positive in a window that did not contain any SWR; False Negative (FN) when the prediction was negative but the window contained a SWR; and True Negative (TN) when the prediction was negative and the window did not contain any SWR event.

To classify predictions into those categories we used the metric known as Intersection over Union (IOU). This value is calculated as the division between the intersection (overlapping) of two windows by the union of them (see Equation 4.2). When the IOU value for two windows was equal or greater than 0.1 then they were considered to match. A positive prediction was acknowledged as a TP when it matched any window containing a SWR, otherwise it was a FP. Any true event that did not have a matching positive prediction was classified as FN, while all negative predictions with no matching true events were considered TN.

$$IOU = \frac{window_1 \cap window_2}{window_1 \cup window_2} \tag{4.2}$$

In order to evaluate the performance of the model, there are three metrics that rely in the classification of true and predicted events into these previous four groups. Precision (P) measures the percentage of predictions that were correct, and is computed as the division of the total number of TPs between the sum of TPs and FPs, in other words, the rate of predictions that were accurate. Recall (R) measures the percentage of true events that were detected by the model, and is calculated as the number of TPs divided by the sum of TPs and FNs, which can also be defined as the rate of true events detected by the model. Finally, F1 score is the harmonic mean of precision and recall and can be seen as the overall model performance (Figure 4.7).

The choice of the probability threshold value is of great importance since it determines the performance of the detection method. When validating the model offline, we used two thresholds to evaluate predictions. A first upper threshold was used to decide if a prediction was positive or negative, as mentioned before. If it was positive, then a second lower threshold was employed to determine the limits of the SWR more accurately. During offline validation all combinations of possible thresholds were tested out for each session, selecting the pair that obtained the best

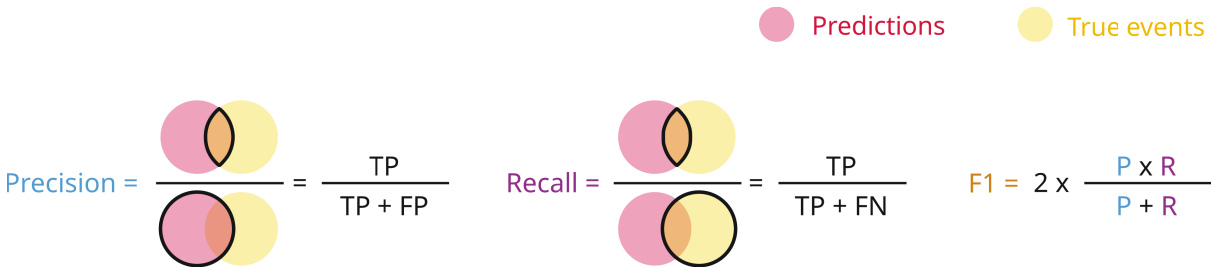

Figure 4.7: Graphical representation and formulas for the three metrics employed: Precision, Recall and F1 score.

possible F1. Possible values for the upper threshold were 0.80, 0.75, 0.70, 0.65, 0.60, 0.55, 0.50, 0.45, 0.40, 0.35, 0.30, 0.25, 0.20, 0.15 and 0.10, while for the lower threshold were 0.80, 0.70, 0.60, 0.50, 0.40, 0.30, 0.20 and 0.10. Only the upper threshold could be used during online detection due to the real time constrains and it was set manually to a fixed value at the beginning of each experiment following the experimenter judgement.

The metric used to measure the delay among SWR true events and TP predictions was the time between the true SWR peak to the prediction beginning. To define SWR peaks, the most relevant LFP channel (usually the one located in Stratum Pyramidale) was filtered using a third-order Butterworth bandpass filter between 70 and 250 Hz. This filtered signal was put through another four-order Savitzky-Golay filter and smoothed twice with windows of 3 and 6.5 ms to obtain the signal envelope. The maximal value of the envelope signal within the temporal boundaries of an event was considered the SWR peak.

4.2.5 Offline detection of SWR events using spectral filters

As mentioned in previous section (see Figure 4.1.3), SWR detection is typically carried out with methods based on spectral filters. In this work we used a bandpass second-order Butterworth filter (100-300 Hz passband), such as the one included in the Open Ephys GUI software, as the gold standard against which to compare the developed CNN models in both offline and online contexts.

For offline validation we followed a two-thresholds approach similar as the one used for the CNN detections. The signal was filtered between 100 and 300 Hz using a *butter* and non-casual *filtfilt* filter to void phase lags. The resulting filtered signal was then amplified twice, filtered by a fourth-order Savitzky-Golay filter, and smoothed by two consecutive *movmean* sliding windows (2.3 and 6.7 ms) to obtain its envelope. All this operations were computed in MATLAB R2019b.

When the envelope surpassed both the first lower threshold, which was used to define the starting and ending point of the event, and the second upper threshold, then the window between those points was considered as a detection. If two detections were less than 15ms apart from each other then were merged into one. When using this kind of SWR detection methods the choice of threshold values has a great impact in the performance. In order to find the pair of thresholds that maximized the F1 score we tested of possible combination for each separate session. Lower threshold ranged from 1, 1.5, 2, 2.5 times the envelope standard deviation, and the upper threshold from 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10 times the envelope standard deviation (resulting in a total of 60 threshold combinations).

For online detection only the upper threshold was used, again because of the real-time constrains, being its value also chosen by the expert.

4.2.6 Open Ephys custom plugins for online detection

Online validation was performed using Open Ephys 0.5.5.3 GUI software in Windows 11 with a RHD2000 Intan USB Board. This application allows the users to write their own C++ plugins to be included in the processing chain. We developed two custom plugins to detect SWR using both spectral methods and CNNs. The plugin that operated the CNN used the Tensorflow 2.3.0 API for C https://www.tensorflow.org/install/lang_c.

4.2.7 SWR intervention in closed-loop optogenetics experiments

In order to detect and intervene SWR in real-time two different setups were designed for closed-loop optogenetics experiments. In both cases an OSC1Lite driver from NeuroLight Technologies was employed to handle the uLED integrated in the optoelectrodes. When an event was detected, the uLED would emit blue light stimulation at 10-20uW, which activated cell-type specific ChR2. Specificity of viral expression and localization of probe tracks were histologically assessed after experiments. In the first setup, Open Ephys 0.5.5.3 GUI running over Windows 11 was used as recording and processing software, connected via USB 2.0 with a RHD2000 Intan acquisition board. Detection was carried out with the CNN plugin mentioned before and an Arduino Nano ATmega328 board, again connected via USB 2.0, was used to generate the output. This output signal controlled the OSC1Lite driver.

For the second setup we used Intan RHX Data Acquisition Software for recording the signals, but also to redirect 8 channels of the LFP signals to the analog output ports of the Intan board. The resulting signals were in a range of -3.3 to 3.3V range, so they went through a level shifter electronic circuit (one per channel) to re-scale them to a 0 - 3.3V range. Afterwards, the 8 channels were introduced in a MCP3008 analog-to-digital converter (ADC) and sent to a Raspberry Pi 4 Model B single-board computer via SPI. This Raspberry Pi, running a Raspberry Pi OS Kernel version 5.10 with desktop, would execute a stand-alone version of the CNN plugin developed for Open Ephys. This code was programmed in C++ and used Tensorflow Lite 2.7.0 for C to handle the CNN model, since this version of the API was optimized to run on microcontrollers and small computers. One of the digital output pins of the Raspberry Pi was used to generate the output signal, which again controlled the OSC1Lite driver. Pigpio library for C was utilized to read and write from the Raspberry Pi pins inside the code (https://abyz.me.uk/rpi/pigpio/cif.html)

The eight level shifter circuits and the ADC were integrated on a single printed circuit board (PCB) for simplicity and robustness of the implementation. The whole circuit schematic and PCB layout was designed using KiCad 5.1.12 open-source software (https://www.kicad.org). The components required to build the circuit were, with their corresponding Manufacturer Part Numbers: sixteen 20kΩ resistors (MCWR12X2002FTL), twenty four 10kΩ resistors (CRCW120610K0FKTA), eight 300Ω resistors (RC1206FR-07300RL), eight 1000pF capacitors (C0603C102F3GACTU), eight 100pF capacitors (C0603C101F5GACTU), sixteen 0.1µF capacitors (C1206C104F3GACTU), eight LMC6482 operational amplifiers (LMC6482IN_NOPB) and one MCP3008 ADC (MCP3008-I/P).

4.2.8 Measuring closed-loop system latencies

As explained in the previous chapters of this work, every closed-loop setup has to comply with certain temporal constrains, which may be less or more strict depending on the task. For example, the hybrid circuits built in Chapter 2 needed to maintain a constant 10kHz acquisition and stimulation rate, therefore hard real-time technologies were employed to always carry out all the required operations within a 100us interval. On the other hand, when dealing with

Computer	OS	Software	Acquisition system	Output device
Intel Core i7-8550U, 8GB	Windows 11	Open Ephys GUI 0.5.5.3	RHD2000 Intan	Arduino Nano ATmega328
Intel Core i9-7920X, 128GB	Windows 11	Open Ephys GUI 0.5.5.3	NIDAQ + BNC-2110	Arduino Nano ATmega328
Intel Core i9-7920X, 128GB	Windows 11	Open Ephys GUI 0.5.5.3	Neuropixels 1.0	Arduino Nano ATmega328
Raspberry Pi 4 Model B	Raspberry OS Desktop	Custom C++ program	Raspberry Pi GPIO pin	Raspberry Pi GPIO pin

Table 4.2: Closed-loop setups tested for SWR detection and intervention.

SWR detection, and since these events last for several milliseconds, soft real-time restrictions are enough. Using soft real-time technologies significantly simplifies the technical requirements for the closed-loop system design and implementation. However, it is still of great importance to test the computational latencies of every possible closed-loop setup in order to minimize the detection and intervention delays.

Latency tests for every setup utilized in this project were conducted in order to compare the performance of all the different configurations. Following the guidelines proposed by Open Ephys developers in https://open-ephys.github.io/gui-docs/Tutorials/Closed-Loop-Latency.html we used an external device to generate an analog squared-wave. The rising edges of these signals were detected with a crossing detection routine and a digital pulse was produced as a result. The time difference of the output pulse regarding the rising edge was measured and considered as the system latency. Two trials of three minutes were performed for each system.

Open Ephys GUI 0.5.5.3 was tested in combination with a RHD2000 Intan USB board, as well as National Instruments and Neuropixels acquisition devices. All trials were conducted using a 5.3ms buffer size on Windows 11. In all three cases, a Crossing Detector plugin was employed, followed by a Record Node and an Arduino Output. The output signal was generated using an Arduino Nano ATmega328 board connected via USB 2.0 and registered again by the acquisition system for posterior analysis. Raspberry Pi detection system utilized a RHD2000 Intan USB board to read both analog input and digital output signals. A custom made crossing detector program was executed and the resulting pulse was generated through a GPIO pin. Table 4.2 summarizes these configurations.

4.2.9 Quantification and statistical analysis

Python 3.8.5 and/or MATLAB R2019b were used for statistical analysis. Sample sizes were not predetermine by any statistical method and were the same to those reported elsewhere. Kolmogorov-Smirnov and Levene's tests were utilized to confirm, respectively, normality and homoscedasticity. Number of repetitions for each test is specified in the text and/or figures. Neural networks input data was z-scored so it was comparable across animals and sessions.

4.3 Results

The final goal of this project was to develop a closed-loop system capable of real-time intervention of SWR. However, in order to reach that objective we first needed to design and implement our novel deep learning-based detection method and validate its proper functioning in both offline and online scenarios. This section describes the whole process.

4.3.1 CNN architecture design

In order to determine the probability of a SWR existing in a specific window of data in real-time we designed and implemented a convolutional neural network. Input data consisted in 8-channel LFP signals recorded *in vivo* using a silicon probe, whose electrodes were distributed along various layers of the hippocampal CA1 subregion (Figure 4.8A). Duration of the windows was initially fixed at 32ms since this is the mean duration of a SWR plus one standard deviation. CNNs were chosen as the working paradigm due to, among other reasons, the little amount of data pre-processing that they require to reach a good performance, which is very convenient when implementing a method that needs to work in real-time. In particular, the neural network architecture was designed taking YOLOv2 as inspiration, a network developed for object recognition and classification in images. This algorithm is able of identifying a wide range of objects in video frames that receives in real-time, so it seemed like a reasonable starting point for a neural network that should detect SWR in LFP signals also in real-time.

Since YOLO architecture was originally designed to process a stream of two-dimensional images (width x height), with a third dimension to represent the color, we needed to adapt some of its elements so they were suitable to work with one-dimensional signals (time), plus a second dimension representing each LFP channel. In this manner, if YOLOv2 input data had dimensions (n, w, h, c), with n being the number of images, w the width of each of them, h their height and c the number of colors (e.g., 3 when the image has color codified in RGB format or 1 when it is in black and white), our network's input data should have dimensions (n, l, c), where n is the number of chunks in which the whole signal was divided, l is the number of samples (sampled at 1250Hz) in every chunk and c is the number of channels, which was 8 in our case (see Section 4.2.2).

In addition to this reduction of the network's dimensionality, we also decided to decrease and simplify the number of layers in comparison to YOLOv2, considering that the original algorithm considers a great number of them and this entails large computational costs and processing times, as well as a higher difficulty to properly train them all. Inspired by YOLOv2 architecture, we designed our network using three-layers blocks: first a 1D-convolutional layer (CNN1D), followed by a normalization layer (BatchNorm) and finally an activation function layer (LeakyReLU). The convolutional layer was in charge of processing the data and used a determined number of kernels to do so, each one of them getting specialized in looking for a particular feature in the data. A kernel is a weights matrix that gets updated on every epoch of the training process, and it is used to perform a convolution operation over the data received by the layer. The result of such operation is a kernel activation signal, obtaining one for each kernel of the layer. For each chunk of data, the analysis window will move forward over the temporal axis by jumping the number of samples indicated by the stride parameter, which was established to a similar value as the kernel length so the behaviour of the network was identical both in offline and online scenarios (Figure 4.8B). With this in mind, we decided to implement two versions of the CNN: the original one with input windows of 32ms (CNN32), which favoured the study of the network internal functioning, and an alternative one with 12.8ms windows (CNN12). This facilitated real-time detection with low latencies since the method needs to wait until the whole window is elapsed to make a prediction, so a shorter window duration means less delay regarding the real event. CNN12 only required to change the first kernel length to be able to evaluate windows of less than half the previous duration, keeping all the other parameters and hyperparameters the same (see Section 4.2.3).

Convolutional layer's output was then inserted in the BatchNorm layer, which normalized the data to provide stability and robustness to the whole network. To end the block, normalized data was inputted to the LeakyReLU layer, that played a similar role as the previous one, but this time by converting negative values to numbers close to zero. For example, the first layer of

our network (L1) received as input a window of shape (length x 8-channels), being length the number of samples in that window and having the recorded LFP eight channels. L1 generated four kernel activation signals as output, one for each kernel (e.g., L1K1, L1K2, etc.), that were then sent to the BatchNorm layer (L2) and the LeakyReLU layer (L3). Afterwards, the resulting signals went into the next block with layers L4, L5 and L6, and this process was repeated until the data went through all blocks (Figure 4.8C, D). Due to this, each layer's kernel dimensions were determined by the shape of the previous layer output. On each block, the length of its kernels was shorter than in the previous one. Again based on YOLOv2, to force an interspersed convolution in the spatial (i.e., channels) and temporal axes we alternated blocks with half the number of kernels than the preceding one and kernel length equal to 1. With all this in mind, we chose the kernels dimensions and number of blocks that optimized real-time processing of the input signals, reducing computation speed while obtaining an optimal performance, obtaining a total of 7 blocks and, therefore, 21 layers.

The final layer of the network, after all those three-layers blocks described beforehand, was a Dense layer with a sigmoid activation function that returned a number between 0 and 1 as output, representing the probability of a SWR existing in the input window data. To identify the events, the user establishes a probability threshold that determines the value over which the CNN output is considered as a SWR detection (Figure 4.8E). Our final CNN architecture comprised seven blocks plus the Dense layer, thus containing a total of 22 layers (Figure 4.6).

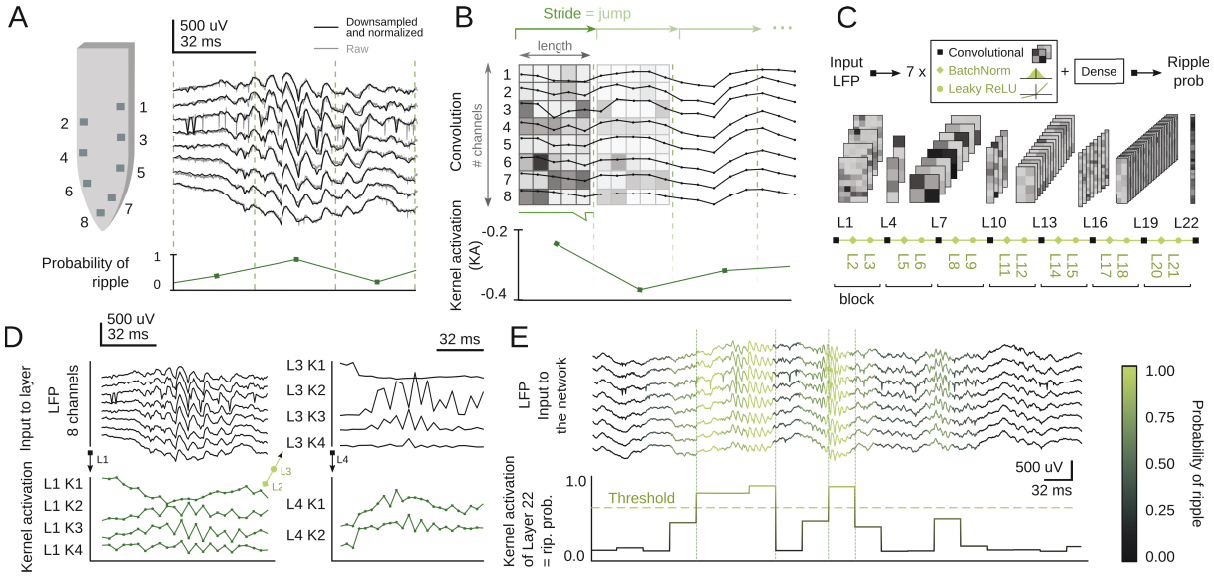

Figure 4.8: CNN-ripple architecture and functioning. **A:** example of a SWR captured with 8-channel silicon probes in the CA1 hippocampal region of a mouse. These LFP signals were divided into windows (32ms) and fed to the CNN, which returned the probability of a ripple existing in them. **B:** example of a convolutional operation between layer's L1 kernel and the input data, producing a kernel activation signal (KA). **C:** overview of the CNN architecture, comprised by seven Convolutional + BatchNorm + Leaky ReLU blocks and a Dense layer. **D:** example of the KA signals generated through the first 4 layers of the CNN over some input data, transforming an 8-channel LFP into two signals. **E:** example of the CNN output at 32ms resolution. The returned values represented the probability of a ripple existing in each window. A probability threshold is used to determine above which confidence value the CNN is considered to have detected an event. Figure adapted from [Navas-Olive et al., 2022].

Once the network's architecture was designed and implemented, we proceeded to train it using a dataset manually labelled by an expert, where the starting and ending time for every

event was indicated. This dataset contained 1794 labelled SWRs recorded in two different experimental sessions, each one of them on a different mouse (Table 4.1).

There were a great number of parameters and hyperparameters in our neural network that could be tuned to optimize its behaviour. Since testing one by one all possible combinations of them was unreasonable, two automatic parametric searches were performed in order to find the combination that maximized its performance. In a preliminary search, the proposed CNN architecture was compared with another that incorporated two additional LSTM layers before the Dense one. At the same time, several values for the initial learning rate, batch size and kernel number factor were tested (Figure 4.9A). For this first search both the train and test subsets were created from the complete train dataset. All networks used an Adam optimizer with epsilon equal to 10^{-7}, a L2 regularizer with its value set to 0.0005 and no learning rate decay. From the resulting 107 variations we selected the one that reached the best F1 score while displaying a stable behaviour during training (Figure 4.9B). CNN32+LSTM architecture obtained similar results to CNN32, but took a much longer time to train. Loss function evolution curves over both the training and testing epochs showed great stability for the selected network performance. Later on, a second and more extensive search was carried out, taking into account various more hyperparameters for testing (architecture, initial learning rate, number of kernels factor, batch size, optimizer, optimizer epsilon, regularizer, regularizer value and decay). In this case, the network was trained with the train dataset and validated with the development set (Table 4.1). 781 combinations were tested and we ratified that the configuration selected after the initial search was still among the thirty ones with best performance (Figure 4.9E). A comparison of the loss value evolution during training and validation epochs was conducted in order to confirm the absence of overfitting in the final model (Figure 4.9C). A similar parametric search was carried out for the Butterworth filter parameters, testing different combinations for its order, low and high frequency cuts. A filter with order 2 and a 100-300Hz frequency band was selected, since this configuration obtained a maximum F1 score.

4.3.2 Offline performance

Once the neural network model was trained with the training dataset, we proceeded to evaluate its offline performance on a different group of datasets, completely independent from the training one. To measure the performance three metrics were employed: Precision, which represents the proportion of correct detections over the total number of detections; Recall, which describes the proportion of ground truth events that have been detected by the method; and F1, which is the harmonic mean of the precision and recall (see Section 4.2.4). We compared the performance of the two versions of the network, CNN32 and CNN12, as well as a traditional spectral method based in a Butterworth filter (see Section 4.2.5). As mentioned in the previous section, a threshold was used to determine the value over which the probability returned by the CNN was enough to consider that there was a SWR in a data window. Similarly, in the case of the spectral method another threshold was utilized to indicate the amplitude value of the filtered signal over which an event was said to be detected.

In the first place, we validated the behaviour of the CNNs on the development dataset, that contained 5695 labelled SWRs from 15 sessions recorded in 5 different mice, all of them totally distinct from the training dataset (Table 4.1). CNNs were executed using Keras and Tensorflow on a Python environment (see Section 4.2.3). Butterworth filter was implemented and run on MATLAB (see Section 4.2.5). For all the different methods, the value of the employed metrics depended on the detection thresholds established by the user. A high threshold resulted in few detections but with a great hit rate (high precision, low recall), while a low threshold yielded a lot of predictions, but many of them wrong (low precision, high recall). For this reason we used the harmonic mean of these two values, F1, as the reference metric. For this first validation we

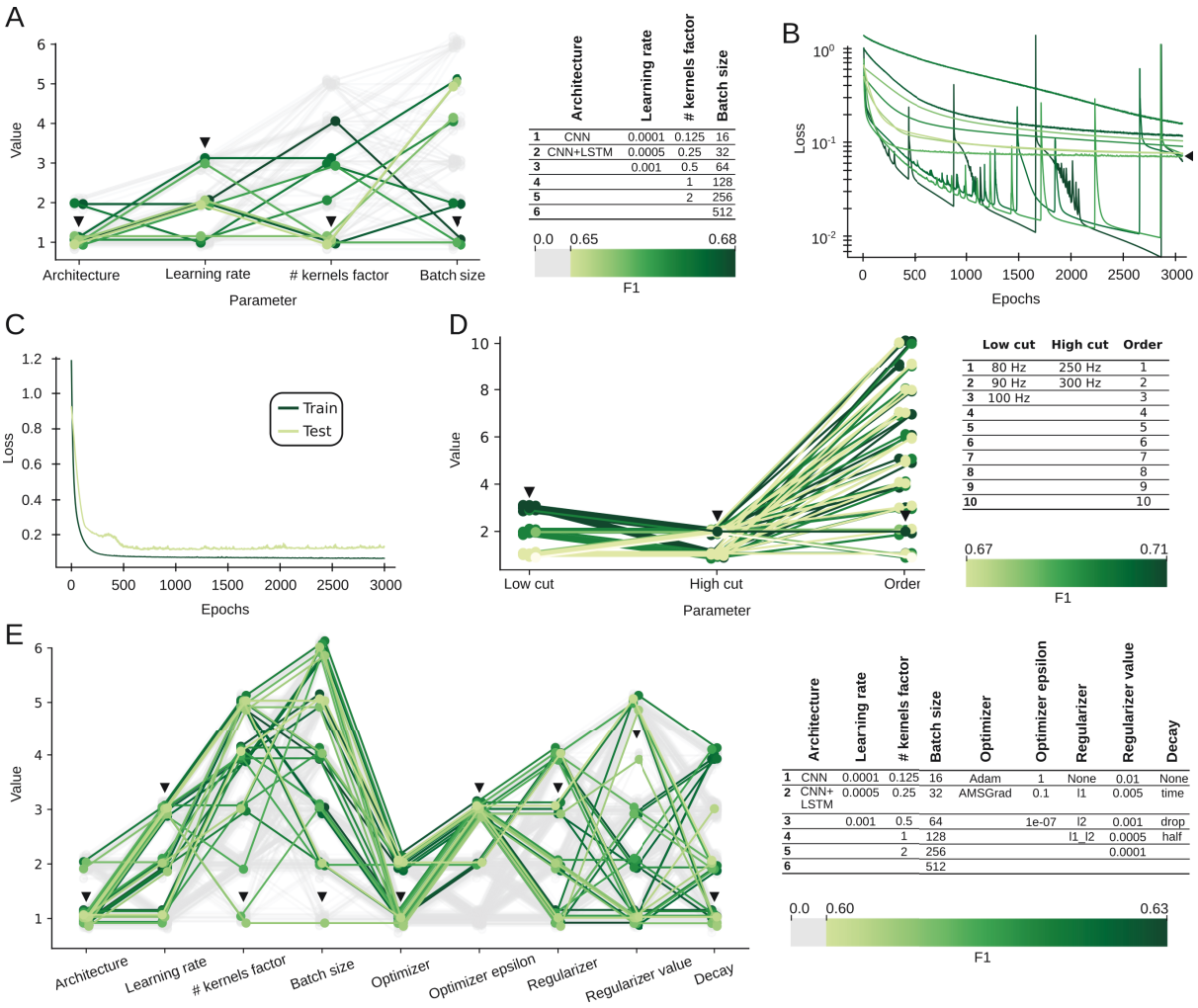

Figure 4.9: Parametric search for CNN-ripple and spectral methods. **A:** preliminary parametric search for the neural network architecture and hyperparameters with 32ms resolution. Two architectures were tested, CNN and LSTM, as well as diverse values for the learning rate, number of kernels factor and batch size. F1 score for the best 10 configurations surpassed 0.65. CNN32 was selected from among these (pointed by arrowheads). **B:** loss value evolution during training for the top-10 models from A. CNN32 was chosen for displaying the most stable and lowest learning curve (arrowhead). **C:** loss value evolution comparison between training and validation for CNN32, showing that there was no overfitting. **D:** parametric search for the Butterworth filter, testing different low and high frequency cuts and orders combinations. **E:** full parametric search, evaluating optimization algorithms and their epsilons, regularizers and their values, learning rate decay strategies and all the parameters from A. 781 models generated, top-30 ones had F1>0.60 (shown in green). CNN32 was among the 30-best ones, its configuration is indicated by arrowheads. Figure adapted from [Navas-Olive et al., 2022].

explored and tested a range of thresholds values, for both networks and the filter, to find the number that maximized the F1 for each session and allowed us to compare the best cases of each method. CNNs thresholds were values between 0 and 1, because they represented a probability (see Section 4.2.4). Filter threshold was defined as a function of the filtered signal's envelope standard deviation (see Section 4.2.5). For this offline analysis, all methods used two thresholds, one to detect the event and the other to delimit its edges.

When the performance of each method was compared using the metrics' values acquired

with the best thresholds for each session we saw that all three approaches obtained similar results, as can be seen in Figure 4.10A (CNN12: F1=0.68 ± 0.05; CNN32: F1=0.63 ± 0.04; Butterworth filter: F1=0.64 ± 0.11). Additionally, another spectral-based method [Dutta et al., 2019] obtained similar results to the CNN (F1=0.66 ± 0.11). We also tested the performance of RippleNET, a deep learning method based on a RNN with convolutional layers that is specialized in detecting SWR in an offline context [Hagen et al., 2021]. Due to this, its behaviour was not as good when the input was a raw continuous signal, just z-scored and downsampled (F1=0.31 ± 0.21; $p<0.00001$ one-way ANOVA for both CNN12 and CNN32). However, when the performance was represented as a Precision-Recall curve for each session and method, which displayed the adjustment between precision and recall for different thresholds, both the CNN12 and CNN32 depicted a better and more robust behaviour than the Butterworth filter (Figure 4.10B).

In an online scenario the best threshold is not known beforehand and must be chosen by the experimenter at the moment by just looking at the incoming data. When comparing the mean F1 value for every session at each possible detection threshold, we observed that both networks displayed a wide range of thresholds at which the F1 was stable and near its maximum. This range was significantly reduced in the filter's case (Figure 4.10C). Hence, selecting an appropriate threshold that optimizes the performance is easier for the CNN than the filter, regardless of the session.

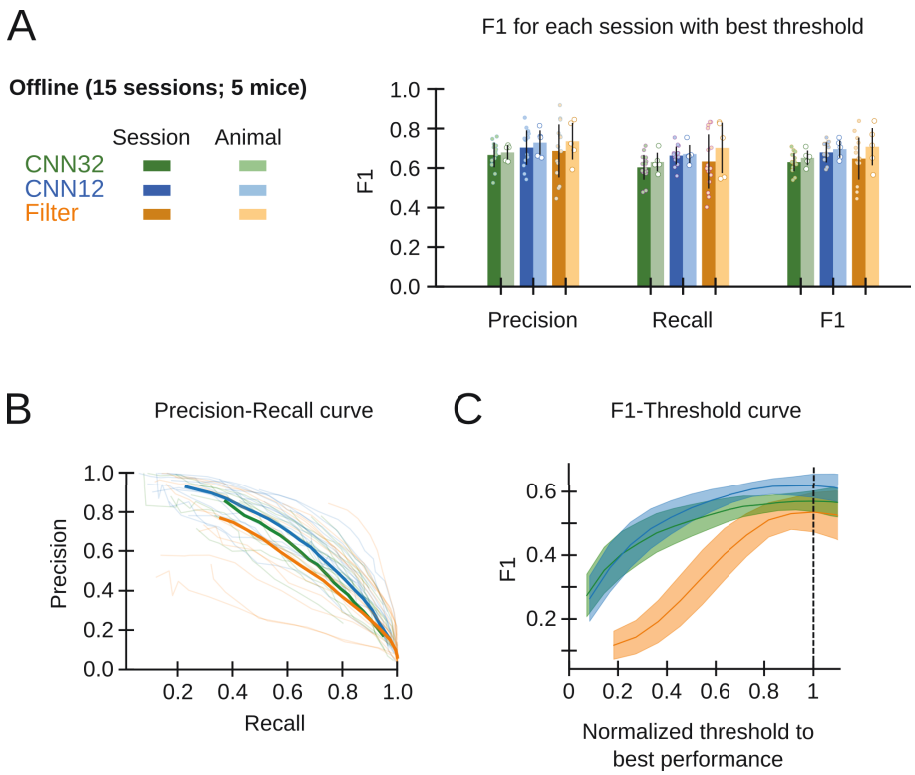

Figure 4.10: CNN-ripple offline performance. **A:** best offline performance of CNN32, CNN12 and filter-based method over the development dataset. Measured as precision (P), recall (R) and F1 score. Results displayed by session (n=15, dark colors) and animals (n=5 light colors). **B:** Precision-Recall curves for each session and method (light) and mean curve for each method (dark). **C:** F1 score for each method when using a range of different detection thresholds on every session. CNN threshold is a probability value while filter threshold is in standard deviations. They are normalized by the best threshold performance to be compared. Results represented as mean ±95% confidence interval. Figure adapted from [Navas-Olive et al., 2022].

4.3.3 Generalization over different databases

In most cases, when neural networks are employed to detect or classify objects there is some general consensus of what and how are the targets that the algorithm must identify. For example, if the method is designed to detect cars in a picture, it would be easy to define what a car is, how does it look like and what are its most common elements. This is in fact a key aspect when utilizing supervised learning algorithms, since their training and performance depend fundamentally on the quality of the training dataset labels. Nevertheless, this is not the case with SWRs, seeing that the events are not always so clearly recognizable by the experts. Moreover, the experts criteria may differ due to diverse reasons, including their previous experience or even the goal of their experiments (e.g., sometimes is preferable to just label the events that are very representative, even if they are few). To evaluate the possible effect that these different personal criteria may have had in the creation of our ground truth, and by extension in the CNN performance, a second expert from the laboratory independently labelled the events of 14 sessions from the development dataset, from 7 different animals.

In this manner, we ended up with a ground truth set containing the events tagged by the original expert (GT original expert), another one with the events identified by the new expert (GT new expert) and the joint set with the events labelled by any of the experts (Consolidated GT). We evaluated then the performance of the CNN models trained in the previous section, as well as the filter-based method, on these fourteen sessions, but differentiating their behaviour regarding the original, new and consolidated ground truths. For both networks, the performance over each expert's GT was similar, with some variations among sessions. However, their behaviour was significantly better when the Consolidated GT was used (one-way ANOVA, CNN12: $F(2)=0.01$, $p=0.026$; CNN32: $F(2)=0.01$, $p=0.013$), meaning that the CNNs were capable of detecting SWRs tagged by both experts, but also several of them that were only labelled by one or the other. On the other hand, the spectral method did not show any improvement with the Consolidated GT (Figure 4.11A). Interestingly enough, when we evaluated the detections made by one expert over the ground truth labelled by the other, we got an F1 score of 0.70 ± 0.13, which was quite similar to the results obtained by the automatic methods (Figure 4.11B).

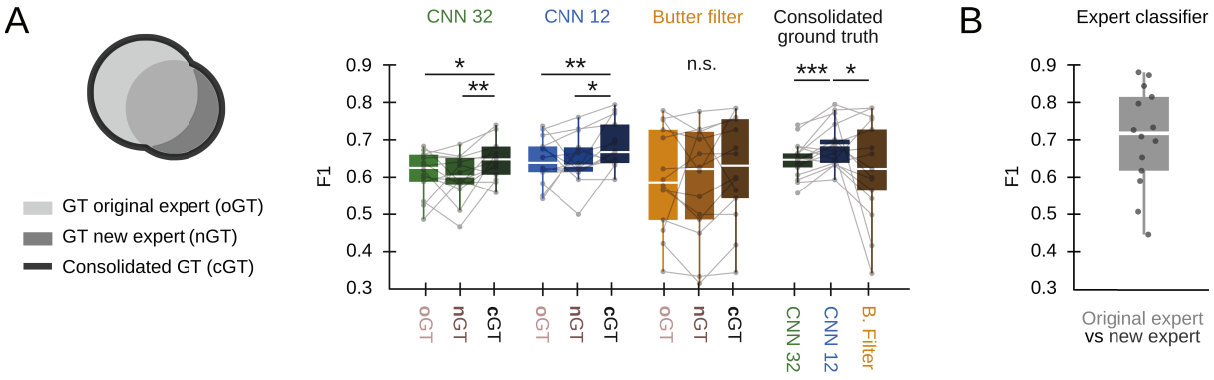

Figure 4.11: Performance over data labelled by different experts. **A:** offline performance of CNN32, CNN12 and filter over the ground truths annotated independently by two experts and the consolidated ground truth (union of the previous two with no repetitions). CNNs improved their behaviour on the Consolidated GT, as opposed to the filter (n=14 sessions; significantly as tested with one-way ANOVA; Paired t-tests *, $p<0.05$; **, $p<0.01$; ***, $p<0.001$). **B:** F1 score of the annotation of one expert over the other's GT. Figure adapted from [Navas-Olive et al., 2022].

Continuing this line of thought, we decided to measure the functioning of the CNNs on data labelled by external experts. For this test we used a public dataset published by other

laboratory [Grosmark and Buzsáki, 2016], available at the CRCNS public repository[3], containing 2041 events distributed in 5 sessions from 2 animals (Table 4.1). These recordings were captured with 10-channels high-density silicon probes on freely moving rats, so on each case 8 channels were randomly selected to match the input dimensions of the network (Figure 4.12A). The event definition used to tag this dataset required the SWRs to have both population synchrony and LFP definition, so the conditions were more restrictive than the ones employed to create our ground truth, which has been used to train the CNN models. For this reason, we observed that the networks were able to find most of the events (CNN32: R=0.75 +- 0.14; CNN12: R=0.80 +- 0.11; Filter: 0.83 +- 0.11), but many of their detections were wrong (CNN32: P=0.42 +- 0.15; CNN12: P=0.49 +- 0.13; Filter: 0.52 +- 0.18). This made sense given the characteristics of the labelled SWRs, so we re-labelled those False Positive events that we considered to be unlabelled SWRs. After this validation process, there was an improvement in precision for all the detection methods, and more significantly for the CNN12 (CNN32: P=0.68 +- 0.14; Filter: 0.77 +- 0.13, $p < 0.05$; CNN12: P=0.87 +- 0.10, $p<0.01$; Figure 4.12B). Recall slightly increased for all methods.

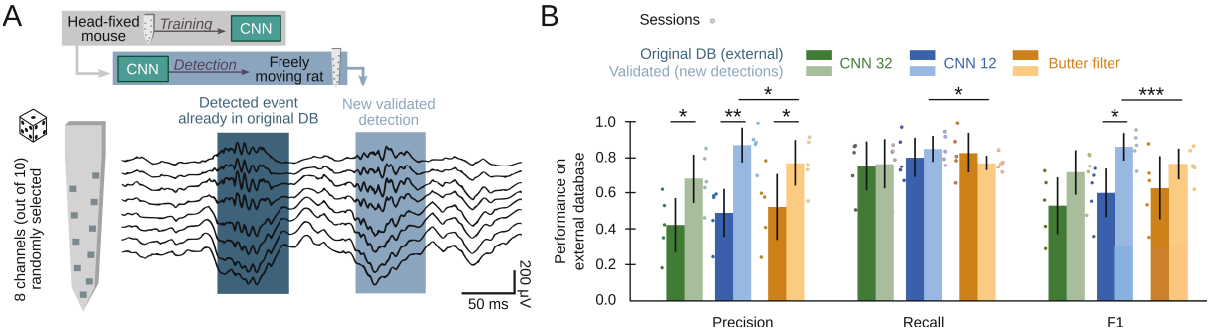

Figure 4.12: Performance over data from external databases. **A:** CNNs generalization capability and effect of the training ground truth on the method functioning was evaluated using an external public database [Grosmark and Buzsáki, 2016]. This data contained 10-channel high-density recordings from the CA1 region of freely moving rats, so 8 channels were randomly selected to match the input dimensions of our neural network. Annotation criteria for this dataset was defined by the existence of both population synchrony and SWR at the same time (dark gray). For this reason, CNN false positives were manually validated and incorporated to the GT if an expert considered them correct SWR (light grey). **B:** performance of CNN32, CNN12 and filter over this external dataset, with both the originally annotated (dark colors) or validated (light colors) ground truths (n=5 sessions, n=2 rats). Figure adapted from [Navas-Olive et al., 2022].

We then decided to take a deeper look into the False Positive events detected by the CNN32 on the development and test sessions (2468 events) and we observed that, in general, they presented patterns and shapes close to those of the True Positives. In particular, there were several cases of sharp-waves without a coupled ripple and some ripples without sharp-wave, as well as some synchronous population firing and artifacts with similar patterns than a SWR. Various events were also found to, despite not belonging to any of the previous categories, display some features typically associated with SWR (Unclassified) (Figure 4.13). These results emphasize the idea that these CNNs are capable, not only of detecting SWRs with a performance equal or even greater than the traditional spectral-based methods, but also of identifying events not initially reflected in the ground truth, showing a great ability to generalize from the training data.

[3] https://crcns.org/data-sets/hc/hc-11/about-hc-11

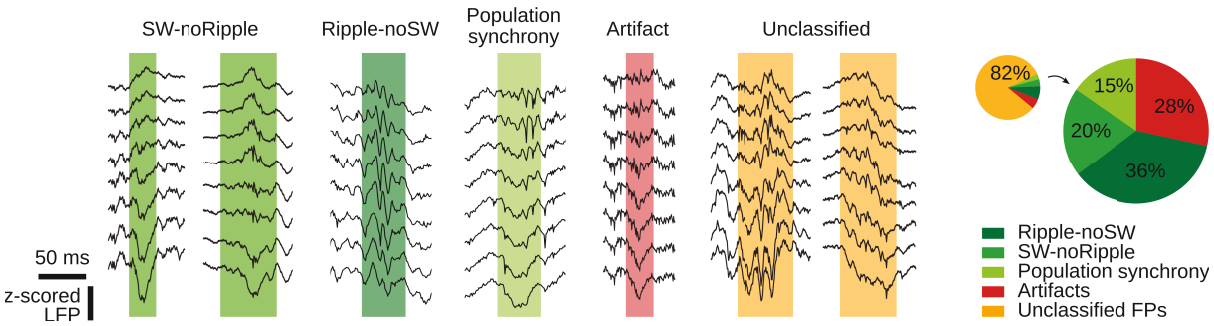

Figure 4.13: Examples of False Positive detections done by the CNN. Some of them are sharp-wave events without an associated ripple and the other way around. Others are the effect of population synchrony or artifacts superimposed to LFP deflections. Some are unclassified events that present some features similar to those of a SWR. Figure adapted from [Navas-Olive et al., 2022].

4.3.4 Validation on data recorded with Neuropixels probes

Even though our neural network model has been trained using recordings from 8-channel high-density probes, we wanted to test its capacity to identify SWR in data acquired with other kinds of probes that could also register the neural activity across various layers of the CA1 hippocampal region. A first trial of this generalization capability was already performed on the data from the external database, obtained utilizing 10-channel probes from which we selected 8 of them randomly as input (Figure 4.12). In this case, a good performance, similar as the one obtained with the test datasets, was achieved by the CNNs. For further exploration of its competence, we applied the neural network on data acquired employing Neuropixels 1.0 probes. These ultra-dense probes record 384 simultaneous LFP channels, that can penetrate through the whole hippocampus from dorsal to ventral, crossing all its layers. Hence, with this technology we were able to not only check the detection methods effectiveness with a distinct acquisition system, but also in different hippocampal regions.

For this purpose, we took recordings from 4 experimental sessions conducted on 4 different mice done with Neuropixels (Table 4.1) and we simulated penetrations similar to those performed with high-density probes by creating groups of 8 channels from the 384 registered. uLED optoelectrode probes have 8 electrodes distributed in the same number of rows, alternating left and right positions with 64µm horizontal and 20 µm separation (Figure 4.8A). To emulate this same pattern with electrodes from a Neuropixels probe we selected one channel from each row alternating between left and right. Since rows are separated vertically 20µm and electrodes on each have an horizontal distance of 70µm, the resulting configuration for the chosen channels was almost similar to that of an uLED probe (Figure 4.14A). For example, a simulated penetration starting from the top of the probe would contain the channels [1 4 5 8 9 12 13 16]. We simulated penetrations across the whole probe to validate the behaviour of the methods on all hippocampal regions and layers, resulting in a total of 96 different simulations. Again starting from the top, the first three of them would be [1 4 5 8 9 12 13 16], [5 8 9 12 13 16 17 20] and [9 12 13 16 17 20 21 24], and so on.

Thanks to an histological analysis performed after the experiment we could determine which brain regions where exactly traversed by the probes and where were each of its electrodes located (Figure 4.14B). An expert manually identified SWRs existing in the CA1 hippocampal region to conform the ground truth that was later employed to evaluate the results. CNN32 behaviour around dorsal CA1 pyramidal layer was just as good as it was during the trials with high-density probes. Performance around ventral SP layer was similar to the achieved in dor-

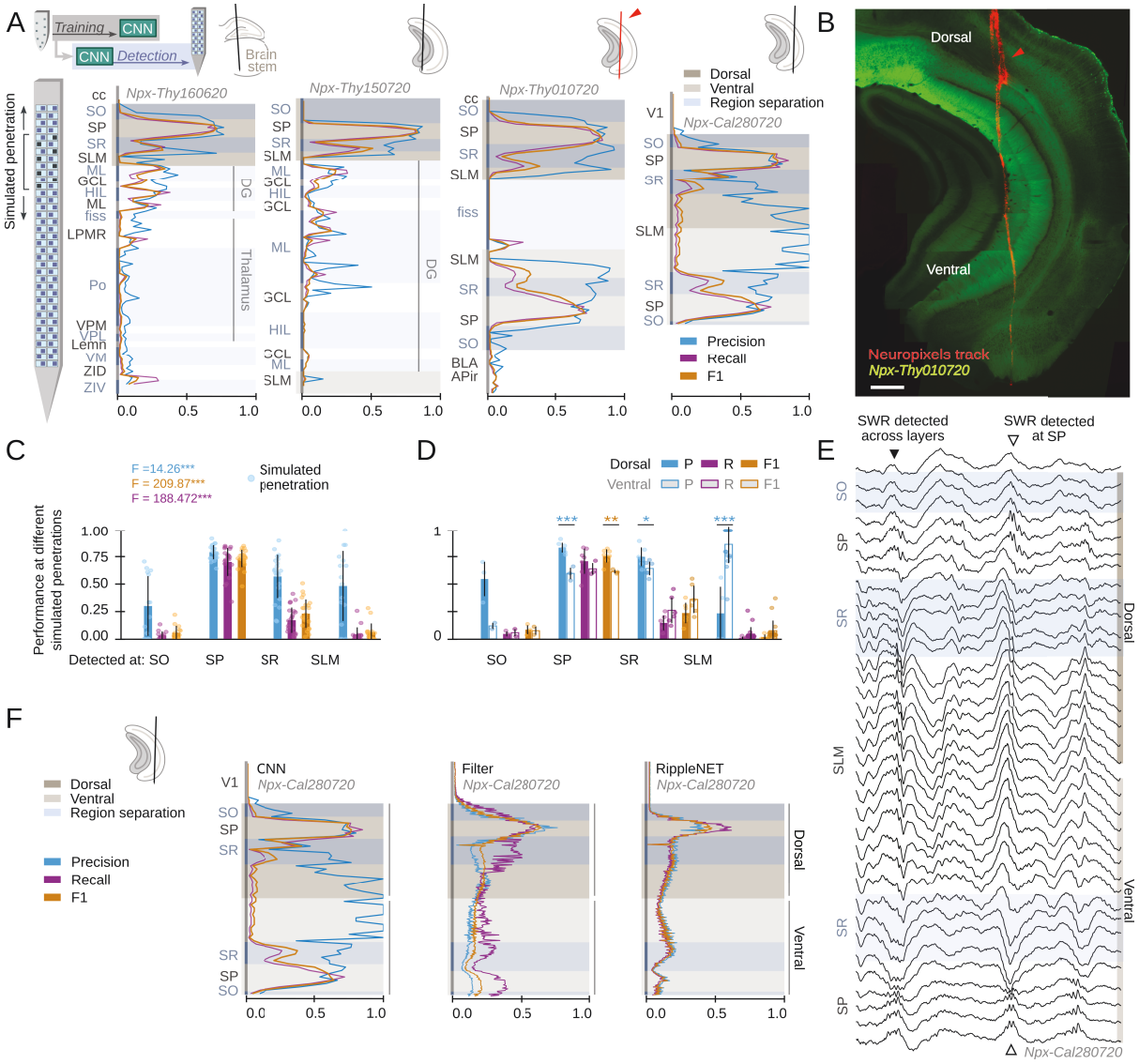

Figure 4.14: CNN-ripple performance with Neuropixels recordings. **A:** CNN32 offline performance over four sessions acquired with ultra-dense Neuropixels probes. Continuous simulated 8-channels penetrations were used as input for the neural network. Precision, recall and F1 were measured across several hippocampal layers and brain regions. **B:** histological validation of one of the mentioned experiments (indicated with the red arrow). **C:** mean CNN32 performance across simulated penetrations on each dorsal hippocampal layer (n=96 simulated penetrations, n=4 mice). Results of an independent one-way ANOVA for P, R and F1 are also shown (***, $p<0.001$). **D:** CNN32 performance pairwise comparison between dorsal and ventral simulated penetrations (n=55 dorsal and ventral simulated penetrations, n=4 mice; *, $p<0.05$; ** $p<0.01$; ***$p<0.001$). **E:** example of a SWR detected across several hippocampal layers and one just identified in the pyramidal layer. **F:** performance over continuous simulated penetrations on mouse Npx-Cal280720 (same as rightmost on A) for CNN32, for a Butterworth filter-based method and RippleNET. Figure adapted from [Navas-Olive et al., 2022].

sal. Furthermore, several events were detected in the surrounding layers, Radiatum (SR) and Lacunosum-Moleculare (SLM). In these layers the achieved recall score was low, meaning that the CNN found just a few SWRs, but the precision was high, so these few detections were mostly correct (Figure 4.14C). Moreover, this was not only observed in dorsal SR and SLM, but also in

their ventral counterparts (Figure 4.14D). In contrast, when a Butterworth filter-based method was applied over these sessions, its proper functioning was limited to the dorsal SP layer (Figure 4.14F).

These tests confirmed the CNN model generalization ability and its capacity to perform adequately over data recorded with probes distinct to the ones utilized to capture the training data, as long as they have at least 8 dense channels. This allows the standardization and dissemination of the detection method on different experimental setups, without regard of the kind of probes employed. Additionally, our neural network was also capable of identifying SWRs in hippocampal layers other than the ones used for training, and even also in the ventral area, thus proving its competence to work on signals with diverse properties. Moreover, Neuropixels is a very recent and ground breaking technology which is enabling researchers to conduct experiments and record data at a scale that was impossible until now. Having an automatic and online SWR detection method already compatible with this technology will be of great utility for the field.

4.3.5 Online performance

After validating their correct functioning in an offline scenario over recorded sessions, we went ahead to evaluate the CNN performance in an online and real-time experimental context. With this goal in mind, we conducted 6 experiments over 3 different mice (Table 4.1). During these sessions we compared the behaviour of CNN12 with the online version of the Butterworth filter-based spectral method. As a recording and processing tool for these experiments we used Open Ephys GUI 0.5.5.3, an open-source software designed to run on GPOS with soft real-time capabilities. This program can read from certain DAQ devices and save the registered data into files. Moreover, it allows to create a workflow of plugins which can be used to process the input data and visualize it in real-time in a flexible and easy way. We developed two custom plugins for this platform to be used during our experiments. The first plugin was designed to detect when a signal crossed a determined amplitude threshold, defined as the signal standard deviation multiplied by some number. It was used in combination with the Bandpass Filter plugin, which implemented a Butterworth filter, so the input for the crossing detector was a filtered signal. In order to avoid artifacts, we used a second input channel from a separate hippocampal region, with no presence of SWRs, selected by the experimenter. Events detected in both channels were discarded. The second plugin was developed to operate the CNN and it used Tensorflow 2.3.0 API for C[4]. Since the network was trained to work with data sampled at 1250 Hz, the plugin downsampled the input channels if necessary. It also separated data into windows of 12.8 ms and 8-channels to be fed into the CNN every 6.4 ms. Detection threshold was defined as a probability between 0 and 1, and it was manually adjusted by the experimenter. Both plugins normalized the input data using z-score normalization. They required a short calibration time (about one minute) to calculate the mean and standard deviation of the signals. The user could establish the detection threshold for both of them and when an event was found they would send a signal through a selected output channel. Figure 4.15 shows the configuration used for these experiments. Developed plugins can be downloaded from the GitHub repositories that can be found in Appendix A.

Equivalently to the offline scenario, a superior performance regarding the F1 was obtained with CNN12 (CNN12: F1=0.70 ± 0.07; Filter: F1=0.61 ± 0.09, $p<0.033$), as shown in Figure 4.16A left. If we take a look to the difference between the detection time and the SWR peak (see Section 4.2.4) for those events that were identified both by the CNN12 and the filter (n=704) we can observe that the network anticipates the filter (Figure 4.16A right; time-to-SWR-peak for CNN12: -4.82 ± 6.70 ms; Butterworth filter: -2.52 ± 3.90 ms; median ± median absolute

[4] https://www.tensorflow.org/install/lang_c

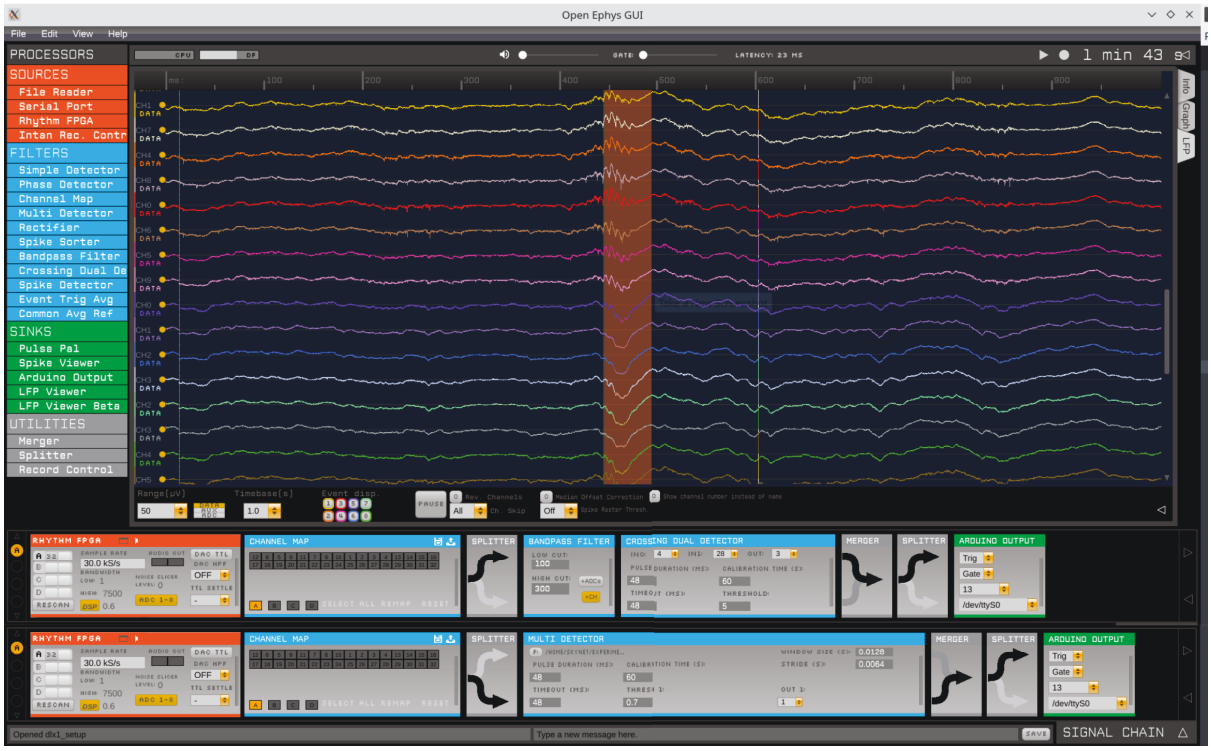

Figure 4.15: Example of an Open Ephys GUI workflow configuration for simultaneous detection using CNN12 and a spectral-based method. *Rhythm FPGA* read the data from the Intan board (it could be replaced for the corresponding Neuropixels input plugin to acquire data using these probes). *Channel Map* reordered the input channels to put the 8 ones corresponding to the selected shank first. *Splitter* duplicated the signals and sent them to each of the detection methods. The first one was formed by a *Bandpass Filter* module followed by the custom *Crossing Detector* plugin. The second one was our custom neural network plugin, that loaded the CNN12 model using Tensorflow API for C and returned a probability for each window. *Merger* collects the output from both plugins. When any of the methods identified a SWR, a TTL event was generated and displayed on screen over the LFP signals using the *LFP Viewer* module. *Arduino Output* was used to send these pulses to an exterior device. Both the electrophysiological data and the TTL pulses were saved during the experiments.

deviation). Lastly, we conducted another exploration of possible detection thresholds for both methods on the recordings of the six online experiments. The same pattern as with the previous sessions was produced by the F1 values for each possible threshold and each session, with the CNNs displaying a more robust behaviour and better performance (Figure 4.16B).

4.3.6 SWR disruption using closed-loop optogenetic stimulation

The overall goal of this project was not only to design a neural network able of detecting SWR in LFP signals, but also to provide an implementation of this method that was capable of performing this task in real-time during *in vivo* experiments. In such a way, we aimed to conduct online interventions in the subject's neural activity in order to modify or cancel upcoming or ongoing SWRs and study the consequences of such disturbances in the animal's behaviour and memory. The CNN12 model trained in previous sections is a computationally-light algorithm when it comes to evaluate a window of data, taking less than a millisecond to complete such an operation. Its only time-consuming aspect is that it requires a window of 12.8 ms to perform the

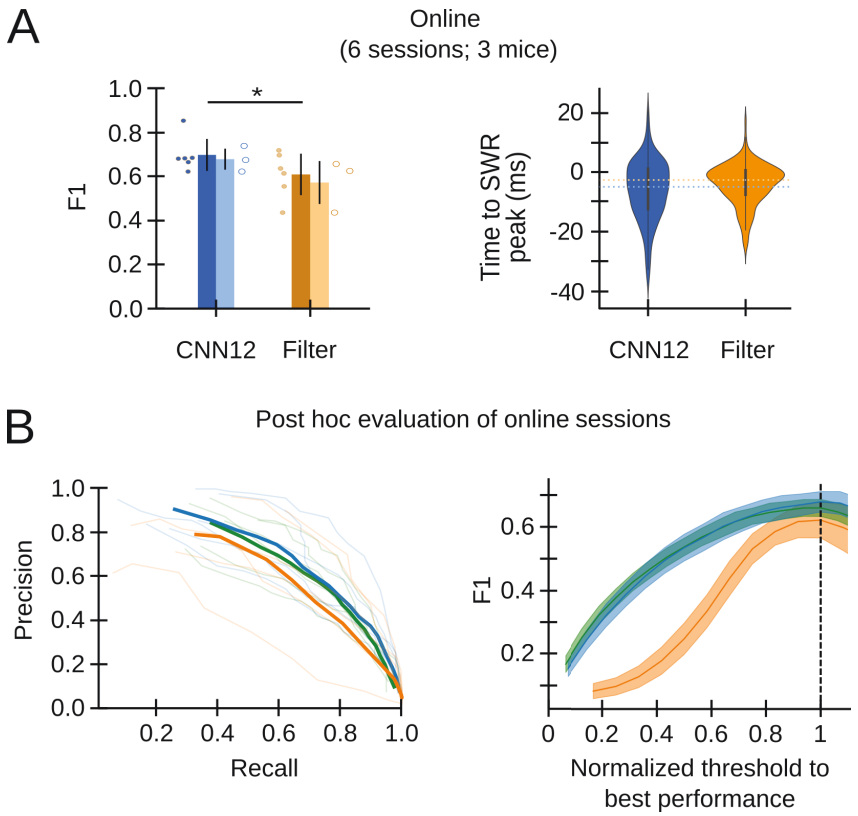

Figure 4.16: CNN-ripple online performance. **A:** Left: online detection performance (F1) for CNN12 and filter over test sessions (n=6 sessions, n=3 mice). Right: time-to-peak in milliseconds for methods. **B:** dataset is the same as in A but this analysis was conducted offline. Left: Precision-Recall curves for each session and method (light) and mean curve for each method (dark). Right: F1 score for each method when using a range of different detection thresholds on every session. Results represented as mean ±95% confidence interval. Figure adapted from [Navas-Olive et al., 2022].

detection, thus always existing a delay of at least that duration regarding the real event when a detection is made. Despite this, and as shown in Figure 4.16B, CNN12 time-to-SWR-peak is still inferior than the value obtained by the filter-based method. The reason for this is that spectral-methods also need to process some cycles of the oscillation in order to determine its frequency.

Once we had developed the deep learning-based method and validated its performance, it was necessary to implement it on a platform that allowed building real-time closed-loop protocols. Temporal constrains in this context were in the order of milliseconds and the system had some fault tolerance to missing events, hence a soft real-time implementation was deemed sufficient. Moreover, since we wanted to favour the dissemination of this technology to be used in other laboratories and experimental setups, all tools and platforms used for its development are user-friendly and easy to access by non-specialized users. For the first trials detailed in the following paragraphs we used mice injected with AAV-DIO-ChR2 to enable us to optogenetically alter SWRs (Table 4.1). Silicon probes employed in these experiments had micro-LEDs whose light modified neurons' behaviour. LED-controlling-device was connected to the output channel of the detection method (see Section 4.2.7).

In the first place, we decided to utilize the same software employed for online recording in the previous section, Open Ephys GUI 0.5.5.3 on Windows 11, in order to keep as much of the

existing experimental setup as unaltered as possible. The same plugins and workflow as in the previous case were used, consisting of an input block to acquire data from an Intan board, a channel mapping block to select the desired 8 channels and our own custom plugin to handle the neural network. One of the native output plugins was also added to send signals to external devices. In particular, Arduino Output plugin was utilized, which enabled the redirection of the CNN output signal to an Arduino's board digital output ports (Figure 4.17A). It was necessary to use an auxiliary Arduino as an output device instead of the Intan board employed for recording, which also has analog and digital output ports, due to technical issues in the programming API provided by the manufacturer that prevented simultaneous reading and writing with the device. This Arduino board sent a digital signal from one of its output ports, with value 1 when an event was detected or 0 otherwise. This signal handled the functioning of the device that controlled the LEDs. When the network detected a SWR, the LEDs lighted up causing an inhibitory effect in the target neurons, thus cancelling the ongoing event. Figure 4.17B shows this effect, but it can also be observed that even if the light's action on the neurons was immediate, there was some delay between the moment when the event was detected and when the LEDs were turned on. Measuring the latency between the expected intervention time and the time when the light was actually turned on in two sessions revealed an overall mean latency of 14.8 ± 3.6 ms.

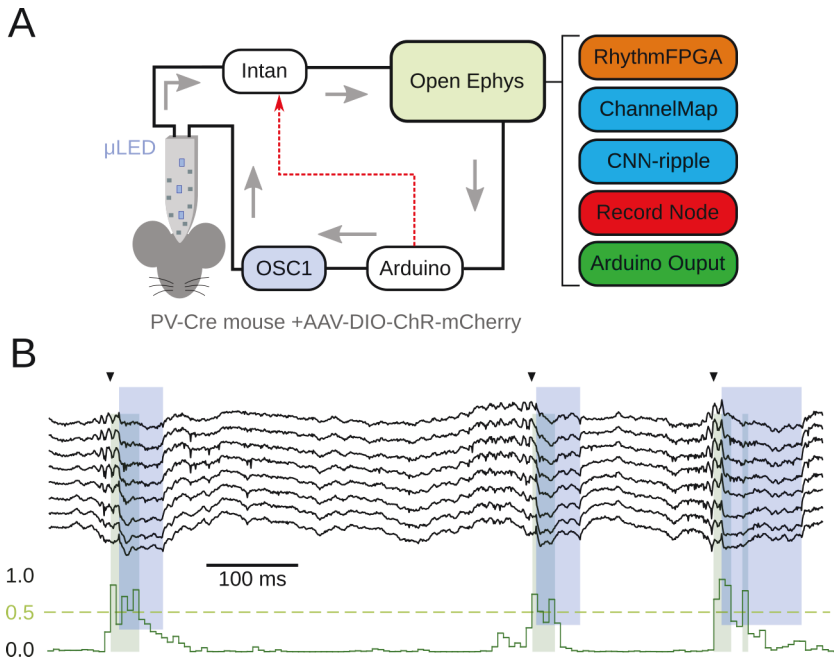

Figure 4.17: Closed-loop setup for SWR detection and intervention using Open Ephys. **A:** experimental setup for closed-loop interventions. An Intan board, connected to the computer via USB 2.0, was used as DAQ device and an OSC1 controlled the LEDs on the opto-electrode probe. A PV-cre mouse injected with AAV-DIO-ChR2 to optogenetically modulate SWR was used. CNN was utilized as detection method and handled through a custom Open Ephys plugin as described in Figure 4.15. Output pulses were sent via an Arduino board, also connected to the computer via USB 2.0. **B:** example of an online closed-loop intervention using the described setup. The green trace at the bottom represents the probability returned by CNN-ripple for each window, and a 0.7 threshold was established to detect SWRs. Note the latency between the detection (green shadow, when the pulse should be sent) and the actual time of the intervention (blue shadow), caused by Open Ephys hardware and software buffers. Figure adapted from [Navas-Olive et al., 2022].

One of the reasons of such a delay can be found in the USB 2.0 connections that both the

Intan and Arduino boards employ to link with the computer. An USB interface transmits the information in packages and there is an overhead cost when each one of them is created. For this reason, Open Ephys is configured to read 10 ms packages from the Intan board at 30kHz. Hence, the buffers in both hardware connections prevent an immediate acquisition or transmission of the data. Moreover, Open Ephys also utilizes buffers to handle the internal information traffic among its plugins and their size can be selected in the GUI. A smaller buffer size transfers the data faster, but also conveys a higher computational cost.

We conducted a set of latency tests in order to measure the amount of delay generated by Open Ephys while using different recording configurations and devices. Following the paradigm proposed by its developers[5] we registered the latency of the software when detecting the rising edge of a square-wave and sending a digital signal as response (see Section 4.2.8). These trials were performed using an Intan USB board, a National Instruments DAQ device and a Neuropixels device. Figure 4.18 summarizes the results of such trials. Despite the much greater number of recordings of the Neuropixels system, and therefore of required processing, the median latency in this case was 10.91 $\pm$ 3.16ms, probably due to the PCIe connection with the computer. On the other hand, using an USB-connected Intan board obtained a median latency of 15.42 $\pm$ 2.95ms. Surprisingly, the National Instruments board configuration got a median of 29.73 $\pm$ 6.49ms despite its PCIe connection due to the Open Ephys plugin that handled such devices.

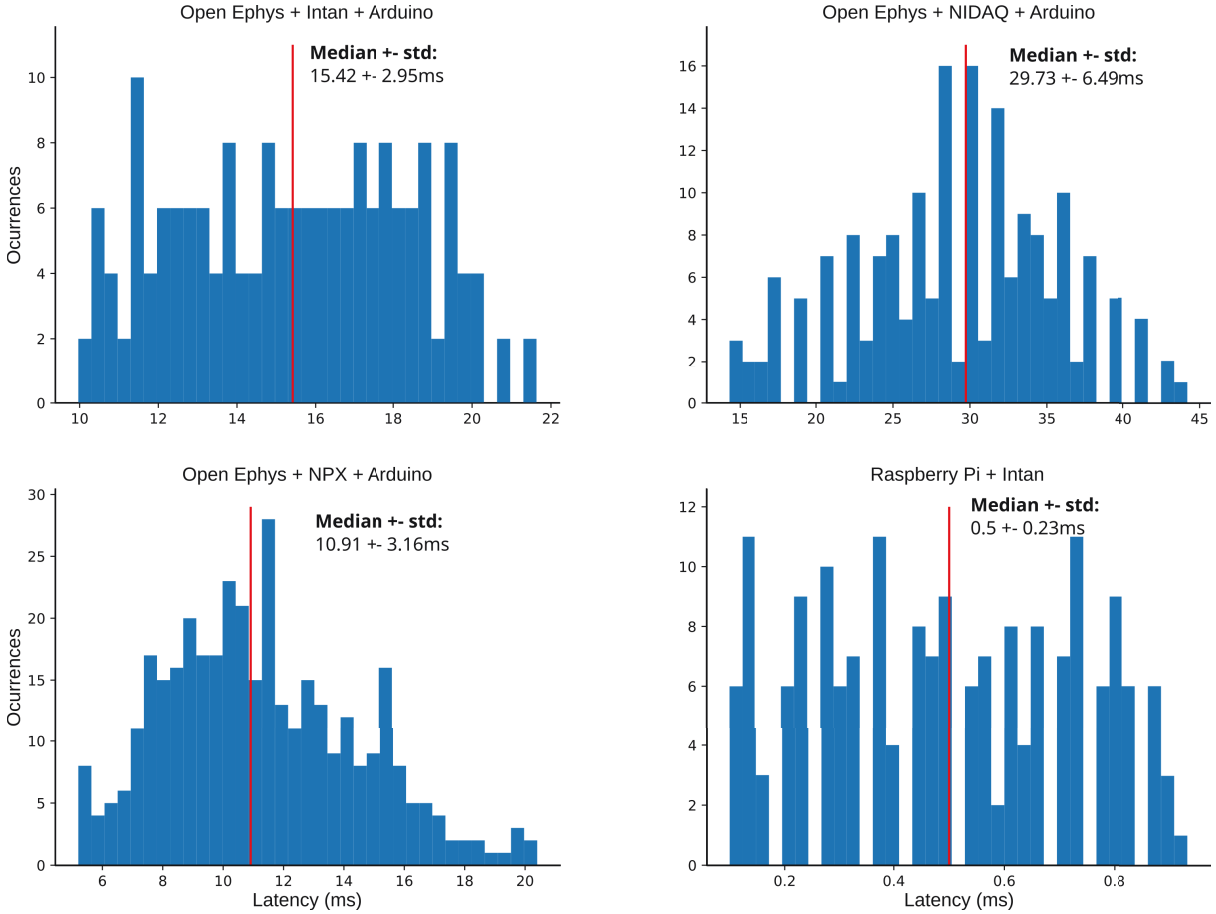

Figure 4.18: Results of the latency tests for each setup. Open Ephys software buffers, in addition to hardware connections, introduce several milliseconds latencies. These latencies are not present for the Raspberry Pi setup.

[5] https://open-ephys.github.io/gui-docs/Tutorials/Closed-Loop-Latency.html

In consideration of the bottleneck of this implementation being the buffers utilized for hardware-software communication and internal data flow between Open Ephys plugins, while CNN processing had an optimal performance, we searched for an alternative solution to avoid these problems. In order to build a system with minimum latencies when reading and writing from external devices we needed a more direct and faster way to interface with the hardware. At the same time, reducing the overhead during the processing steps would remove the worst part of the registered delays. For this reason, we decided to employ a single-board Raspberry Pi 4 computer to run the detection method, since it has enough computational power to execute our Tensorflow neural network and a flexible GPOS that made it easy to operate it. At the same time, this board comprised several input/output digital pins that could be used to communicate directly with other hardware devices (see Section 4.2.7). The system was built in a way that the software executed in the Raspberry Pi received the 8 channels that the CNN required to detect SWR, already sorted, via a SPI port. The returned values were outputted through a GPIO pin as a binary digital signal, indicating with a 1 or a 0 if there was a SWR in the window or not. Although for this project we used an Intan board as DAQ device, this solution was designed to work with any device that is able to provide the analog signals of the 8 LFP channels required. Repeating the same latency tests than before with this new implementation we obtained a median latency of 0.5 ± 0.23ms (Figure 4.18).

RHD2000 Intan USB Interface Board contains numerous output ports, both analog and digital. The manufacturer offers their own free acquisition software, called RHX Data Acquisition Software, and it is available for most GPOS. One of the functionalities provided by this program is the possibility of reprogramming the FPGA that serves as the board core. This way, eight of the LFP channels acquired through its SPI port, to which the silicon probes are connected, can be redirected on hardware level to the analog output ports. Internally, the board takes the signals of these 8 channels, which are already digitized at this point, and transforms them into analog signals, which are then put through a level shifter circuit to leave them in a -3.3V to 3.3V range[6]. Being all these operations carried out on hardware, the latencies are kept below 0.2ms[7]. Since a Raspberry Pi 4 board only has digital ports it was necessary to use a analog-to-digital converter (ADC) to transform the signals returned by the Intan board. In this case we employed a MCP3008 chip due to its capacity to handle eight channels simultaneously. However, this ADC only works with signals within a 0V to 3.3V range, which would discard all negative values of the input LFP. The Intan board manufacturer provides the schematics of the level shifter circuit required to re-scale signals in a -3.3V - 3.3V range, as the ones coming out the board's analog ports, to a 0V - 3.3V one, compatible with our ADC[8]. This circuit had to be implemented eight times, one per channel. For this reason, we decided to create a PCB board that grouped all eight level shifters as well as the ADC. The PCB was basically undoing the level shifting and digital-to-analog-conversion operations performed by the Intan board, but we still considered its use a better and more feasible solution than modifying the Intan board itself to bypass those actions. Again, since all this procedures were carried out on hardware level the latencies were minimal.

Regarding the software executed on the Raspberry Pi, it consisted in a modified version of the code used for the previous Open Ephys plugin, adapted to run as a stand-alone executable. Some parameters of the experiment, such as the detection threshold value or the duration of the pulses sent when an event was detected, could be entered by the user through command-line. This program did not save on file or prompted on screen any data at any point of the experiment to avoid unbounded latencies. Therefore, data recording had to be performed on a separated computer. The open-source pigpio library was employed to communicate with Raspberry Pi 4

[6] https://intantech.com/files/RHD2000_USB_interface_board_PCB.zip
[7] https://intantech.com/files/Intan_RHX_user_guide.pdf
[8] https://intantech.com/files/Intan_RHD2000_IO_level_shifters.pdf

ports, both SPI and GPIO. To optimize the neural network performance we used Tensorflow Lite, a lighter version of the library especially designed to run on smaller devices. Specifically, we utilized version 2.7.0 and its C language API, compiled for ARMv7 processors. CNN12 model had to be converted to Tensorflow Lite's format using the appropriate methods provided in the Python library.

Proper performance of this implementation was validated with three experiments similar to those described previously in Figure 4.17. In this case, the subjects were not sensitive to optogenetic stimulation and therefore SWRs were not interrupted. Nevertheless, the triggering signal produced by the Raspberry Pi device was recorded for further analysis (Figure 4.19A). CNN-ripple did not only run smoothly on a Raspberry Pi 4 computer with execution times under the millisecond order, but also the overall latency for all hardware operations remained below 1ms, consistent with the latency tests conducted earlier (Figure 4.19B).

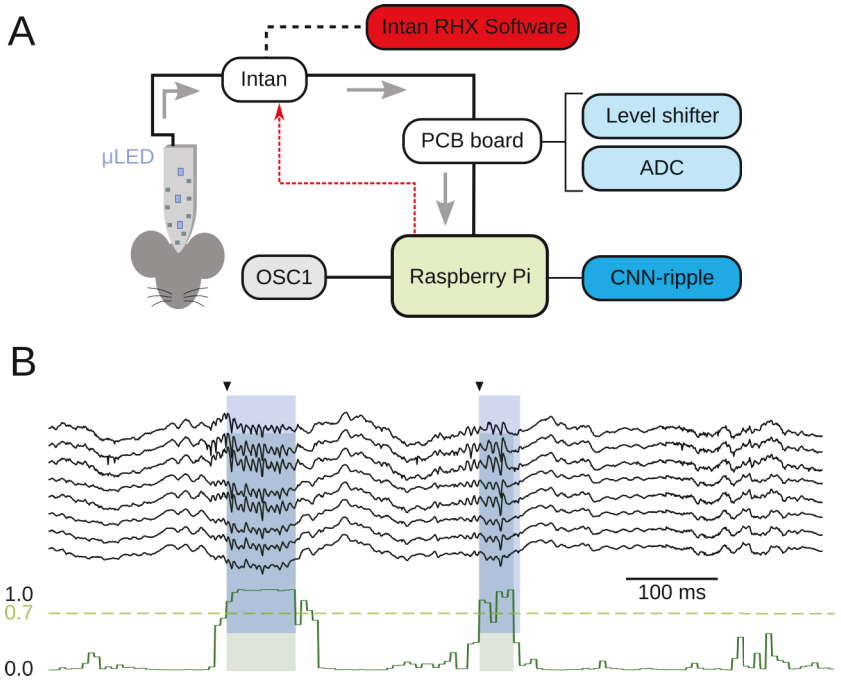

Figure 4.19: Closed-loop setup for SWR detection and intervention using a Raspberry Pi. **A:** experimental setup for closed-loop interventions. An Intan board was employed as acquisition device and Intan RHX Software was used to reprogram the FPGA to send 8 channels directly to the analog output ports. The outcoming signals were shifted and converted to digital values using a custom PCB and sent to a Raspberry Pi 4 device. This computer executed an instance of CNN-ripple specially adapted to smaller machines and produced a binary digital output that codified SWR detections. In this case the animal was not optogenetically stimulated but the triggering pulse was recorded for analysis purposes. **B:** example of an online closed-loop intervention using the described setup. The green trace at the bottom represents the probability returned by CNN-ripple for each window, and a 0.7 threshold was established to detect SWRs. Note that in this case the latency between the detection (green shadow, when the pulse should be sent) and the actual time of the intervention (blue shadow) is practically 0.

4.4 Discussion

In this project we designed and implemented a convolutional neural network model capable of real-time detection of sharp-wave ripples in LFP signals with a high degree of precision

and sensitivity. When compared to the spectral-based techniques traditionally employed to identify this kind of events, our neural network usually achieved a better perfomance while maintaining a more stable behaviour and less dependent on the detection threshold value. Even though the model was trained using data from two experimental sessions recorded from two mice, obtained with 8-channels high-density probes located in the CA1 hippocampal region, we validated its capacity to generalize and find SWR in totally independent data acquired in several experiments with other animals. Moreover, it also performs correctly on recordings from distinct animal species, hippocampal regions and layers and conducted with other types of dense probes, including Neuropixels. Furthermore, our CNN model has proved itself able of spotting events beyond those that constituted the training ground truth, sometimes distinguishing SWRs not labelled by the original expert but pointed by a second one. CNN-ripple is available freely as an interactive online notebook at https://github.com/PridaLab/cnn-ripple.

Capacity of our model to detect SWRs in real-time during closed-loop interventions has also been tested. In this context, we validated that this method was capable of identifying events over an online experiment with a smaller detection delay than filter-based methods. A custom plugin has been developed for Open Ephys in order to enable the construction of closed-loop systems using an user-friendly and widely spread platform. Additionally, the goal of reducing latencies caused by software and hardware connection buffers during data acquisition, processing and stimulation, we implemented a software and hardware hybrid real-time solution. It employed a Raspberry Pi 4 board that run an optimized instance of the detection algorithm. This computer received LFP signals after they have been scaled and digitized by a custom PCB designed for this purpose and sent an output digital signal to the stimulation device. Since all connections were carried out through low level management of the Raspberry Pi ports, the transfer latencies were minimal.

SWR detection based on spectral methods is limited to those with high power in their ripple-band oscillations, neglecting others with different and valuable characteristics not reflected in their frequency. The presented CNN model provides an alternative to these methods and the opportunity to generalize beyond frequency-related properties and identify events with a wider range of features. The use of machine learning techniques for the detection of SWRs opens a wide new range of possibilities in a field that was getting stuck in utilizing heuristics that did not permit to capture the whole complexity and richness of SWRs. The possibility of training these kind of algorithms with datasets labelled by various experts allows to overcome individual biases and move forward towards a broader and more precise definition of SWRs. These kind of solutions can be implemented in low cost platforms and using open-source tools, favouring an easier standardization and dissemination of this technology in other contexts and laboratories.

4.5 Future work

From a technical point of view, alternatives to the current implementation of the CNN could be explored in order to obtain a better offline and online performance, changing its architecture or even using different deep learning techniques and types of layers, such as RNNs or Transformers. Training the model with more data and tagged by multiple experts could also lead to a better performance. Tensorflow was used as machine learning framework due to the wide range of tools and options that it provides, as well as its usability and efficiency. However, a low level implementation of CNN-ripple could be developed, or even a full hardware implementation employing FPGAs, in order to reduce to the minimum the execution time.

CNN-ripple could be also trained to identify and classify SWRs with particular characteristics, as well as other electrophysiological events from different brain regions, including those that may precede SWR. Such a functionality, combined with the possibility of using Neuropixels

probes for online detection, would provide an early-intervention system for SWR. CNN-ripple performance on data recorded using different types of probes, such as tetrodes, can be explored. Additionally, identified functional events could be used to tag SWRs once the relationship between they common occurrence is established. All these possibilities enable the implementation of novel closed-loop protocols for further understanding the role that SWRs play in memory consolidation, spatial coding and other neural processes.

5

Conclusions and discussion

5.1 Conclusions

Three projects at different description levels were carried out throughout this thesis, aimed at the development of novel real-time tools that enable the implementation of closed-loop protocols to be utilized in experimental neuroscience. The experimental setups and research goals of all these works differed greatly from each other, thus employing distinct acquisition and stimulation techniques and requiring the use of different data processing approaches to identify diverse kinds of events. For this reason numerous technologies, both software and hardware, were used and developed in order to create effective real-time closed-loop interactions between biological and computational elements. However, despite this heterogeneity in goals and setups, all these protocols shared a common design pattern consisting in a goal-driven closed-loop interaction with the nervous system built using a periodic loop that iterated with a specified frequency. In each iteration of the loop the following key steps took place:

1. Neural data from the living neurons was acquired through a DAQ device.
2. Data was processed to detect the target events and extract their information.
3. Stimuli were generated using this information and sent back to the living cells in accordance to the specific goal given to the closed-loop.

Therefore, the same ideas, technologies and principles could be applied to all three projects, regardless of the use of different neural systems, experimental setups and temporal restrictions. The development and results of each one of them are summarized below:

- The first project framed in this thesis work consisted in the development of RTHybrid [Amaducci et al., 2019, Reyes-Sanchez et al., 2020], a software application designed to build hybrid circuits between biological neurons and neuron models in real-time in a simple and user-friendly way. On this account, it includes a neuron and synapse model library that can be easily expanded by the users, as well as an intuitive and handy user interface. Moreover, it also includes a set of calibration algorithms that automatically adjust all the required parameters for a proper interaction both in the amplitude and temporal

scales. Due to the necessity of temporal accuracy in the connection among biological and computational components, under the scale of milliseconds, the implemented software is compatible with hard real-time environments. We carried out an extensive study in order to determine which of the existing real-time operating systems were more suited to host our application, in terms of performance, usability and accessibility. This way we solved the technical complexities of building a hard real-time closed-loop experimental setup by providing and easy to install and use software solution. RTHybrid functioning was validated in various experiments with living neurons from the pyloric CPG of the crab *Carcinus maenas*. For this task, we employed the dynamic clamp technique to capture the neuron activity while we injected back a current with synaptic feedback generated by the computational models in real-time. RTHybrid is open-source and is available as a GitHub repository `https://github.com/GNB-UAM/RTHybrid`.

- For the second project we explored real-time interactions between neural circuits and robotic systems in order to create a so called FLC-Hybrot. Hybrots are a convenient tool to study the functional role of neural dynamics since they provide a physical "body" for experimental setups that would lack one otherwise, like the *in vitro* experiments with the aforementioned CPGs. With the goal of better understating the functional role of dynamical invariants present in these circuits in the form of robust cycle-by-cycle relationships, we used their activity to coordinate the locomotion of a robot. Specifically, we utilized time intervals that are known to maintain this kind of dynamical invariant relationship to online modulate the period and amplitude of the robot's leg oscillation. At the same time, the robot incorporated a photoresistor that detected the presence of lights and shadows over itself. When it was located under a shadow, the robot sent feedback to the living circuit in the form of electrical current that was injected into one of the CPG neurons. This closed-loop interaction was controlled in real-time by an intermediary computer that linked the robot and the CPG. We conducted a series of experiments to validate the coordinated locomotion of the hybrot and its adaptation to the living neurons behaviour, as well as the effective translation of the neural dynamical invariant to the movement of the legs. The code for this project can be found at `https://github.com/GNB-UAM/FLC-Hybrot`.

- Lastly, for the third project we created CNN-ripple [Navas-Olive et al., 2022], a sharp-wave ripple real-time detection system based on convolutional neural networks (available with an interactive notebook on GitHub `https://github.com/PridaLab/cnn-ripple`). SWR are brain oscillations related to memory consolidation and retrieval tasks, and therefore online interaction with them helps to understand their role in these processes. CNNs are a kind of machine learning algorithm that require very little pre-processing of the input data and are able of performing fast evaluations with reduced computational cost. Hence, they are ideal to be employed in a real-time context. Our CNN was trained using data recorded in the dorsal hippocampal region of two mice, but is capable of optimally generalize to sessions from distinct animals, species, hippocampal areas and recorded with different acquisition probes. Its performance is more robust and less detection-threshold-dependent than the one obtained by a spectral-based method, which are the current standard in the field. It also shows an improvement in terms of detection latency. When utilized on signals acquired with Neuropixels probes, we validated CNN-ripple proper functioning across multiple hippocampal layers, including those in the ventral region, as opposed to the filter method. CNN-ripple was also utilized to carry out online detection of SWR in closed-loop experiments where these events were disrupted by means of optogenetic intervention. In order to integrate the CNN trained model into a closed-loop workflow we implemented a custom plugin for Open Ephys, an open-source software platform widely spread in the experimental neuroscience community, which can be found at `https://github.com/PridaLab/CNNRippleDetectorOEPlugin`. Moreover, for the purpose of preventing latencies caused

by devices connections and Open Ephys internal handling of the data, our CNN-based detection system was implemented on a Raspberry Pi 4 board, reducing the delays caused by data processing and transfer to negligible values. This version of the code can be downloaded from https://github.com/PridaLab/cnn-ripple-raspberry-pi.

Since one of the fundamental goals of this work is to promote the standardization and dissemination of closed-loop protocols and real-time technologies in experimental neuroscience, all the developed software and tools are open-source and can be freely downloaded, modified and utilized. Appendix A contains links to every GitHub repository related to each of this thesis projects.

5.2 Discussion

Neural systems are typically studied through experiments that follow the stimulus-response paradigm. This approach is hindered when it comes to understand their complex non-linear dynamics, affected by numerous transient and interacting processes that take place in many different temporal and amplitude scales. Closed-loop techniques emerged as a solution to these issues, enabling the creation of stimuli based on the neural system own activity during the experiment. In order for this interaction to be established online and effectively, it is required to use real-time technologies (soft or hard) to ensure that data acquisition, information processing and the stimulation of biological components are all completed within the determined temporal margins. However, these tools are often complicated to install and utilize, therefore causing many researchers to overlook closed-loop implementations due to their complexity.

Closed-loop systems must adapt to the particular conditions of every experimental setup, which can vary greatly from one another in several aspects, as has been shown in the various projects carried out during this thesis. Data can be recorded by means of numerous techniques, with different degrees of spatial and temporal resolution, that range from intra or extracelullar electrophysiological recordings to imaging (voltage, calcium, fMRI, etc.), video tracking or robotic and neuroprostheses sensoring. Likewise, stimulating the biological elements can be done in many distinct ways: electrical current injection, chemical injection, light manipulation, mechanical manipulation, etc. All these methods can be applied at various precision levels, from individual neurons to whole systems. Regarding the processing of the acquired data, any procedure that generates stimuli based on the registered activity can be applied. Some examples of such operations are the calculation of the voltage that keeps the membrane potential of a neuron clamped at a fixed value, computing synapse and neuron models to build hybrid circuits or detecting specific neural events and generate pulses accordingly. Throughout this work we showed how diverse software and hardware technologies can be used and combined in order to implement real-time systems that allow to perform closed-loop experiments with different goals, tasks and components. Despite this heterogeneity in the type of the experiments and employed tools, there exist some pattern common to all closed-loop designs that can be standardized and reutilized in different contexts, as was done along the three projects of this thesis.

Real-time technology requirements when implementing closed-loop protocols are usually a handicap for many laboratories, since they demand highly specialized knowledge, skillsets and equipment. Fortunately, the advances in software and hardware achieved over the last decades provide a much more user-friendly and affordable framework to build closed-loop protocols. This is an outstanding opportunity to develop flexible tools that bring these paradigms and techniques closer to researchers and to facilitate their installation and implementation in different experimental environments, regardless of their acquisition, processing or stimulation setups. Similarly, they must be accessible and easy to use for any interested user, despite their technical expertise.

Moreover, employing open-source and low cost technologies promotes the dissemination and standardization of closed-loop techniques in the experimental neuroscience field, encouraging their use and making them available to every laboratory and researcher, independently of their budget. Characterization and control of neural systems dynamics, as well as experimental protocol automation, can largely benefit from the use of closed-loop techniques, favouring the design of novel experiments that allow the study of neural dynamics until now hardly understandable through the stimulus-response paradigm.

Designing closed-loop protocols is more complex and costly than stimulus-response ones since they require defining an experiment target, online event detection, validation metrics and a stimulation strategy, along with all the technical difficulties previously mentioned. Nevertheless, implementing this kind of paradigms is worth the effort since they provide new insights and data not obtainable by traditional approaches, as well as they enable the possibility of online control and interaction with neural systems, bringing experimental neuroscience research closer to biomedical applications.

5.3 Future work

Beyond the specific future work ideas from each chapter discussed in sections 2.5, 3.5 and 4.5, all the designed experiments and developed tools (for detection, analysis, stimulation, etc.) can be generalized and employed in other neuroscience experiments that use distinct recording and stimulation techniques (mechanical, optical, pharmacologial, etc.). Furthermore, these protocols can be expanded to integrate various levels of description, codification and knowledge.

Closed-loop protocols and real-time neurotechnologies contribute to identify, characterize and utilize neural dynamics' key elements with benefits not only in basic neural systems research, but also for their use in implementing brain-computer interfaces, rehabilitation paradigms, prosthetic devices, etc. Moreover, this approach provides novel opportunities for experimental designs in other biological research fields as well as the possibility to implement new frameworks for clinical applications.

Current technological trends are moving towards a deeper interaction between neural and artificial elements. There exist a lot of research aimed at making come true many ideas that sounded like science fiction not so long ago, such as neural implants that allow us to control computers with our minds, accessing the internet or watching TV in our heads, bionic prostheses or exoesqueletons that move with just a thought, or medical artefacts that enable monitoring and intervening patients remotely. All these efforts focused on the hybridation of neuroscience and engineered devices, mediated in many cases by artificial intelligence algorithms, rely in closed-loop protocols and require real-time technology. The development of this kind of techniques, as done throughout this thesis, is therefore crucial in order to reach these goals.

6

Conclusiones y discusión

6.1 Conclusiones

A lo largo de esta tesis se han llevado a cabo tres proyectos en diferentes niveles de descripción del sistema nervioso, todos ellos enfocados al desarrollo de herramientas de tiempo real que permitan la implementación de protocolos de ciclo cerrado para su uso en neurociencia experimental. Tanto los entornos experimentales como los objetivos de investigación en estos trabajos eran enormemente diferentes entre ellos, y por lo tanto emplearon distintas técnicas de adquisición y estimulación que requirieron del uso de aproximaciones específicas para la identificación de varios tipos de eventos. Por este motivo, se utilizaron y desarrollaron múltiples tecnologías, tanto software como hardware, con el objetivo de crear interacciones de ciclo cerrado en tiempo real entre elementos biológicos y computacionales. Sin embargo, y a pesar de la heterogeneidad tanto en configuraciones como en propósitos, todos estos protocolos compartieron un esquema de diseño común basado en una interacción de ciclo cerrado con el sistema nervioso enfocada a un objetivo y construida sobre un bucle periódico ejecutado con una frecuencia determinada. En cada iteración de este bucle se ejecutaron las siguientes tareas:

1. Adquisición de la actividad de las neuronas vivas a través de un dispositivo DAQ.
2. Procesamiento de los datos obtenidos para detectar los eventos de interés.
3. Generación y envío de vuelta a las neuronas vivas de estímulos generados a partir de la información adquirida y del objetivo concreto del ciclo cerrado.

Por lo tanto, se han podido aplicar las mismas ideas, tecnologías y principios a los tres proyectos, sin importar que cada uno utilizase diferentes sistemas neuronales y configuraciones experimentales o que tuviesen distintas restricciones temporales. El desarrollo y los resultados de cada uno de estos proyectos se resumen a continuación:

- El primer proyecto enmarcado dentro de esta tesis doctoral ha consistido en el desarrollo de RTHybrid [Amaducci et al., 2019, Reyes-Sanchez et al., 2020], una aplicación software diseñada para construir circuitos híbridos, formados por neuronas vivas y modelos de neuronas, en tiempo real y de una manera sencilla y accesible. Es por ello que incluye una

librería de modelos de neurona y sinapsis que puede ser ampliada fácilmente por los usuarios así como una interfaz de usuario intuitiva y práctica. Incluye además un conjunto de algoritmos de calibración que automáticamente ajustan todos los parámetros necesarios para una interacción correcta tanto en la escala temporal como de amplitud. Debido a la necesidad de precisión temporal en la conexión entre componentes biológicos y computacionales, a nivel de los milisegundos o inferior, el software implementado es compatible con entornos de tiempo real duro. Hemos realizado un estudio detallado para decidir cuáles de los sistemas operativos de tiempo real existentes eran, en términos de rendimiento, usabilidad y accesibilidad, los más adecuados para trabajar con una herramienta de las características que queríamos desarrollar. De esta manera, mediante la creación de una solución software fácil de instalar y usar, hemos hecho frente a las complicaciones técnicas que supone construir un sistema de ciclo cerrado con tiempo real duro. El funcionamiento de RTHybrid ha sido validado en varios experimentos con neuronas vivas extraídas del CPG pilórico de un cangrejo *Carcinus maenas*. Para estos experimentos hemos utilizado la técnica de *dynamic clamp* para registrar la actividad neuronal mientras que inyectábamos de vuelta en el circuito una corriente sináptica de retroalimentación generada por modelos computacionales en tiempo real. RTHybrid es una herramienta de código abierto y está disponible en un repositorio de GitHub `https://github.com/GNB-UAM/RTHybrid`.

- Durante el segundo proyecto hemos explorado las interacciones en tiempo real entre los circuitos neuronales y los sistemas robóticos con el propósito de crear lo que hemos bautizado como FLC-Hybrot. Los hybrots son una herramienta muy útil a la hora de estudiar el rol funcional que juegan las dinámicas neuronales, ya que proporcionan un "cuerpo" físico a preparaciones experimentales que carecen de él, como los experimentos *in vitro* con CPGs mencionados anteriormente. Con el objetivo de comprender mejor el papel que los invariantes dinámicos juegan en el funcionamiento de estos circuitos, los cuales aparecen en forma de relaciones ciclo-a-ciclo robustas entre intervalos específicos que construyen la secuencia motora, hemos utilizado la actividad del CPG para controlar el movimiento de un robot. En concreto, hemos usado aquellos intervalos temporales que sabemos que mantienen este tipo de invariante dinámico para modular en tiempo real el periodo y la amplitud de la oscilación de las patas robóticas. El robot a su vez disponía de una fotoresistencia que detectaba la presencia de luces o sombras sobre el mismo. En el caso de encontrarse situado bajo una sombra, el robot enviaba retroalimentación al circuito vivo en la forma de corriente eléctrica que se inyectaba en una de las neuronas. Esta interacción en ciclo cerrado estaba controlada por un ordenador que actuaba como intermediario entre el robot y el CPG. Hemos llevado a cabo una serie de experimentos para validar el movimiento coordinado del hybrot y su adaptación al comportamiento de las neuronas vivas, así como la eficacia al transferir el invariante dinámico de la actividad neuronal a la oscilación de las patas. El código utilizado en este proyecto se puede encontrar en `https://github.com/GNB-UAM/FLC-Hybrot`.

- Por último, para el tercer proyecto hemos creado CNN-ripple [Navas-Olive et al., 2022], un sistema de detección de *sharp-wave ripples* basado en redes neuronas convolucionales (disponible como un notebook interactivo en GitHub `https://github.com/PridaLab/cnn-ripple`). Los *sharp-wave ripples* son oscilaciones neuronales relacionadas con tareas de consolidación y recuperación de memoria. La interacción en vivo con dichos eventos ayuda a comprender su función dentro de estos procesos. Las CNNs son un tipo de algoritmo de aprendizaje automático que requiere de muy poco pre-procesado de los datos para trabajar y puede realizar evaluaciones rápidamente y con poco coste computacional. Es por ello que son ideales para ser utilizadas en contextos de tiempo real. Nuestra CNN ha sido entrenada con datos grabados en la región dorsal del hipocampo en dos ratones, y sin embargo hemos comprobado que es capaz de generalizar de manera adecuada en sesiones de

otros animales, especies, zonas del hipocampo u obtenidas con sondas de adquisición diferentes. Su rendimiento es más robusto y menos dependiente del umbral de detección que el de los métodos basados en frecuencia que habitualmente se utilizan para estas tareas. También proporcionan una mejora en cuanto a las latencias en la detección. CNN-ripple se ha utilizado también para realizar detecciones de SWR en tiempo real en experimentos de ciclo cerrado en los que estos eventos eran interrumpidos mediante intervención optogenética. Para poder integrar nuestro modelo de CNN en dicho sistema de ciclo cerrado hemos implementado un complemento para Open Ephys, una plataforma software de código abierto cuyo uso esta muy extendido entre la comunidad experimental en neurociencia, y se puede descargar de `https://github.com/PridaLab/CNNRippleDetectorOEPlugin`. Además, y con el objetivo de evitar las latencias causadas por las conexiones entre dispositivos y el manejo interno de los datos que hace Open Ephys, nuestro sistema de detección basado en CNNs ha sido implementado en una Raspberry Pi 4, reduciendo los retrasos causados por el procesamiento y envío de datos a valores insignificantes. Esta versión del código se puede encontrar en `https://github.com/PridaLab/cnn-ripple-raspberry-pi`.

Puesto que uno de los objetivos fundamentales de este trabajo es el promocionar la estandarización y diseminación de los protocolos de ciclo cerrado y las tecnologías de tiempo real en el campo de la neurociencia experimental, todo el software y las herramientas desarrolladas son de código abierto y pueden descargarse, modificarse y utilizarse gratuitamente. El Apéndice A contiene los enlaces a todos los repositorios de GitHub relacionados con los proyectos de esta tesis doctoral.

6.2 Discusión

Los sistemas neuronales normalmente son estudiados a través de experimentos que siguen el paradigma de estímulo-respuesta. Este enfoque se encuentra limitado cuando se trata de entender las dinámicas neuronales, complejas y altamente no lineales, moduladas además por la interacción con numerosos procesos transitorios que tienen lugar en diversas escalas de tiempo y amplitud. Las técnicas de ciclo cerrado surgen como una solución a estos problemas, permitiendo la creación de estímulos basados en la propia actividad del sistema que se estudia durante el experimento. Para que está interacción pueda establecerse en vivo y de manera efectiva, se requiere del uso de tecnologías de tiempo real (suave o duro) que aseguren que la adquisición de datos, el procesado de la información y la estimulación de los componentes biológicos sean llevados a cabo dentro de unos margenes temporales determinados. Sin embargo, estas herramientas son a menudo difíciles de instalar y utilizar, provocando que muchos investigadores desestimen el uso de protocolos de ciclo cerrado debido a su complejidad.

Los sistemas de ciclo cerrado deben adaptarse a las condiciones particulares de cada configuración experimental que, como se ha visto en los múltiples proyectos realizados a lo largo de esta tesis, pueden variar de un caso a otro en muchos aspectos. Los datos se pueden adquirir con diversas técnicas, que a su vez pueden tener diferentes grados de resolución espacial y temporal, que abarcan desde las grabaciones electrofisiológicas intra o extracelulares hasta las basadas en imagen (voltaje, calcio, fMRI, etc.), el seguimiento en video o los sensores incorporados en robots y neuroprótesis. De la misma manera, la estimulación de los elementos biológicos se puede realizar de muchas maneras diferentes: inyección de corriente eléctrica, inyección de sustancias químicas, manipulación mediante luz, manipulación mecánica, etc. Todos estos métodos pueden aplicarse en distintos niveles de precisión, desde neuronas individuales a sistemas completos. En cuanto al procesado de los datos registrados, cualquier procedimiento capaz de generar estímulos basados en la actividad neuronal recién grabada puede ser utilizado. Algunos ejemplos de este tipo operaciones son el cálculo de un voltaje que mantenga el potencial de membrana de una

neurona fijo en un valor determinado, computar modelos de sinapsis y neuronas para construir circuitos híbridos o detectar eventos neuronales concretos y generar pulsos cuando aparezcan. A lo largo de este trabajo hemos mostrado cómo distintas tecnologías software y hardware pueden combinarse para implementar sistemas de tiempo real que permitan realizar experimentos de ciclo cerrado con distintas metas, tareas y componentes. A pesar de esta heterogeneidad en los tipos de experimentos y las herramientas utilizadas, existe un patrón común en el diseño de todos los protocolos de ciclo cerrado que puede ser estandarizado y reutilizado en diferentes contextos, tal y como se ha hecho a lo largo los tres proyectos de esta tesis.

El hecho de necesitar tecnologías de tiempo real al implementar protocolos de ciclo cerrado es habitualmente un inconveniente para muchos laboratorios, puesto que requieren de conocimientos, habilidades y equipamientos muy especializados. Afortunadamente, los avances en software y hardware en las últimas décadas proporcionan posibilidades mucho más accesibles y asequibles a la hora de construir protocolos de ciclo cerrado. Esto proporciona una maravillosa oportunidad para desarrollar herramientas flexibles que acerquen estos paradigmas y técnicas a los investigadores y faciliten su instalación e incorporación en distintos entornos experimentales, indistintamente de sus métodos de adquisición, procesado de datos o estimulación. De igual manera, estas deben ser fáciles de usar para cualquier usuario interesado, sin importar su experiencia técnica. Además, el uso de tecnologías abiertas y de bajo coste los pone a disposición de cualquier laboratorio o investigador, independientemente de su presupuesto. La caracterización y el control de los sistemas neuronales, así como la automatización de protocolos experimentales, se pueden beneficiar enormemente del uso de técnicas de ciclo cerrado, favoreciendo el diseño de nuevos experimentos que permitan el estudio de dinámicas neuronales hasta ahora difícilmente interpretables a través del paradigma de estímulo-respuesta.

El diseño de protocolos de ciclo cerrado es más complejo y costoso que el de los protocolos de estímulo-respuesta, puesto que requieren definir un objetivo experimental, detectar eventos en tiempo real, y establecer métricas de validación y una estrategia de estimulación, además de lidiar con todas las dificultades técnicas mencionadas anteriormente. A pesar de esto, la implementación de esta clase de paradigmas merece la pena ya que proporcionan nuevas perspectivas y datos difícilmente obtenibles mediante los enfoques tradicionales, a la vez que permiten la interacción y el control de los sistemas neuronales en vivo, acercando la neurociencia experimental a las aplicaciones biomédicas.

6.3 Trabajo futuro

Más allá de las ideas para el trabajo futuro discutidas en las secciones 2.5, 3.5 y 4.5 específicas para cada proyecto, todos los experimentos diseñados y las herramientas desarrolladas (para detección, análisis, estimulación, etc.) pueden ser generalizadas y utilizadas en otros experimentos de neurociencia que usen técnicas de adquisición y estimulación diferentes (mecánica, óptica, farmacológica, etc.). Asimismo, estos protocolos puede ampliarse para integrar varios niveles de descripción, codificación y seguimiento.

Los protocolos de ciclo cerrado y las neurotecnologías de tiempo real contribuyen a la identificación, caracterización y utilización de elementos claves presentes en las dinámicas neuronales, con beneficios no solo para la investigación básica en sistemas neuronales sino también a la hora de implementar interfaces cerebro-máquina, paradigmas de rehabilitación, dispositivos prostéticos, etc. De la misma manera, este acercamiento proporciona nuevas oportunidades para el diseño experimental en otros campos de la investigación biológica, así como la posibilidad de crear nuevos marcos para aplicaciones clínicas.

Las tendencias tecnológicas actuales avanzan hacia una mayor interacción entre elementos neuronales y artificiales. Existe mucha investigación dedicada a hacer realidad ideas que hace

no mucho tiempo nos sonaban a ciencia ficción, tales como los implantes cerebrales que nos permitan controlar ordenadores con la mente, el acceso a internet o ver la televisión directamente en nuestras cabezas, las prótesis biónicas y los exoesqueletos que se pueden manejar solo con el pensamiento, o los dispositivos médicos que permiten monitorizar e intervenir pacientes a distancia. Todos estos esfuerzos enfocados en la hibridación de la neurociencia y la ingeniería, mediada en muchos casos por algoritmos de inteligencia artificial, se basan en protocolos de ciclo cerrado y requieren de herramientas de tiempo real. El desarrollo de este tipo de técnicas, tal y como se ha hecho a lo largo de esta tesis, es por lo tanto crucial para alcanzar estas metas.

Bibliography

L. F. Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50:303–304, 11 1999. ISSN 0361-9230. doi: 10.1016/S0361-9230(99)00161-6.

Adrian Aleman-Zapata, Jacqueline van der Meij, and Lisa Genzel. Disrupting ripples: Methods, results, and caveats in closed-loop approaches in rodents. *Journal of Sleep Research*, 12 2021. doi: 10.1111/JSR.13532. URL https://onlinelibrary.wiley.com/doi/full/10.1111/jsr.13532.

Rodrigo Amaducci, Manuel Reyes-Sanchez, Irene Elices, Francisco B. Rodriguez, and Pablo Varona. RTHybrid: A standardized and open-source real-time software model library for experimental neuroscience. *Frontiers in Neuroinformatics*, 13:1–14, 2 2019. ISSN 16625196. doi: 10.3389/FNINF.2019.00011. URL https://doi.org/10.3389/fninf.2019.00011.

David Amaral and Pierre Lavenex. Hippocampal neuroanatomy. *The Hippocampus Book*, 5 2006. doi: 10.1093/ACPROF:OSO/9780195100273.003.0003.

Per Andersen, Richard Morris, David Amaral, Tim Bliss, and John O'Keefe. *Historical Perspective: Proposed Functions, Biological Characteristics, and Neurobiological Models of the Hippocampus*. Oxford University Press, 5 2006. ISBN 9780199864133. doi: 10.1093/ACPROF:OSO/9780195100273.003.0002.

Noriyasu Ando and Ryohei Kanzaki. Insect-machine hybrid robot. *Current Opinion in Insect Science*, 42:61–69, 12 2020. ISSN 2214-5745. doi: 10.1016/J.COIS.2020.09.006.

Rafael V Aroca and Glauco Caurin. A real time operating systems (RTOS) comparison. page 12, 2009. URL http://csbc2009.inf.ufrgs.br/anais/pdf/wso/st04_03.pdf.

Cédric Arrouët, Marco Congedo, Jean-Eudes Marvie, Fabrice Lamarche, Anatole Lécuyer, and Bruno Arnaldi. Open-ViBE: A three dimensional platform for real-time neuroscience. *Journal of Neurotherapy*, 9:3–25, 7 2005. doi: 10.1300/J184v09n01_02. URL http://www.isnr-jnt.org/article/view/16837.

Maura Arsiero, H R Lüscher, and M Giugliano. Real-time closed-loop electrophysiology: towards new frontiers in in vitro investigations in the neurosciences. *Archives italiennes de biologie*, 145:193–209, 2007.

Joseph Ayers and Jan Witting. Biomimetic approaches to the control of underwater walking machines. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365:273–295, 11 2006. ISSN 1364503X. doi: 10.1098/RSTA.2006.1910. URL https://royalsocietypublishing.org/doi/full/10.1098/rsta.2006.1910.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, 3 2018. doi: 10.48550/arxiv.1803.01271. URL https://arxiv.org/abs/1803.01271v2.

A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio. Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application. *IEEE Transactions on Nuclear Science*, 55:435–439, 2008. ISSN 0018-9499. doi: 10.1109/TNS.2007.905231. URL http://ieeexplore.ieee.org/document/4448543/.

Alim Louis Benabid, Stephan Chabardes, John Mitrofanis, and Pierre Pollak. Deep brain stimulation of the subthalamic nucleus for the treatment of parkinson's disease. *The Lancet Neurology*, 8:67–81, 1 2009. ISSN 1474-4422. doi: 10.1016/S1474-4422(08)70291-6.

István Biró and Michele Giugliano. A reconfigurable visual-programming library for real-time closed-loop cellular electrophysiology. *Frontiers in Neuroinformatics*, 9:17, 2015. doi: 10.3389/fninf.2015.00017. URL http://www.ncbi.nlm.nih.gov/pubmed/26157385.

Marta Bisio, Alexey Pimashkin, Stefano Buccelli, Jacopo Tessadori, Marianna Semprini, Timothee Levi, Ilaria Colombi, Arseniy Gladkov, Irina Mukhina, Alberto Averna, Victor Kazantsev, Valentina Pasquale, and Michela Chiappalone. Closed-loop systems and in vitro neuronal cultures: Overview and applications. *Advances in Neurobiology*, 22:351–387, 2019. ISSN 21905223. doi: 10.1007/978-3-030-11135-9_15. URL https://link.springer.com/chapter/10.1007/978-3-030-11135-9_15.

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *ArXiv*, 4 2020. doi: 10.48550/arxiv.2004.10934. URL https://arxiv.org/abs/2004.10934v1.

Frédéric D Broccard, Siddharth Joshi, Jun Wang, and Gert Cauwenberghs. Neuromorphic neural interfaces: From neurophysiological inspiration to biohybrid coupling with nervous systems. *Journal of Neural Engineering*, 14, 2017. ISSN 17412552. doi: 10.1088/1741-2552/aa67a9. URL http://iopscience.iop.org/1741-2552/14/4/041002.

Ludmila Brochini, Pedro V Carelli, and Reynaldo D Pinto. Single synapse information coding in intraburst spike patterns of central pattern generator motor neurons. *Journal of Neuroscience*, 31:12297–12306, 2011.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020-December, 5 2020. ISSN 10495258. doi: 10.48550/arxiv.2005.14165. URL https://arxiv.org/abs/2005.14165v4.

N. Buduma and N. Locascio. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. O'Reilly Media, 2017. ISBN 9781491925584. URL https://books.google.es/books?id=80glDwAAQBAJ.

György Buzsáki. Hippocampal sharp wave-ripple: A cognitive biomarker for episodic memory and planning. *Hippocampus*, 25:1073–1188, 10 2015. ISSN 1098-1063. doi: 10.1002/HIPO.22488. URL https://onlinelibrary.wiley.com/doi/full/10.1002/hipo.22488.

György Buzsáki, Costas A. Anastassiou, and Christof Koch. The origin of extracellular fields and currents — EEG, ECoG, LFP and spikes. *Nature Reviews Neuroscience*, 13:407–420, 5 2012. ISSN 1471-0048. doi: 10.1038/nrn3241. URL https://www.nature.com/articles/nrn3241.

György Buzsáki, Eran Stark, Antal Berényi, Dion Khodagholy, Daryl R. Kipke, Euisik Yoon, and Kensall D. Wise. Tools for probing local circuits: High-density silicon probes combined with optogenetics. *Neuron*, 86:92–105, 4 2015. ISSN 0896-6273. doi: 10.1016/J.NEURON.2015.01.028.

Santiago R Cajal. Histologie du système nerveux de l'homme et des vertébrés. grand sympathique. *Paris Maloine*, 2:891–942, 1911.

James Cavanaugh, Ilya E. Monosov, Kerry McAlonan, Rebecca Berman, Mitchell K. Smith, Vania Cao, Kuan H. Wang, Edward S. Boyden, and Robert H. Wurtz. Optogenetic inactivation modifies monkey visuomotor behavior. *Neuron*, 76:901–907, 12 2012. ISSN 0896-6273. doi: 10.1016/J.NEURON.2012.10.016.

Numan Celik, Fiona O'Brien, Sean Brennan, Richard D. Rainbow, Caroline Dart, Yalin Zheng, Frans Coenen, and Richard Barrett-Jolley. Deep-channel uses deep neural networks to detect single-molecule events from patch-clamp data. *Communications Biology*, 3:1–10, 1 2020. ISSN 2399-3642. doi: 10.1038/s42003-019-0729-3. URL https://www.nature.com/articles/s42003-019-0729-3.

Pablo Chamorro, Rafael Levi, Francisco B Rodríguez, Reynaldo D Pinto, and Pablo Varona. Real-time activity-dependent drug microinjection. *BMC Neuroscience*, 10:P296, 2009. ISSN 1471-2202. doi: 10.1186/1471-2202-10-S1-P296. URL http://bmcneurosci.biomedcentral.com/articles/10.1186/1471-2202-10-S1-P296.

Pablo Chamorro, Carlos Muñiz, Rafael Levi, David Arroyo, Francisco B. Rodriguez, and Pablo Varona. Generalization of the dynamic clamp concept in neurophysiology and behavior. *PLoS ONE*, 7:e40887, 7 2012. doi: 10.1371/journal.pone.0040887. URL http://dx.plos.org/10.1371/journal.pone.0040887.

Ritchie Chen, Andres Canales, and Polina Anikeeva. Neural recording and modulation technologies. *Nature Reviews Materials*, 2:1–16, 1 2017. ISSN 2058-8437. doi: 10.1038/natrevmats.2016.93. URL https://www.nature.com/articles/natrevmats201693.

Zheng Chen, Tae I. Um, and Hilary Bart-Smith. Bio-inspired robotic manta ray powered by ionic polymer-metal composite artificial muscles. *International Journal of Smart and Nano Materials*, 3:296–308, 2012. ISSN 1947542X. doi: 10.1080/19475411.2012.686458. URL https://www.tandfonline.com/doi/abs/10.1080/19475411.2012.686458.

Karen C. Cheung. Implantable microscale neural interfaces. *Biomedical Microdevices*, 9:923–938, 12 2007. ISSN 13872176. doi: 10.1007/S10544-006-9045-Z. URL https://link.springer.com/article/10.1007/s10544-006-9045-z.

David J Christini, Kenneth M Stein, Steven M Markowitz, and Bruce B Lerman. Practical real-time computing system for biomedical experiment interface. *Annals of Biomedical Engineering*, 27:180–186, 1999. ISSN 00906964. doi: 10.1114/1.185.

Davide Ciliberti and Fabian Kloosterman. Falcon: a highly flexible open-source software for closed-loop neuroscience. *Journal of Neural Engineering*, 14, 2017. doi: 10.1088/1741-2552/aa7526. URL http://stacks.iop.org/1741-2552/14/i=4/a=045004.

Dan Ciregan, Ueli Meier, and Jurgen Schmidhuber. Multi-column deep neural networks for image classification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2012. ISSN 10636919. doi: 10.1109/CVPR.2012.6248110.

Stuart F. Cogan. Neural stimulation and recording electrodes. *Annual Review of Biomedical Engineering*, 10:275–309, 7 2008. ISSN 15239829. doi: 10.1146/ANNUREV.BIOENG.10.061807.160518. URL https://www.annualreviews.org/doi/abs/10.1146/annurev.bioeng.10.061807.160518.

Michael X. Cohen. Where does EEG come from and what does it mean? *Trends in Neurosciences*, 40:208–218, 4 2017. ISSN 0166-2236. doi: 10.1016/J.TINS.2017.02.004.

Kenneth S. Cole and Howard J. Curtis. Electric impedance of the squid giant axon during activity. *The Journal of General Physiology*, 22, 1939. URL http://jgp.rupress.org/content/22/5/649.

Barry W. Connors. Synchrony and so much more: Diverse roles for electrical synapses in neural circuits. *Developmental Neurobiology*, 77:610–624, 5 2017. ISSN 1932-846X. doi: 10.1002/DNEU.22493. URL https://onlinelibrary.wiley.com/doi/full/10.1002/dneu.22493.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels. pages 109–116. AUAI Press, 5 2012. doi: 10.48550/arxiv.1205.2653. URL https://arxiv.org/abs/1205.2653v1.

Ellen Covey and Matt Carter. *Basic electrophysiological methods*. Oxford University Press, 2015. ISBN 9780199939862. URL http://www.worldcat.org/title/basic-electrophysiological-methods/oclc/900633113.

Alessandro Crespi, André Badertscher, André Guignard, and Auke Jan Ijspeert. AmphiBot I: an amphibious snake-like robot. *Robotics and Autonomous Systems*, 50:163–175, 3 2005. ISSN 0921-8890. doi: 10.1016/J.ROBOT.2004.09.015.

Alessandro Crespi, Konstantinos Karakasiliotis, Andre Guignard, and Auke Jan Ijspeert. Salamandra Robotica II: An amphibious robot to study salamander-like swimming and walking gaits. *IEEE Transactions on Robotics*, 29:308–320, 2013. ISSN 15523098. doi: 10.1109/TRO.2012.2234311.

Yann Le Cun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. pages 396–404. Morgan Kaufmann, 1990.

Yann Le Cun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2323, 1998. ISSN 00189219. doi: 10.1109/5.726791.

Yann Le Cun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 5 2015. ISSN 1476-4687. doi: 10.1038/nature14539. URL https://www.nature.com/articles/nature14539.

Fernando Lopes da Silva. EEG and MEG: Relevance to neuroscience. *Neuron*, 80:1112–1128, 12 2013. ISSN 0896-6273. doi: 10.1016/J.NEURON.2013.10.017.

Thomas B. DeMarse, Daniel A. Wagenaar, Axel W. Blau, and Steve M. Potter. The neurally controlled animat: Biological brains acting with simulated bodies. *Autonomous Robots*, 11:305–310, 11 2001. ISSN 1573-7527. doi: 10.1023/A:1012407611130. URL https://link.springer.com/article/10.1023/A:1012407611130.

Li Deng and Dong Yu. *Deep Learning: Methods and Applications*, volume 7, pages 197–387. Now Publishers, Inc., 2013. ISBN 9781601988157. doi: 10.1561/2000000039. URL http://dx.doi.org/10.1561/2000000039.

Niraj S Desai, Richard Gray, and Daniel Johnston. A dynamic clamp on every rig. *eneuro*, 10, 2017. ISSN 2373-2822. doi: 10.1523/ENEURO.0250-17.2017. URL http://eneuro.sfn.org/lookup/doi/10.1523/ENEURO.0250-17.2017.

A. Destexhe, Z. F. Mainen, and T. J. Sejnowski. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation*, 6:14–18, 1 1994. doi: 10.1162/neco.1994.6.1.14. URL http://www.mitpressjournals.org/doi/10.1162/neco.1994.6.1.14.

Alain Destexhe and Thierry. Bal. Dynamic-clamp: From principles to applications. *From Principles to Applications*, 1:443, 2009. ISSN 0387892796. doi: 10.1007/978-0-387-89279-5.

Ilka Diester, Matthew T. Kaufman, Murtaza Mogri, Ramin Pashaie, Werapong Goo, Ofer Yizhar, Charu Ramakrishnan, Karl Deisseroth, and Krishna V. Shenoy. An optogenetic toolbox designed for primates. *Nature Neuroscience*, 14:387–397, 1 2011. ISSN 1546-1726. doi: 10.1038/nn.2749. URL https://www.nature.com/articles/nn.2749.

Sven-Thorsten Dietrich and Daniel Walker. The evolution of Real-Time Linux. pages 3–4, 2005. URL http://linuxdevices.linuxgizmos.com/ldfiles/rtlws-2005/SvenThorstenDietrich.pdf.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55:78–87, 10 2012. ISSN 00010782. doi: 10.1145/2347736.2347755. URL https://dl.acm.org/doi/abs/10.1145/2347736.2347755.

Alan D. Dorval, David J. Christini, and John A. White. Real-Time Linux dynamic clamp: A fast and flexible way to construct virtual ion channels in living cells. *Annals of Biomedical Engineering*, 29:897–907, 10 2001. ISSN 1573-9686. doi: 10.1114/1.1408929. URL https://doi.org/10.1114/1.1408929.

Lorenzo Dozio and Paolo Mantegazza. Linux Real Time Application Interface (RTAI) in low cost high performance motion control. 2003. URL https://www.rtai.org/userfiles/documentation/documents/motioncontrol2003.pdf.

Shayok Dutta, Etienne Ackermann, and Caleb Kemere. Analysis of an open source, closed-loop, realtime system for hippocampal sharp-wave ripple disruption. *Journal of Neural Engineering*, 16, 12 2019. ISSN 1741-2552. doi: 10.1088/1741-2552/AAE90E. URL https://iopscience.iop.org/article/10.1088/1741-2552/aae90e.

Florin Dzeladini, Nadine Ait-Bouziad, and Auke Ijspeert. *CPG-Based Control of Humanoid Robot Locomotion*, pages 1–35. Springer, Dordrecht, 2017. doi: 10.1007/978-94-007-7194-9_49-1. URL https://link.springer.com/referenceworkentry/10.1007/978-94-007-7194-9_49-1.

Irene Elices and Pablo Varona. Closed-loop control of a minimal central pattern generator network. *Neurocomputing*, 170:55–62, 12 2015. ISSN 09252312. doi: 10.1016/j.neucom.2015.04.097. URL http://linkinghub.elsevier.com/retrieve/pii/S0925231215008796.

Irene Elices and Pablo Varona. Asymmetry factors shaping regular and irregular bursting rhythms in central pattern generators. *Frontiers in computational neuroscience*, 11:9, 2017. doi: 10.3389/fncom.2017.00009. URL http://www.ncbi.nlm.nih.gov/pubmed/28261081.

Irene Elices, Rafael Levi, David Arroyo, Francisco B. Rodriguez, and Pablo Varona. Robust dynamical invariants in sequential neural activity. *Scientific Reports*, 9:1–13, 6 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-44953-2. URL https://www.nature.com/articles/s41598-019-44953-2.

Zhong-Ping Feng. Synaptic transmission: Model systems. *Encyclopedia of Neuroscience*, pages 3971–3976, 11 2009. doi: 10.1007/978-3-540-29678-2_5825. URL https://link.springer.com/referenceworkentry/10.1007/978-3-540-29678-2_5825.

Antonio Fernández-Ruiz, Azahara Oliva, Eliezyer Fermino de Oliveira, Florbela Rocha-Almeida, David Tingley, and György Buzsáki. Long-duration hippocampal sharp wave ripples improve memory. *Science*, 364:1082–1086, 6 2019. ISSN 0036-8075. doi: 10.1126/science.aax0758. URL https://www.science.org/doi/10.1126/science.aax0758.

Stanley Finger. *Origins of neuroscience: a history of explorations into brain function*. Oxford University Press, USA, 2001.

N. A. Fitzsimmons, W. Drake, T. L. Hanson, M. A. Lebedev, and M. A.L. Nicolelis. Primate reaching cued by multichannel spatiotemporal cortical microstimulation. *Journal of Neuroscience*, 27:5593–5602, 5 2007. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.5297-06.2007. URL https://www.jneurosci.org/content/27/21/5593.

Dario Floreano and Francesco Mondada. Automatic creation of an autonomous agent. pages 421–430. The MIT Press, 2014. ISBN 978-0262531221. doi: 10.3929/ETHZ-A-010111549. URL https://doi.org/10.3929/ethz-a-010111549.

Caroline Garcia Forlim, Reynaldo Daniel Pinto, Pablo Varona, and Francisco B. Rodriguez. Delay-dependent response in weakly electric fish under closed-loop pulse stimulation. *PLOS ONE*, 10:1–14, 4 2015. doi: 10.1371/journal.pone.0141007. URL https://doi.org/10.1371/journal.pone.0141007.

Felix Franke, David Jäckel, Jelena Dragas, Jan Müller, Milos Radivojevic, Douglas Bakkum, and Andreas Hierlemann. High-density microelectrode array recordings and real-time spike sorting for closed-loop experiments: an emerging technology to study neural plasticity. *Frontiers in Neural Circuits*, 6:105, 2012. ISSN 1662-5110. doi: 10.3389/fncir.2012.00105. URL http://journal.frontiersin.org/article/10.3389/fncir.2012.00105/abstract.

Karl J. Friston, André M. Bastos, Dimitris Pinotsis, and Vladimir Litvak. LFP and oscillations—what do they tell us? *Current Opinion in Neurobiology*, 31:1–6, 4 2015. ISSN 0959-4388. doi: 10.1016/J.CONB.2014.05.004.

G. Fritsch and E. Hitzig. Ueber die elektrische Erregbarkeit des Grosshirns. *Archiv fur Anatomie, Physiologie und wissenschaftliche Medizin*, 37:300–339, 1870.

Leonardo A. Frizon, Erin A. Yamamoto, Sean J. Nagel, Marian T. Simonson, Olivia Hogue, and Andre G. Machado. Deep brain stimulation for pain in the modern era: A systematic review. *Clinical Neurosurgery*, 86:191–202, 2 2020. ISSN 15244040. doi: 10.1093/NEUROS/NYY552. URL https://journals.lww.com/neurosurgery/Fulltext/2020/02000/Deep_Brain_Stimulation_for_Pain_in_the_Modern_Era_.4.aspx.

Borko Furht, Dan Grostick, David Gluch, Guy Rabbat, John Parker, and Meg McRoberts. *Introduction to Real-Time Computing*, pages 1–35. Springer US, 1991. doi: 10.1007/978-1-4615-3978-0_1. URL http://link.springer.com/10.1007/978-1-4615-3978-0_1.

Luigi Galvani. *Aloysii Galvani De viribus electricitatis in motu musculari commentarius*. Ex Typographia Instituti Scientiarium, 1791. doi: 10.5479/SIL.324681.39088000932442.

Carlos Garcia-Saura. Central pattern generators for the control of robotic systems, 9 2015. URL https://arxiv.org/abs/1509.02417v1.

Carlos Garcia-Saura, Francisco B. Rodriguez, and Pablo Varona. Design principles for cooperative robots with uncertainty-aware and resource-wise adaptive behavior. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8608 LNAI:108–117, 2014. ISSN 16113349. doi: 10.1007/978-3-319-09435-9_10. URL https://link.springer.com/chapter/10.1007/978-3-319-09435-9_10.

Alicia Garrido-Peña. Hybrid experimental and model approaches for the caracterization of lymnaea stagnalis neural activity. Part. II: Design and implementation of hybrid circuits in Lymnaea stagnalis. Master's thesis, Universidad Autónoma de Madrid, 2019.

Richard George, Michela Chiappalone, Michele Giugliano, Timothée Levi, Stefano Vassanelli, Johannes Partzsch, and Christian Mayr. Plasticity and adaptation in neuromorphic biohybrid systems. *iScience*, 23, 10 2020. ISSN 25890042. doi: 10.1016/J.ISCI.2020.101589.

R M Ghigliazza and P Holmes. Minimal models of bursting neurons: How multiple currents, conductances, and timescales affect bifurcation diagrams. *Society for Industrial and Applied Mathematics*, 3:636–670, 2004. doi: 10.1137/030602307. URL https://epubs.siam.org/doi/pdf/10.1137/030602307#equation.4.1.

Gabrielle Girardeau, Karim Benchenane, Sidney I. Wiener, György Buzsáki, and Michaël B. Zugaro. Selective suppression of hippocampal ripples impairs spatial memory. *Nature Neuroscience*, 12:1222–1223, 9 2009. ISSN 1546-1726. doi: 10.1038/nn.2384. URL https://www.nature.com/articles/nn.2384.

Thomas Gleixner and Douglas Niehaus. Hrtimers and beyond: Transforming the Linux time subsystems. pages 333–346, 2006. URL https://www.landley.net/kdocs/ols/2006/ols2006v1-pages-333-346.pdf.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. pages 315–323. PMLR, 2011. URL https://proceedings.mlr.press/v15/glorot11a.html.

Jorge Golowasch, Michael Casey, L F Abbott, and Eve Marder. Network stability from activity-dependent regulation of neuronal conductances. *Neural Computation*, 11:1079–1096, 1999. ISSN 0899-7667. doi: 10.1162/089976699300016359. URL http://www.mitpressjournals.org/doi/10.1162/089976699300016359.

Rachel Grashow, Ted Brookings, and Eve Marder. Compensation for variable intrinsic neuronal excitability by circuit-synaptic interactions. *Journal of Neuroscience*, 30:9145–9156, 2010. doi: 10.1523%2FJNEUROSCI.0980-10.2010.

Idan Greenberg and Yair Manor. Synaptic depression in conjunction with a-current channels promote phase constancy in a rhythmic network. *Journal of Neurophysiology*, 93:656–677, 2 2005. ISSN 0022-3077. doi: 10.1152/jn.00640.2004. URL http://www.ncbi.nlm.nih.gov/pubmed/15356180.

Sten Grillner. The motor infrastructure: from ion channels to neuronal networks. *Nature Reviews Neuroscience*, 4:573–586, 7 2003. ISSN 1471003X. doi: 10.1038/nrn1137. URL http://www.nature.com/doifinder/10.1038/nrn1137.

Martin Grimmer and André Seyfarth. *Mimicking Human-Like Leg Function in Prosthetic Limbs*, pages 105–155. Springer, Dordrecht, 2014. doi: 10.1007/978-94-017-8932-5_5. URL https://link.springer.com/chapter/10.1007/978-94-017-8932-5_5.

Andres D. Grosmark and György Buzsáki. Diversity in neural firing dynamics supports both rigid and learned hippocampal sequences. *Science*, 351:1440–1443, 3 2016. ISSN 10959203. doi: 10.1126/SCIENCE.AAD1935. URL https://www.science.org/doi/full/10.1126/science.aad1935.

Dámaris K. Rangel Guerrero, James G. Donnett, Jozsef Csicsvari, and Krisztián A. Kovács. Tetrode recording from the hippocampus of behaving mice coupled with four-point-irradiation closed-loop optogenetics: A technique to study the contribution of hippocampal SWR events to learning. *eNeuro*, 5, 7 2018. ISSN 2373-2822. doi: 10.1523/ENEURO.0087-18.2018. URL https://www.eneuro.org/content/5/4/ENEURO.0087-18.2018.

Maria Guix, Rafael Mestre, Tania Patiño, Marco de Corato, Judith Fuentes, Giulia Zarpellon, and Samuel Sánchez. Biohybrid soft robots with self-stimulating skeletons. *Science Robotics*, 6, 4 2021. ISSN 24709476. doi: 10.1126/SCIROBOTICS.ABE7577. URL https://www.science.org/doi/abs/10.1126/scirobotics.abe7577.

Espen Hagen, Anna R. Chambers, Gaute T. Einevoll, Klas H. Pettersen, Rune Enger, and Alexander J. Stasik. RippleNet: a recurrent neural network for sharp wave ripple (SPW-R) detection. *Neuroinformatics*, 19:493–514, 7 2021. ISSN 15590089. doi: 10.1007/S12021-020-09496-2/. URL https://link.springer.com/article/10.1007/s12021-020-09496-2.

Prasanna Hambarde, Rachit Varma, and Shivani Jha. The survey of real time operating system: Rtos. pages 34–39. IEEE, 1 2014. ISBN 978-1-4799-2102-7. doi: 10.1109/ICESC.2014.15. URL http://ieeexplore.ieee.org/document/6745342/.

Hananel Hazan and Noam E. Ziv. Closed loop experiment manager (CLEM)—an open and inexpensive solution for multichannel electrophysiological recordings and closed loop experiments. *Frontiers in Neuroscience*, 11:579, 10 2017. ISSN 1662-453X. doi: 10.3389/fnins.2017.00579. URL http://journal.frontiersin.org/article/10.3389/fnins.2017.00579/full.

D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, 1949. URL https://psycnet.apa.org/record/1950-02200-000.

Luís Henriques. Threaded IRQs on Linux PREEMPT-RT. pages 23–32, 2009. URL http://www.artist-embedded.org/docs/Events/2009/OSPERT/OSPERT09-Henriques.pdf.

F. Herrero-Carrón, F. B. Rodríguez, and P. Varona. Bio-inspired design strategies for central pattern generator control in modular robotics. *Bioinspiration and Biomimetics*, 6:016006, 2 2011. ISSN 1748-3190. doi: 10.1088/1748-3182/6/1/016006. URL https://iopscience.iop.org/article/10.1088/1748-3182/6/1/016006.

J L Hindmarsh and R M Rose. A model of neuronal bursting using three coupled first order differential equations. *Proceedings of the Royal Society of London. Series B, Biological sciences*, 221:87–102, 3 1984. ISSN 0950-1193. URL http://www.ncbi.nlm.nih.gov/pubmed/6144106.

Mark H. Histed, Vincent Bonin, and R. Clay Reid. Direct activation of sparse, distributed populations of cortical neurons by electrical microstimulation. *Neuron*, 63:508–522, 8 2009. ISSN 0896-6273. doi: 10.1016/J.NEURON.2009.07.016.

A. L. Hodgkin and A. F. Huxley. Action potentials recorded from inside a nerve fibre. *Nature*, 144:710–711, 1939. ISSN 1476-4687. doi: 10.1038/144710a0. URL https://www.nature.com/articles/144710a0.

A. L. Hodgkin and A. F. Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo. *The Journal of Physiology*, 116:449–472, 4 1952a. doi: 10.1113/jphysiol.1952.sp004717. URL http://doi.wiley.com/10.1113/jphysiol.1952.sp004717.

A. L. Hodgkin and A. F. Huxley. The components of membrane conductance in the giant axon of Loligo. *The Journal of Physiology*, 116:473–496, 4 1952b. doi: 10.1113/jphysiol.1952.sp004718. URL http://doi.wiley.com/10.1113/jphysiol.1952.sp004718.

A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117:500–44, 8 1952c. URL http://www.ncbi.nlm.nih.gov/pubmed/12991237http://www.ncbi.nlm.nih.gov/pubmed/12991237.

A. L. Hodgkin, A. F. Huxley, and B. Katz. Measurement of current-voltage relations in the membrane of the giant axon of Loligo. *The Journal of Physiology*, 116:424–448, 4 1952. doi: 10.1113/jphysiol.1952.sp004716. URL http://doi.wiley.com/10.1113/jphysiol.1952.sp004716.

Merrel T. Holley, Neerajha Nagarajan, Christian Danielson, Pinar Zorlutuna, and Kidong Park. Development and characterization of muscle-based actuators for self-stabilizing swimming biorobots. *Lab on a Chip*, 16:3473–3484, 8 2016. ISSN 1473-0189. doi: 10.1039/C6LC00681G. URL https://pubs.rsc.org/en/content/articlehtml/2016/lc/c6lc00681g.

Guosong Hong and Charles M. Lieber. Novel electrode technologies for neural recordings. *Nature Reviews Neuroscience*, 20:330–345, 3 2019. ISSN 1471-0048. doi: 10.1038/s41583-019-0140-6. URL https://www.nature.com/articles/s41583-019-0140-6.

Ryan M Hooper, Ruben A Tikidji-Hamburyan, Carmen C Canavier, and Astrid A Prinz. Feedback control of variability in the cycle period of a central pattern generator. *Journal of Neurophysiology*, 114:jn.00365.2015, 11 2015. ISSN 0022-3077. doi: 10.1152/jn.00365.2015.

Jiaqi V. Huang, Yiran Wei, and Holger G. Krapp. A biohybrid fly-robot interface system that performs active collision avoidance. *Bioinspiration and Biomimetics*, 14, 9 2019. ISSN 1748-3190. doi: 10.1088/1748-3190/AB3B23. URL https://iopscience.iop.org/article/10.1088/1748-3190/ab3b23.

R. Huerta, P. Varona, M. I. Rabinovich, and Henry D. I. Abarbanel. Topology selection by chaotic neurons of a pyloric central pattern generator. *Biological Cybernetics*, 84:L1–L8, 1 2001. ISSN 0340-1200. doi: 10.1007/PL00007976. URL http://link.springer.com/10.1007/PL00007976.

Christopher L. Hughes, Sharlene N. Flesher, Jeffrey M. Weiss, Michael Boninger, Jennifer L. Collinger, and Robert A. Gaunt. Perception of microstimulation frequency in human somatosensory cortex. *eLife*, 10, 7 2021. ISSN 2050084X. doi: 10.7554/ELIFE.65128.

T E Hull, W H Enright, B M Fellen, A E Sedgwick, and A E Sedgwickt. Comparing numerical methods for ordinary differential equations. *SIAM Journal on Numerical Analysis*, 9:603–637, 1972. URL http://www.jstor.org/stable/2156215.

Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Netw*, 21(4):642–653, may 2008. doi: 10.1016/j.neunet.2008.03.014. URL http://dx.doi.org/10.1016/j.neunet.2008.03.014.

Auke Jan Ijspeert, Alessandro Crespi, Dimitri Ryczko, and Jean Marie Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315:1416–1420, 3 2007. ISSN 00368075. doi: 10.1126/SCIENCE.1138353. URL https://www.science.org/doi/10.1126/science.1138353.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2 2015. doi: 10.48550/arxiv.1502.03167. URL https://arxiv.org/abs/1502.03167v3.

Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactionson Neural Networks*, 14, 2003. doi: 10.1109/TNN.2003.820440.

Shantanu P. Jadhav, Caleb Kemere, P. Walter German, and Loren M. Frank. Awake hippocampal sharp-wave ripples support spatial memory. *Science*, 336:1454–1458, 6 2012. ISSN 10959203. doi: 10.1126/science.1217230. URL https://www.science.org/doi/full/10.1126/science.1217230.

Hannah R. Joo and Loren M. Frank. The hippocampal sharp wave–ripple in memory retrieval for immediate use and consolidation. *Nature Reviews Neuroscience*, 19:744–757, 10 2018. ISSN 1471-0048. doi: 10.1038/s41583-018-0077-1. URL https://www.nature.com/articles/s41583-018-0077-1.

James J. Jun, Nicholas A. Steinmetz, Joshua H. Siegle, Daniel J. Denman, Marius Bauza, Brian Barbarits, Albert K. Lee, Costas A. Anastassiou, Alexandru Andrei, Çağatay Aydin, Mladen Barbic, Timothy J. Blanche, Vincent Bonin, João Couto, Barundeb Dutta, Sergey L. Gratiy, Diego A. Gutnisky, Michael Häusser, Bill Karsh, Peter Ledochowitsch, Carolina Mora Lopez, Catalin Mitelut, Silke Musa, Michael Okun, Marius Pachitariu, Jan Putzeys, P. Dylan Rich, Cyrille Rossant, Wei Lung Sun, Karel Svoboda, Matteo Carandini, Kenneth D. Harris, Christof Koch, John O'Keefe, and Timothy D. Harris. Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551:232–236, 11 2017. ISSN 1476-4687. doi: 10.1038/nature24636. URL https://www.nature.com/articles/nature24636.

Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell. *Synaptic Transmission*, page 175. McGraw-Hill, 5th edition, 2013. ISBN 9780071390118.

Matthew Keennon, Karl Klingebiel, Henry Won, and Alexander Andriukov. Development of the nano hummingbird: A tailless flapping wing micro air vehicle. *50th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2012. doi: 10.2514/6.2012-588. URL https://arc.aiaa.org/doi/abs/10.2514/6.2012-588.

Helmut Kettenmann and Alexei Verkhratsky. Neuroglia: the 150 years after. *Trends in Neurosciences*, 31:653–659, 12 2008. ISSN 0166-2236. doi: 10.1016/J.TINS.2008.09.003.

Jinseok Kim, Jungyul Park, Sungwook Yang, Jeongeun Baek, Byungkyu Kim, Sang Ho Lee, Eui Sung Yoon, Kukjin Chun, and Sukho Park. Establishment of a fabrication method for a long-term actuated hybrid cell robot. *Lab on a Chip*, 7:1504–1508, 10 2007. ISSN 1473-0189. doi: 10.1039/B705367C. URL https://pubs.rsc.org/en/content/articlehtml/2007/lc/b705367c.

Sangbae Kim, Matthew Spenko, Salomon Trujillo, Barrett Heyneman, Daniel Santos, and Mark R. Cutkoskly. Smooth vertical surface climbing with directional adhesion. *IEEE Transactions on Robotics*, 24:65–74, 2 2008. ISSN 15523098. doi: 10.1109/TRO.2007.909786.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014. doi: 10.48550/arxiv.1412.6980. URL https://arxiv.org/abs/1412.6980v9.

Tilman J. Kispersky, Michael N. Economo, Pratik Randeria, and John A. White. GenNet: A platform for hybrid network experiments. *Frontiers in Neuroinformatics*, 5:11, 2011. ISSN 1662-5196. doi: 10.3389/fninf.2011.00011. URL http://journal.frontiersin.org/article/10.3389/fninf.2011.00011/abstract.

Alexander O. Komendantov and Nikolai I. Kononenko. Deterministic chaos in mathematical model of pacemaker activity in bursting neurons of snail, Helix Pomatia. *Journal of Theoretical Biology*, 183:219–230, 11 1996. ISSN 0022-5193. doi: 10.1006/JTBI.1996.0215.

Esther Krook-Magnuson, Caren Armstrong, Mikko Oijala, and Ivan Soltesz. On-demand optogenetic control of spontaneous seizures in temporal lobe epilepsy. *Nature communications*, 4: 1376, 2013. ISSN 2041-1723. doi: 10.1038/ncomms2376. URL http://www.ncbi.nlm.nih.gov/pubmed/23340416.

Peter J. Kyberd, Colin Light, Paul H. Chappell, Jim M. Nightingale, Dave Whatley, and Mervyn Evans. The design of anthropomorphic prosthetic hands: A study of the southampton hand. *Robotica*, 19:593–600, 2001. ISSN 1469-8668. doi: 10.1017/S0263574701003538.

Peter M. Lalley. *Intracellular Recording*, pages 2019–2026. Springer, Berlin, Heidelberg, 11 2009. doi: 10.1007/978-3-540-29678-2_2554. URL https://link.springer.com/referenceworkentry/10.1007/978-3-540-29678-2_2554.

Angel Lareo, Caroline G Forlim, Reynaldo D Pinto, Pablo Varona, and Francisco de Borja Rodriguez. Temporal code-driven stimulation: Definition and application to electric fish signaling. *Frontiers in Neuroinformatics*, 10:41, 2016. doi: 10.3389/fninf.2016.00041. URL http://www.ncbi.nlm.nih.gov/pubmed/27766078.

Moritz Leber, Julia Körner, Christopher F. Reiche, Ming Yin, Rajmohan Bhandari, Robert Franklin, Sandeep Negi, and Florian Solzbacher. Advances in penetrating multichannel microelectrodes based on the Utah array platform. *Advances in Experimental Medicine and Biology*, 1101:1–40, 2019. ISSN 22148019. doi: 10.1007/978-981-13-2050-7_1. URL https://link.springer.com/chapter/10.1007/978-981-13-2050-7_1.

Timothee Levi, Paolo Bonifazi, Paolo Massobrio, and Michela Chiappalone. Editorial: Closed-loop systems for next-generation neuroprostheses. *Frontiers in Neuroscience*, 12:26, 2018. ISSN 1662-453X. doi: 10.3389/fnins.2018.00026. URL https://www.frontiersin.org/article/10.3389/fnins.2018.00026.

Fen Li, Ming Liu, Yuejin Zhao, Lingqin Kong, Liquan Dong, Xiaohua Liu, and Mei Hui. Feature extraction and classification of heart sound using 1D convolutional neural networks. *EURASIP Journal on Advances in Signal Processing*, 2019:1–11, 12 2019. ISSN 16876180. doi: 10.1186/S13634-019-0651-3. URL https://asp-eurasipjournals.springeropen.com/articles/10.1186/s13634-019-0651-3.

Yongcheng Li, Rong Sun, Yuechao Wang, Hongyi Li, and Xiongfei Zheng. A novel robot system integrating biological and mechanical intelligence based on dissociated neural network-controlled closed-loop environment. *PLOS ONE*, 11:e0165600, 11 2016. ISSN 1932-6203. doi: 10.1371/JOURNAL.PONE.0165600. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0165600.

Daniele Linaro, João Couto, and Michele Giugliano. Command-line cellular electrophysiology for conventional and real-time closed-loop experiments. *Journal of Neuroscience Methods*, 230:5–19, 2014. ISSN 01650270. doi: 10.1016/j.jneumeth.2014.04.003. URL http://www.sciencedirect.com/science/article/pii/S0165027014001198.

Grace W. Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience*, 33:2017–2031, 9 2021. ISSN 0898-929X. doi: 10.1162/JOCN_A_01544. URL https://direct.mit.edu/jocn/article/33/10/2017/97402/Convolutional-Neural-Networks-as-a-Model-of-the.

Francisco López-Muñoz, Jesús Boya, and Cecilio Alamo. Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal. *Brain Research Bulletin*, 70:391–405, 10 2006. ISSN 0361-9230. doi: 10.1016/J.BRAINRESBULL.2006.07.010.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

P. Mantegazza, E. L. Dozio, and S. Papacharalambous. RTAI: Real-Time Application Interface. *Linux Journal*, 2000:10, 2000. URL http://dl.acm.org/citation.cfm?id=348564.

E Marder and J S Eisen. Transmitter identification of pyloric neurons: electrically coupled neurons use different transmitters. *Journal of neurophysiology*, 51:1345–61, 6 1984. ISSN 0022-3077. doi: 10.1152/jn.1984.51.6.1345. URL http://www.ncbi.nlm.nih.gov/pubmed/6145757.

G Le Masson, S Le Masson, and M Moulins. From conductances to neural network properties: analysis of simple circuits using the hybrid network method. *Progress in biophysics and molecular biology*, 64:201–20, 1995. ISSN 0079-6107. URL http://www.ncbi.nlm.nih.gov/pubmed/8987384.

Gwendal Le Masson, Sylvie Renaud-Le Masson, Damien Debay, and Thierry Bal. Feedback inhibition controls spike transfer in hybrid thalamic circuits. *Nature*, 417:854–858, 2002.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *ArXiv*, 4 2018. doi: 10.48550/arxiv.1804.07612. URL https://arxiv.org/abs/1804.07612v1.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 12 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL https://link.springer.com/article/10.1007/BF02478259.

Timothy P. Meehan and Steven L. Bressler. Neurocognitive networks: Findings, models, and theory. *Neuroscience and Biobehavioral Reviews*, 36:2232–2247, 11 2012. ISSN 0149-7634. doi: 10.1016/J.NEUBIOREV.2012.08.002.

Nature Methods. Method of the year 2010. *Nature Methods*, 8:1–1, 12 2010. ISSN 1548-7105. doi: 10.1038/nmeth.f.321. URL https://www.nature.com/articles/nmeth.f.321.

J. P. Miller and A. I. Selverston. Mechanisms underlying pattern generation in lobster stomatogastric ganglion as determined by selective inactivation of identified neurons. network properties of pyloric system. *Journal of Neurophysiology*, 48, 1982. URL http://jn.physiology.org/content/48/6/1416.long.

Ryo Minegishi, Atsushi Takashima, Daisuke Kurabayashi, and Ryohei Kanzaki. Construction of a brain–machine hybrid system to evaluate adaptability of an insect. *Robotics and Autonomous Systems*, 60:692–699, 5 2012. ISSN 0921-8890. doi: 10.1016/J.ROBOT.2011.06.012.

Meera E. Modi and Mustafa Sahin. Translational use of event-related potentials to assess circuit integrity in ASD. *Nature Reviews Neurology*, 13:160–170, 2 2017. ISSN 1759-4766. doi: 10.1038/nrneurol.2017.15. URL https://www.nature.com/articles/nrneurol.2017.15.

Wael Mohamed. The Edwin Smith surgical papyrus: Neuroscience in Ancient Egypt. *IBRO History of Neuroscience*, 2008. URL https://archive.ph/20140706060915/http://www.ibro1.info/Pub/Pub_Main_Display.asp?LC_Docs_ID=3199.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2nd edition, 2018. URL https://cs.nyu.edu/~mohri/mlbook/.

Joao Monteiro. Building your way through RTAI, 2008. URL https://code.ua.pt/attachments/download/2171/Starting_With_RTAI4.pdf.

Aaron Montero, Ramon Huerta, and Francisco B. Rodriguez. Regulation of specialists and generalists by neural variability improves pattern recognition performance. *Neurocomputing*, 151:69–77, 3 2015. ISSN 0925-2312. doi: 10.1016/J.NEUCOM.2014.09.073.

Yuya Morimoto, Hiroaki Onoe, and Shoji Takeuchi. Biohybrid robot powered by an antagonistic pair of skeletal muscle tissues. *Science Robotics*, 3, 5 2018. ISSN 24709476. doi: 10.1126/SCIROBOTICS.AAT4440. URL https://www.science.org/doi/abs/10.1126/scirobotics.aat4440.

May-Britt Moser and Edvard I Moser. Functional differentiation in the hippocampus. *Hippocampus*, 1999. doi: https://doi.org/10.1002/(SICI)1098-1063(1998)8:6%3C608::AID-HIPO3%3E3.0.CO;2-7.

Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, 2012. ISBN 0262304325. URL https://books.google.com/books/about/Machine_Learning.html?hl=es&id=RC43AgAAQBAJ.

Carlos Muñiz, Sara Arganda, Francisco B Rodriguez, Gonzalo G. Polavieja, and Pablo Varona. Realistic stimulation through advanced dynamic-clamp protocols. pages 95–105. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-31672-5.

Carlos Muñiz, Rafael Levi, Meriem Benkrid, Francisco B. Rodríguez, and Pablo Varona. Real-time control of stepper motors for mechano-sensory stimulation. *Journal of Neuroscience Methods*, 172:105–111, 2008. ISSN 01650270. doi: 10.1016/j.jneumeth.2008.04.017. URL http://www.sciencedirect.com/science/article/pii/S0165027008002446.

Carlos Muñiz, Francisco Rodríguez, and Pablo Varona. RtBiomanager: a software platform to expand the applications of real-time technology in neuroscience. *BMC Neuroscience*, 10:P49, 2009. ISSN 1471-2202. doi: 10.1186/1471-2202-10-S1-P49.

Carlos Muñiz, Caroline G Forlim, Rafael T Guariento, Reynaldo D Pinto, Francisco B Rodriguez, and Pablo Varona. Online video tracking for activity-dependent stimulation in neuroethology. *BMC Neuroscience*, 12:P358, 2011. doi: 10.1186/1471-2202-12-S1-P358. URL http://bmcneurosci.biomedcentral.com/articles/10.1186/1471-2202-12-S1-P358.

Jan Müller, Douglas J. Bakkum, and Andreas Hierlemann. Sub-millisecond closed-loop feedback stimulation between arbitrary sets of individual neurons. *Frontiers in Neural Circuits*, 6:121, 2013. doi: 10.3389/fncir.2012.00121. URL http://journal.frontiersin.org/article/10.3389/fncir.2012.00121/abstract.

Andrea Navas-Olive, Rodrigo Amaducci, Maria-Teresa Jurado-Parras, Enrique R Sebastian, and Liset M de la Prida. Deep learning based feature extraction for prediction and interpretation of sharp-wave ripples in the rodent hippocampus. *eLife*, 11, 9 2022. doi: 10.7554/ELIFE.77772. URL https://doi.org/10.7554/eLife.77772.

Sharon E Norman, Robert J Butera, and Carmen C Canavier. Stochastic slowly adapting ionic currents may provide a decorrelation mechanism for neural oscillators by causing wander in the intrinsic period. *Journal of Neurophysiology*, 2016. ISSN 0022-3077. doi: 10.1152/jn.00193.2016.

A. Novellino, P. D'Angelo, L. Cozzi, M. Chiappalone, V. Sanguineti, and S. Martinoia. Connecting neurons to a mobile robot: An in vitro bidirectional neural interface. *Computational Intelligence and Neuroscience*, 2007, 2007. ISSN 16875265. doi: 10.1155/2007/12725.

Thomas Nowotny and Pablo Varona. *Dynamic Clamp Technique*, pages 1048–1051. Springer New York, 2015. ISBN 978-1-4614-6675-8. doi: 10.1007/978-1-4614-6675-8_126. URL http://link.springer.com/10.1007/978-1-4614-7320-6_126-2.

Thomas Nowotny, Valentin P Zhigulin, Allan I Selverston, Henry D I Abarbanel, and Mikhail I Rabinovich. Enhancement of synchronization in a hybrid neural circuit by spike-timing dependent plasticity. *Journal of Neuroscience*, 23:9776–9785, 2003. ISSN 1529-2401. doi: 23/30/9776[pii].

Thomas Nowotny, Attila Szűcs, Reynaldo D. Pinto, and Allen I. Selverston. StdpC: A modern dynamic clamp. *Journal of Neuroscience Methods*, 158:287–299, 12 2006. ISSN 01650270. doi: 10.1016/j.jneumeth.2006.05.034. URL http://linkinghub.elsevier.com/retrieve/pii/S0165027006002810.

Thomas Nowotny, Rafael Levi, and Allen I. Selverston. Probing the dynamics of identified neurons with a data-driven modeling approach. *PLOS ONE*, 3:e2627, 7 2008. ISSN 1932-6203. doi: 10.1371/JOURNAL.PONE.0002627. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0002627.

H. Freyja Olafsdottir, Daniel Bush, and Caswell Barry. The role of hippocampal replay in memory and planning. *Current Biology*, 28:R37–R50, 1 2018. ISSN 0960-9822. doi: 10.1016/J.CUB.2017.10.073.

Azahara Oliva, Antonio Fernández-Ruiz, Felix Leroy, and Steven A. Siegelbaum. Hippocampal CA2 sharp-wave ripples reactivate and promote social memory. *Nature*, 587:264–269, 9 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2758-y. URL https://www.nature.com/articles/s41586-020-2758-y.

S A Oprisan, A A Prinz, and C C Canavier. Phase resetting and phase locking in hybrid circuits of one model and one biological neuron. *Biophysical journal*, 87:2283–2298, 2004. ISSN 00063495. doi: 10.1529/biophysj.104.046193.

Erika Pastrana. Optogenetics: controlling cell function with light. *Nature Methods*, 8:24–25, 12 2010. ISSN 1548-7105. doi: 10.1038/nmeth.f.323. URL https://www.nature.com/articles/nmeth.f.323.

Yogi A. Patel, Ansel George, Alan D. Dorval, John A. White, David J. Christini, and Robert J. Butera. Hard real-time closed-loop electrophysiology with the real-time experiment interface (RTXI). *PLOS Computational Biology*, 13:e1005430, 5 2017. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1005430. URL http://www.ncbi.nlm.nih.gov/pubmed/28557998.

R. D. Pinto, P. Varona, A. R. Volkovskii, A. Szücs, Henry D. I. Abarbanel, and M. I. Rabinovich. Synchronous behavior of two coupled electronic neurons. *Physical Review E*, 62:2644–2656, 8 2000. ISSN 1063-651X. doi: 10.1103/PhysRevE.62.2644. URL https://link.aps.org/doi/10.1103/PhysRevE.62.2644.

R.D. Pinto, R.C. Elson, A. Szücs, M.I. Rabinovich, A.I. Selverston, and H.D.I. Abarbanel. Extended dynamic clamp: controlling up to four neurons using a single desktop computer and interface. *Journal of Neuroscience Methods*, 108:39–48, 2001. ISSN 01650270. doi: 10.1016/S0165-0270(01)00368-5. URL http://www.sciencedirect.com/science/article/pii/S0165027001003685.

R. M.R. Pizzi, D. Rossetti, G. Cino, D. Marino, A.L.Vescovi, and W. Baer. A cultured human neural network operates a robotic actuator. *Biosystems*, 95:137–144, 2 2009. ISSN 0303-2647. doi: 10.1016/J.BIOSYSTEMS.2008.09.006.

Dean Pomerleau. *Neural Network Vision for Robot Driving*, pages 53–72. Springer, Boston, MA, 1997. doi: 10.1007/978-1-4615-6325-9_4. URL https://link.springer.com/chapter/10.1007/978-1-4615-6325-9_4.

Steve M. Potter, Thomas B. DeMarse, Douglas J. Bakkum, Mark C. Booth, John R. Brumfield, Zenas Chao, Radhika Madhavan, Peter A. Passaro, Komal Rambani Alexander C. Shkolnik, R. Blythe Towal, and Daniel A. Wagenaar. Hybrots: hybrids of living neurons and robots for studying neural computation. 2004. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.298.6887.

Steve M. Potter, Daniel A. Wagenaar, and Thomas B. Demarse. *Closing the Loop: Stimulation Feedback Systems for Embodied MEA Cultures*, pages 215–242. Springer, Boston, MA, 2006. ISBN 9780387258584. doi: 10.1007/0-387-25858-2_9. URL https://link.springer.com/chapter/10.1007/0-387-25858-2_9.

Steve M Potter, Ahmed El Hady, and Eberhard E Fetz. Closed-loop neuroscience and neuro-engineering. *Frontiers in Neural Circuits*, 8, 2014. doi: 10.3389/fncir.2014.00115.

William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Integration of Ordinary Differential Equations*, pages 710–714. Cambridge University Press, 1988. ISBN 0521880688. URL http://www.nr.comhttp://www.nr.com.

Liset M. De La Prida. Potential factors influencing replay across ca1 during sharp-wave ripples. *Philosophical Transactions of the Royal Society B*, 375, 5 2020. ISSN 14712970. doi: 10.1098/RSTB.2019.0236. URL https://royalsocietypublishing.org/doi/full/10.1098/rstb.2019.0236.

Astrid A Prinz, L F Abbott, and Eve Marder. The dynamic clamp comes of age. *Trends in Neurosciences*, 27:218–224, 4 2004a. ISSN 01662236. doi: 10.1016/j.tins.2004.02.004.

Astrid A Prinz, Dirk Bucher, and Eve Marder. Similar network activity from disparate circuit parameters. *Nature Neuroscience*, 7:1345–1352, 12 2004b. ISSN 1097-6256. doi: 10.1038/nn1352. URL http://www.ncbi.nlm.nih.gov/pubmed/15558066http://www.nature.com/articles/nn1352.

Mario Prsa, Gregorio L. Galiñanes, and Daniel Huber. Rapid integration of artificial sensory feedback during operant conditioning of motor cortex neurons. *Neuron*, 93:929–939.e6, 2017. ISSN 08966273. doi: 10.1016/j.neuron.2017.01.023. URL http://www.sciencedirect.com/science/article/pii/S0896627317300478.

Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal on Selected Topics in Signal Processing*, 13:206–219, 5 2019. ISSN 19410484. doi: 10.1109/JSTSP.2019.2908700.

Giovanni Racciu and Paolo Mantegazza. RTAI 3.4 user manual, 2006. URL https://www.rtai.org/userfiles/documentation/documents/RTAI_User_Manual_34_03.pdf.

P Raghavan, Amol Lad, and Sriram Neelakandan. *Embedded Linux system design and development*. Auerbach Publications, 2005. ISBN 9780367391416.

Mohit Rana, Andrew Q Varan, Anis Davoudi, Ronald A Cohen, Ranganatha Sitaram, and Natalie C Ebner. Real-time fMRI in neuroscience research and its use in studying the aging brain. *Frontiers in aging neuroscience*, 8:239, 2016. doi: 10.3389/fnagi.2016.00239. URL http://www.ncbi.nlm.nih.gov/pubmed/27803662http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5067937.

Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January: 6517–6525, 12 2016. doi: 10.48550/arxiv.1612.08242. URL https://arxiv.org/abs/1612.08242v1.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:779–788, 6 2015. ISSN 10636919. doi: 10.48550/arxiv.1506.02640. URL https://arxiv.org/abs/1506.02640v5.

Hernan Gonzalo Rey, Carlos Pedreira, and Rodrigo Quian Quiroga. Past, present and future of spike sorting techniques. *Brain Research Bulletin*, 119:106–117, 10 2015. ISSN 0361-9230. doi: 10.1016/J.BRAINRESBULL.2015.04.007.

Manuel Reyes-Sanchez, Rodrigo Amaducci, Irene Elices, Francisco B. Rodriguez, and Pablo Varona. Automatic adaptation of model neurons and connections to build hybrid circuits with living networks. *Neuroinformatics*, 18:377–393, 6 2020. ISSN 15590089. doi: 10.1007/S12021-019-09440-Z. URL https://link.springer.com/article/10.1007/s12021-019-09440-z.

H P Robinson and N Kawai. Injection of digitally synthesized synaptic conductance transients to measure the integrative properties of neurons. *Journal of neuroscience methods*, 49:157–65, 9 1993. ISSN 0165-0270. URL http://www.ncbi.nlm.nih.gov/pubmed/7903728.

W. R.T. Roderick, M. R. Cutkosky, and D. Lentink. Bird-inspired dynamic grasping and perching in arboreal environments. *Science Robotics*, 6, 12 2021. ISSN 24709476. doi: 10.1126/SCIROBOTICS.ABJ7562. URL https://www.science.org/doi/abs/10.1126/scirobotics.abj7562.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 11 1958. ISSN 0033295X. doi: 10.1037/H0042519.

E. Roth, S. Sponberg, and N. J. Cowan. A comparative approach to closed-loop computation. *Current Opinion in Neurobiology*, 25:54–62, 2014.

Demetris K. Roumis and Loren M. Frank. Hippocampal sharp-wave ripples in waking and sleeping states. *Current Opinion in Neurobiology*, 35:6–12, 12 2015. ISSN 0959-4388. doi: 10.1016/J.CONB.2015.05.001.

Nikolai F. Rulkov. Modeling of spiking-bursting neural behavior using two-dimensional map. *Physical Review E*, 65:041922, 4 2002. ISSN 1063-651X. doi: 10.1103/PhysRevE.65.041922. URL http://link.aps.org/doi/10.1103/PhysRevE.65.041922.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://www.nature.com/articles/323533a0.

Akira Sakurai, Charuni A. Gunaratne, and Paul S. Katz. Two interconnected kernels of reciprocally inhibitory interneurons underlie alternating left-right swim motor pattern generation in the mollusk melibe leonina. *Journal of Neurophysiology*, 112, 2014. doi: 10.1152/jn.00261.2014. URL http://jn.physiology.org/content/112/6/1317.long.

Uluc Saranli, Martin Buehler, and Daniel E. Koditschek. RHex: A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20:616–631, 7 2001. ISSN 02783649. doi: 10.1177/02783640122067570. URL https://journals.sagepub.com/doi/10.1177/02783640122067570.

Haruto Sawada, Naoki Wake, Kazuhiro Sasabuchi, Jun Takamatsu, Hirokazu Takahashi, and Katsushi Ikeuchi. Design strategies for controlling neuron-connected robots using reinforcement learning. *ArXiv*, 3 2022. doi: 10.48550/arxiv.2203.15290. URL https://arxiv.org/abs/2203.15290v1.

Massimo Scanziani and Michael Hausser. Electrophysiology in the age of light. *Nature*, 461:930–939, 10 2009. doi: 10.1038/nature08540. URL http://www.nature.com/doifinder/10.1038/nature08540.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 1 2015. ISSN 0893-6080. doi: 10.1016/J.NEUNET.2014.09.003.

Donald L. Schomer and Fernando H. Lopes da Silva. *Niedermeyer's electroencephalography : basic principles, clinical applications, and related fields*. Wolters Kluwer, 5th edition, 2005. ISBN 0190228482.

Terrence J. Sejnowski. The unreasonable effectiveness of deep learning in artificial intelligence. *Proceedings of the National Academy of Sciences of the United States of America*, 117:30033–30038, 12 2020. ISSN 10916490. doi: 10.1073/PNAS.1907373117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1907373117.

Allen I. Selverston, Mikhail I. Rabinovich, Henry D.I. Abarbanel, Robert Elson, Attila Szücs, Reynaldo D. Pinto, Ramón Huerta, and Pablo Varona. Reliable circuits from irregular neurons: A dynamical approach to understanding central pattern generators. *Journal of Physiology-Paris*, 94:357–374, 2000. ISSN 09284257. doi: 10.1016/S0928-4257(00)01101-3. URL http://www.sciencedirect.com/science/article/pii/S0928425700011013.

L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39:1175–1185, 1990. ISSN 00189340. doi: 10.1109/12.57058. URL http://ieeexplore.ieee.org/document/57058/.

Philip Shamash, Matteo Carandini, Kenneth Harris, and Nick Steinmetz. A tool for analyzing electrode tracks from slice histology. *bioRxiv*, 10 2018. doi: 10.1101/447995. URL https://www.biorxiv.org/content/10.1101/447995v1.

A A Sharp, M B ONeil, L F Abbott, and E Marder. Dynamic clamp: computer-generated conductances in real neurons. *Journal of Neurophysiology*, 69:992–995, 1993. ISSN 0022-3077. URL http://jn.physiology.org/content/69/3/992.

K.G. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82:6–24, 1994. ISSN 00189219. doi: 10.1109/5.259423. URL http://ieeexplore.ieee.org/document/259423/.

Joshua H Siegle, Aarón Cuevas López, Yogi A Patel, Kirill Abramov, Shay Ohayon, and Jakob Voigts. Open Ephys: an open-source, plugin-based platform for multichannel electrophysiology. *Journal of Neural Engineering*, 14:045003, 8 2017. ISSN 1741-2560. doi: 10.1088/1741-2552/aa5eea. URL http://stacks.iop.org/1741-2552/14/i=4/a=045003?key=crossref.72f4c42751bd1cca996f430a5a7a3758.

Ranganatha Sitaram, Tomas Ros, Luke Stoeckel, Sven Haller, Frank Scharnowski, Jarrod Lewis-Peacock, Nikolaus Weiskopf, Maria Laura Blefari, Mohit Rana, Ethan Oblak, Niels Birbaumer, and James Sulzer. Closed-loop brain training: the science of neurofeedback. *Nature Reviews Neuroscience*, 18:86–100, 12 2016. ISSN 1471-003X. doi: 10.1038/nrn.2016.164. URL http://www.ncbi.nlm.nih.gov/pubmed/28003656http://www.nature.com/doifinder/10.1038/nrn.2016.164.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, 12 2014. doi: 10.48550/arxiv.1412.6806. URL https://arxiv.org/abs/1412.6806v3.

Shrey Srivastava, Amit Vishvas Divekar, Chandu Anilkumar, Ishika Naik, Ved Kulkarni, and V. Pattabiraman. Comparative analysis of deep learning image detection algorithms. *Journal of Big Data*, 8:1–27, 12 2021. ISSN 21961115. doi: 10.1186/S40537-021-00434-W/TABLES/2. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00434-w.

William Stallings. *Multiprocessor and Real-Time Scheduling*, pages 430–474. Prentice Hall, 7 edition, 2012. ISBN 978-0132309981. URL http://dinus.ac.id/repository/docs/ajar/Operating_System.pdf.

R. G. D. Steel and J. H. Torrie. *Principles and Procedures of Statistics*. McGraw-Hill Book Company, 1 1960.

M. Steriade, P. Gloor, R. R. Llinás, F. H. Lopes da Silva, and M. M. Mesulam. Basic mechanisms of cerebral rhythmic activities. *Electroencephalography and Clinical Neurophysiology*, 76:481–508, 12 1990. ISSN 0013-4694. doi: 10.1016/0013-4694(90)90001-Z.

Ian H. Stevenson and Konrad P. Kording. How advances in neural recording affect data analysis. *Nature Neuroscience*, 14:139–142, 1 2011. ISSN 1546-1726. doi: 10.1038/nn.2731. URL https://www.nature.com/articles/nn.2731.

Friedrich Striggow, Mariana Medina-Sánchez, Günter K Auernhammer, Veronika Magdanz, Benjamin M Friedrich, and Oliver G Schmidt. Sperm-driven micromotors moving in oviduct fluid and viscoelastic media. *Small*, 16:2000213, 6 2020. ISSN 1613-6829. doi: 10.1002/SMLL.202000213.

Ho Jun Suk, Ingrid van Welie, Suhasa B. Kodandaramaiah, Brian Allen, Craig R. Forest, and Edward S. Boyden. Closed-loop real-time imaging enables fully automated cell-targeted patch-clamp neural recording in vivo. *Neuron*, 95, 8 2017. ISSN 0896-6273. doi: 10.1016/J.NEURON.2017.08.011.

Mario Svirsky. Cochlear implants and electronic hearing. *Physics Today*, 70:52, 8 2017. ISSN 0031-9228. doi: 10.1063/PT.3.3661. URL https://physicstoday.scitation.org/doi/abs/10.1063/PT.3.3661.

Attila Szücs, Pablo Varona, Alexander R. Volkovskii, Henry D.I. Abarbanel, Mikhail I. Rabinovich, and Allen I. Selverston. Interacting biological and electronic neurons generate realistic oscillatory rhythms. *Neuroreport*, 11:563–569, 2000. ISSN 0959-4965. doi: 10.1097/00001756-200002280-00027. URL https://pubmed.ncbi.nlm.nih.gov/10718315/.

Yo Tanaka, Shun ichi Funano, Yuji Noguchi, Yaxiaer Yalikun, and Norihiro Kamamichi. A valve powered by earthworm muscle with both electrical and 100-per-cent chemical control. *Scientific Reports*, 9:1–10, 7 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-44116-3. URL https://www.nature.com/articles/s41598-019-44116-3.

Jacopo Tessadori, Marta Bisio, Sergio Martinoia, and Michela Chiappalone. Modular neuronal assemblies embodied in a closed-loop environment: Toward future integration of brains and machines. *Frontiers in Neural Circuits*, 6:99, 2012. ISSN 1662-5110. doi: 10.3389/fncir.2012.00099. URL http://journal.frontiersin.org/article/10.3389/fncir.2012.00099/abstract.

Umeshkanta S Thounaojam, Jianxia Cui, Sharon E Norman, Robert J Butera, and Carmen C Canavier. Slow noise in the period of a biological oscillator underlies gradual trends and abrupt transitions in phasic relationships in hybrid neural networks. *PLoS Computational Biology*, 10, 2014. ISSN 15537358. doi: 10.1371/journal.pcbi.1003622.

Joaquin J. Torres and Pablo Varona. *Modeling Biological Neural Networks*, pages 533–564. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-540-92910-9_17. URL http://link.springer.com/10.1007/978-3-540-92910-9_17.

Joaquin J. Torres, Fabiano Baroni, Roberto Latorre, and Pablo Varona. Temporal discrimination from the interaction between dynamic synapses and intrinsic subthreshold oscillations. *Neurocomputing*, 417:543–557, 12 2020. ISSN 0925-2312. doi: 10.1016/J.NEUCOM.2020.07.031. URL https://doi.org/10.1016/j.neucom.2020.07.031.

P. Varona, J. J. Torres, H. D.I. Abarbanel, M. I. Rabinovich, and R. C. Elson. Dynamics of two electrically coupled chaotic neurons: Experimental observations and model analysis. *Biological Cybernetics*, 84:91–101, 2001. ISSN 1432-0770. doi: 10.1007/S004220000198. URL https://link.springer.com/article/10.1007/s004220000198.

P. Varona, D. Arroyo, F.B. Rodríguez, and T. Nowotny. *Online Event Detection Requirements in Closed-Loop Neuroscience*, pages 81–91. Elsevier, 2016. ISBN 9780128024522. doi: 10.1016/B978-0-12-802452-2.00006-8. URL http://linkinghub.elsevier.com/retrieve/pii/B9780128024522000068.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. pages 1058–1066. PMLR, 2013.

Shuoguo Wang, Lakshmi Chandrasekaran, Fernando R Fernandez, John A White, and Carmen C Canavier. Short conduction delays cause inhibition rather than excitation to favor synchrony in hybrid neuronal networks of the entorhinal cortex. *PLoS computational biology*, 8, 2012.

Ting Wang, Ming Wang, Jianwu Wang, Le Yang, Xueyang Ren, Gang Song, Shisheng Chen, Yuehui Yuan, Ruiqing Liu, Liang Pan, Zheng Li, Wan Ru Leow, Yifei Luo, Shaobo Ji, Zequn Cui, Ke He, Feilong Zhang, Fengting Lv, Yuanyuan Tian, Kaiyu Cai, Bowen Yang, Jingyi Niu, Haochen Zou, Songrui Liu, Guoliang Xu, Xing Fan, Benhui Hu, Xian Jun Loh, Lianhui Wang, and Xiaodong Chen. A chemically mediated artificial neuron. *Nature Electronics*, 5 (9):586–595, 2022a. ISSN 2520-1131. doi: 10.1038/s41928-022-00803-0. URL https://doi.org/10.1038/s41928-022-00803-0.

X J Wang. Ionic basis for intrinsic 40 Hz neuronal oscillations. *Neuroreport*, 5:221–224, 12 1993. ISSN 0959-4965. doi: 10.1097/00001756-199312000-00008. URL http://www.ncbi.nlm.nih.gov/pubmed/8298079.

Yanchao Wang, Ye Tian, Haotian She, Yinlai Jiang, Hiroshi Yokoi, and Yunhui Liu. Design of an effective prosthetic hand system for adaptive grasping with the control of myoelectric pattern recognition approach. *Micromachines*, 13:219, 1 2022b. ISSN 2072-666X. doi: 10.3390/MI13020219. URL https://www.mdpi.com/2072-666X/13/2/219/htm.

Kevin Warwick, Mark Gasson, Benjamin Hutt, Iain Goodhew, Peter Kyberd, Brian Andrews, Peter Teddy, and Amjad Shad. The application of implant technology for cybernetic systems. *Archives of Neurology*, 60:1369–1373, 10 2003. ISSN 0003-9942. doi: 10.1001/ARCHNEUR.60.10.1369. URL https://jamanetwork.com/journals/jamaneurology/fullarticle/784743.

Kevin Warwick, Dimitris Xydas, Slawomir J. Nasuto, Victor M. Becerra, Mark W. Hammond, Julia H. Downes, Simon Marshall, and Benjamin J. Whalley. Controlling a mobile robot with a biological brain. *Defence Science Journal*, 60:5–14, 3 2010. ISSN 0976-464X. doi: 10.14429/DSJ.60.11. URL https://publications.drdo.gov.in/ojs/index.php/dsj/article/view/11.

Tomasz Wasilewski, Jacek Gębicki, and Wojciech Kamysz. Bio-inspired approaches for explosives detection. *TrAC Trends in Analytical Chemistry*, 142:116330, 9 2021. ISSN 0165-9936. doi: 10.1016/J.TRAC.2021.116330.

Victoria A. Webster-Wood, Ozan Akkus, Umut A. Gurkan, Hillel J. Chiel, and Roger D. Quinn. Organismal engineering: Toward a robotic taxonomic key for devices using organic materials. *Science Robotics*, 2, 11 2017. ISSN 24709476. doi: 10.1126/SCIROBOTICS.AAP9281. URL https://www.science.org/doi/abs/10.1126/scirobotics.aap9281.

Uwe Windhorst. *Extracellular Recording*, pages 1514–1517. Springer, Berlin, Heidelberg, 11 2009. doi: 10.1007/978-3-540-29678-2_3236. URL https://link.springer.com/referenceworkentry/10.1007/978-3-540-29678-2_3236.

Kensall D. Wise and James B. Angell. A low-capacitance multielectrode probe for use in extracellular neurophysiology. *IEEE Transactions on Biomedical Engineering*, BME-22:212–219, 1975. ISSN 15582531. doi: 10.1109/TBME.1975.324562.

Gaowei Xu, Tianhe Ren, Yu Chen, and Wenliang Che. A one-dimensional CNN-LSTM model for epileptic seizure recognition using EEG signal analysis. *Frontiers in Neuroscience*, 14:1253, 12 2020. ISSN 1662453X. doi: 10.3389/fnins.2020.578126.

Karim Yaghmour. *The Xenomai Real-Time System*, page 391. O'Reilly, 1 edition, 2003. ISBN 0596550480. URL https://books.google.es/books/about/Building_Embedded_Linux_Systems.html?id=xnFdWfJAK9wC&redir_esc=y.

Y. Yarom. Rhythmogenesis in a hybrid system—interconnecting an olivary neuron to an analog network of coupled oscillators. *Neuroscience*, 44:263–275, 1 1991. ISSN 0306-4522. doi: 10.1016/0306-4522(91)90053-Q.

Oncay Yasa, Pelin Erkoc, Yunus Alapan, and Metin Sitti. Microalga-powered microswimmers toward active cargo delivery. *Advanced Materials*, 30:1804130, 11 2018. ISSN 1521-4095. doi: 10.1002/ADMA.201804130.

Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168, 2 2019. ISSN 1742-6596. doi: 10.1088/1742-6596/1168/2/022022. URL https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022.

Victor Yodaiken. The RTLinux Manifesto. 1999. URL http://www.yodaiken.com/papers/rtlmanifesto.pdf.

Guoyin Zhang, Luyuan Chen, and Aihong Yao. Study and comparison of the RTHAL-based and ADEOS-based RTAI real-time solutions for Linux. pages 771–775. IEEE, 6 2006. doi: 10.1109/IMSCCS.2006.272. URL http://ieeexplore.ieee.org/document/4673801/.

Özal Yıldırım, Paweł Pławiak, Ru San Tan, and U. Rajendra Acharya. Arrhythmia detection using deep convolutional neural network with long duration ECG signals. *Computers in Biology and Medicine*, 102:411–420, 11 2018. ISSN 0010-4825. doi: 10.1016/J.COMPBIOMED.2018.09.009.

A Repositories

All the tools and software developed as part of this thesis are accessible as open-source repositories. These are the links to the repositories related to each project.

A.1 RTHybrid

- **RTHybrid:**
  www.github.com/GNB-UAM/RTHybrid
- **RTHybrid plugins for RTXI:**
  https://github.com/GNB-UAM/rthybrid-for-rtxi
- **Neuron model generator:**
  https://github.com/GNB-UAM/rthybrid-neuron-model-generator
- **Latency test:**
  www.github.com/RoyVII/Latency_tests.git

A.2 FLC-Hybrot

- **FLC-Hybrot controlling software:**
  https://github.com/GNB-UAM/FLC-Hybrot

A.3 CNN-ripple

- **CNN-ripple Jupyter notebook:**
  https://github.com/PridaLab/cnn-ripple
- **CNN-ripple executable example:**
  https://colab.research.google.com/github/PridaLab/cnn-ripple/blob/main/src/notebooks/cnn-example.ipynb

- **CNN-ripple plugin for Open Ephys GUI:**
  https://github.com/PridaLab/CNNRippleDetectorOEPlugin
- **CNN-ripple stand-alone for Raspberry Pi:**
  https://github.com/RoyVII/CNN_RippleDetector_RaspberryPi
- **CNN-ripple Matlab API**
  https://github.com/PridaLab/cnn-matlab

B

RTHybrid neuron and synapse models

Equations for every neuron and synapse model included in RTHybrid library at the moment of writing this thesis. Parameter values for specific behaviours are described in their corresponding papers.

B.1 Neuron models

B.1.1 [Izhikevich, 2003]

$$\frac{dv}{dt} = 0.004v^2 + 5v + 140 - u + I_{ext} - I_{syn} \tag{B.1}$$

$$\frac{du}{dt} = a(bv - u) \tag{B.2}$$

$$\text{if } v \geq 30 \text{ then } \begin{cases} v = c \\ u = u + d \end{cases}$$

B.1.2 [Hindmarsh and Rose, 1984]

$$\frac{dx}{dt} = y - ax^3 + bx^2 - z + I_{ext} - I_{syn} \tag{B.3}$$

$$\frac{dy}{dt} = c - dx^2 - y \tag{B.4}$$

$$\frac{dz}{dt} = r[s(x - x_R) - z] \tag{B.5}$$

B.1.3 [Rulkov, 2002]

$$x_{n+1} = \frac{\alpha}{1 + x_n^2} + y_n - I_{syn} \tag{B.6}$$

$$y_{n+1} = y_n - \mu(x_n - \sigma) \tag{B.7}$$

B.1.4 [Ghigliazza and Holmes, 2004]

$$\frac{dv}{dt} = \frac{-(I_{Ca} + I_K + I_{KS} + I_l + I_{ext} - I_{syn})}{C_m} \tag{B.8}$$

$$\frac{dm}{dt} = \frac{\epsilon}{sech(k_K(v - v_{th\_K})} \cdot (\frac{1.0}{1 + e^{-k_K(v - v_{th\_K})}} - m) \tag{B.9}$$

$$\frac{dc}{dt} = \frac{\delta}{sech(k_{KS}(v - v_{th\_KS})} \cdot (\frac{1.0}{1 + e^{-k_{KS}(v - v_{th\_KS})}} - c) \tag{B.10}$$

$$I_{Ca} = \frac{g_{Ca}(v - E_{Ca})}{1 + e^{-k_{Ca}(v - v_{th\_Ca})}} \tag{B.11}$$

$$I_K = g_K \cdot m \cdot (v - E_K) \tag{B.12}$$

$$I_{KS} = g_{KS} \cdot c \cdot (v - E_K) \tag{B.13}$$

$$I_l = g_l(v - E_l) \tag{B.14}$$

B.1.5 [Wang, 1993]

$$\frac{dv}{dt} = \frac{-(I_{NaP} + I_{Na} + I_K + I_{KS} + I_l + I_{ext} - I_{syn})}{C_m} \tag{B.15}$$

$$w_\infty(x, y, z) = \frac{1.0}{1.0 + exp(\frac{-x-y}{z})} \tag{B.16}$$

$$I_{NaP} = g_{NaP} \cdot w_\infty(v, 51.0, 5.0) \cdot (v - v_{Na}) \tag{B.17}$$

$$I_{Na} = g_{Na} \cdot m_{Na\infty}^3 \cdot h \cdot (v - v_{Na}) \tag{B.18}$$

$$m_{Na\infty} = \frac{\alpha_m}{\alpha_m + \beta_m} \quad \alpha_m = \frac{-0.1 * (v - 30 - \sigma)}{exp(-0.1 * (v - 30 - \sigma)) - 1} \quad \beta_m = 4 \cdot exp(\frac{-v - 55 + \sigma}{18}) \tag{B.19}$$

$$\frac{dh_{Na}}{dt} = \Phi(\alpha_h \cdot (1 - h_{Na})) - (\beta_h \cdot h_{Na})$$

$$\alpha_h = 0.07 \cdot exp(\frac{-(v+44-\sigma)}{20}) \qquad \beta_h = \frac{1}{exp(-0.1 \cdot (v+14-\sigma)) + 1} \tag{B.20}$$

$$I_K = g_K \cdot n_K^4 \cdot (v - v_K) \tag{B.21}$$

$$\frac{dn_K}{dt} = \Phi(\alpha_n \cdot (1 - n_K)) - (\beta_n \cdot n_K)$$

$$\alpha_n = \frac{-0.01(v+34-\sigma)}{exp(-0.1(v+34-\sigma))} \qquad \beta_n = 0.125 \cdot exp(\frac{-(v+44-\sigma)}{80}) \tag{B.22}$$

$$I_{KS} = g_{KS} \cdot m_{KS} \cdot (\rho \cdot h_1 + (1 - \rho \cdot h_2)) \cdot (v - v_K) \tag{B.23}$$

$$\frac{dm_{KS}}{dt} = \frac{\Phi(w_\infty(v, 34.0, 6.5) - m_{KS})}{\tau_m} \qquad \tau_m = 6ms \tag{B.24}$$

$$\frac{dh_1}{dt} = \frac{\Phi(w_\infty(v, -65.0, 6.6) - h_1)}{\tau_{h1}} \qquad \tau_{h1} = 200 + \frac{220}{1 + exp(\frac{-(v+71.6)}{6.85})} \tag{B.25}$$

$$\frac{dh_2}{dt} = \frac{\Phi(w_\infty(v, -65.0, 6.6) - h_2)}{\tau_{h2}} \qquad \tau_{h2} = 200 + \frac{3200}{1 + exp(\frac{-(v+63.6)}{4})} \tag{B.26}$$

$$I_l = g_l(v - v_l) \tag{B.27}$$

B.1.6 [Komendantov and Kononenko, 1996]

$$\frac{dv}{dt} = \frac{-(I_{Na} + I_K + I_B + I_{Na(TTX)} + I_{K(TEA)} + I_{Na}(v) + I_{Ca} + I_{Ca-Ca} + I_{ext} - I_{syn})}{C_m} \tag{B.28}$$

$$I_{Na}(v) = g_{Na}(v)\frac{1}{(1 + exp(-0.2(v+45)))}(v - v_{Na}) \tag{B.29}$$

$$I_K = g_K(v - v_K) \tag{B.30}$$

$$I_{Na} = g_{Na}(v - v_{Na}) \tag{B.31}$$

$$I_B = g_B \cdot m_B \cdot h_B \cdot (v - v_B)$$

$$\frac{dm_B}{dt} = \frac{\frac{1}{1+exp(0.4(v+34))} - m_B}{0.05} \qquad \frac{dh_B}{dt} = \frac{\frac{1}{1+exp(-0.55(v+43))} - h_B}{1.5} \tag{B.32}$$

$$I_{Na(TTX)} = g_{Na(TTX)} \cdot m^3 \cdot h \cdot (v - v_{Na})$$

$$\frac{dm_{Na}}{dt} = \frac{\frac{1}{1+exp(-0.4(v+31))} - m_{Na}}{0.0005} \qquad \frac{dh_{Na}}{dt} = \frac{\frac{1}{1+exp(0.25(v+45))} - h_{Na}}{0.01} \tag{B.33}$$

$$I_{K(TEA)} = g_{K(TEA)} \cdot n^4 \cdot (v - v_K) \qquad \frac{dn_K}{dt} = \frac{\frac{1}{1+exp(-0.18(v+25))} - n_K}{0.015} \tag{B.34}$$

$$I_{Ca} = g_{Ca} \cdot m_{Ca}^2 \cdot (v - v_{Ca}) \qquad \frac{dm_{Ca}}{dt} = \frac{\frac{1}{1+exp(-0.2v)} - m_{Ca}}{0.01} \tag{B.35}$$

$$I_{Ca-Ca} = g_{Ca-Ca} \frac{1}{1 + exp(-0.06 \cdot (v + 45))} \cdot \frac{1}{1 + exp(k_\beta \cdot ([Ca] - \beta))} \cdot (v - v_{Ca})$$

$$\frac{d[Ca]}{dt} = \rho\{\frac{-I_{Ca}}{2F \cdot (\frac{4\pi R^3}{3})} - k_s[Ca]\} \tag{B.36}$$

B.1.7 [Nowotny et al., 2008]

$$I_{VV} = g_{VV}(v_{soma} - v_{axon}) \tag{B.37}$$

$$\frac{dv_{axon}}{dt} = \frac{-(I_{Na} + I_{Kd} + I_M + I_{leak,a} - I_{VV})}{C_a} \tag{B.38}$$

$$\frac{dv_{soma}}{dt} = \frac{-(I_{Ca} + I_{KCa} + I_A + I_h + I_{leak,s} + I_{VV} - I_{scale} * (I_{ext} - I_{syn} + I_{offset}))}{C_s} \tag{B.39}$$

All ionic currents, unless explicitly stated differently below, are modeled by an equation of the form

$$I_x = g_x m^p h^q (V - V_x) \tag{B.40}$$

where activation and inactivation variables are governed by

$$\frac{dm}{dt} = a_m(1 - m) - b_m m \tag{B.41}$$

$$\frac{dh}{dt} = a_h(1 - h) - b_h h \tag{B.42}$$

for I_{Na} and I_{Kd}, and

$$\frac{dm_x}{dt} = (m_{\infty x}(v) - m_x)k_{mx} \tag{B.43}$$

$$\frac{dh_x}{dt} = (h_{\infty x}(v) - h_x)k_{hx} \tag{B.44}$$

for currents CaT, CaS, KCa, A, h and M. Their activation and inactivation functions are defined by

$$a_{mNa} = 0.32\frac{v_{axon} + 52}{1 - exp(-\frac{v_{axon}+52}{4})} \qquad b_{mNa} = 0.28\frac{v_{axon} + 25}{exp(\frac{v_{axon}+25}{5}) - 1} \tag{B.45}$$

$$a_{hNa} = 0.128exp(-\frac{v_{axon} + 48}{18}) \qquad b_{hNa} = \frac{4}{exp(-\frac{v_{axon}+25}{5}) + 1} \tag{B.46}$$

$$a_{mKd} = 0.032\frac{v_{axon} + 50}{1 - exp(-\frac{v_{axon}+50}{5})} \qquad b_{mKd} = 0.5exp(-\frac{v_{axon} + 55}{40}) \tag{B.47}$$

$$m_{\infty CaT} = \frac{1}{1 + exp(\frac{v_{soma}-v_{mCat}}{s_{mCaT}})} \qquad h_{\infty CaT} = \frac{1}{1 + exp(\frac{v_{soma}-v_{hCaT}}{s_{hCaT}})} \tag{B.48}$$

$$k_{mCaT} = k_{mCaT} \qquad k_{hCaT} = \frac{k_{hCat}}{1 + exp(\frac{v_{soma}-v_{khCaT}}{s_{khCaT}})} \tag{B.49}$$

$$m_{\infty CaS} = \frac{1}{1 + exp(\frac{v_{soma}-V_{mCaS}}{s_{mCaS}})} \qquad k_{mCaS} = k_{mCaS} \tag{B.50}$$

$$m_{\infty KCa} = \frac{[Ca]}{c_{mKCa} + [Ca]} \cdot \frac{1}{1 + exp(\frac{v_{soma}-(v_{mKCa1}-f[Ca])}{s_{mKCa1}})} \cdot \frac{1}{1 + exp(\frac{v_{soma}-(v_{mKCa2}-f[Ca])}{s_{mKCa2}})}$$

$$h_{\infty KCa} = \frac{c_{hKCa1}}{c_{hKCa2} + [Ca]} \tag{B.51}$$

$$k_{mKCa} = k_{mKCa} \qquad k_{hKCa} = k_{hKCa} \tag{B.52}$$

$$m_{\infty A} = \frac{1}{1 + exp(\frac{v_{soma}-v_{mA}}{s_{mA}})} \qquad h_{\infty A1} = h_{\infty A2} = \frac{1}{1 + exp(\frac{v_{soma}-v_{hA}}{s_{hA}})} \tag{B.53}$$

$$k_{mA} = k_{mA} \qquad h_{A1} : k_{hA1} : k_{hA2} = \frac{k_{hA2}}{1 + exp(\frac{v_{soma}-v_{khA2}}{s_{khA2}})} \tag{B.54}$$

$$m_{\infty h} = \frac{1}{1 + exp(\frac{v_{soma} - v_{mh}}{s_{mh}})} \qquad k_{mh} = k_{mh} \cdot (1 + exp(\frac{v_{soma} - v_{kmh}}{s_{kmh}})) \tag{B.55}$$

$$m_{\infty M} = \frac{1}{1 + exp(\frac{v_{axon} - v_{mM}}{s_{mM}})} \qquad k_{mM} = \frac{k_{mh}}{1 + exp(\frac{v_{axon} - v_{kmM}}{s_{kmM}})} \tag{B.56}$$

$$I_{Ca} = (g_{CaT} m_{CaT} h_{CaT} + g_{CaS} m_{CaS}) \frac{[Ca] exp(\frac{v_{soma}}{RT/F}) - [Ca]_{out}}{exp(\frac{v_{soma}}{RT/F}) - 1} P_{Ca} v_{soma} \tag{B.57}$$

$$I_A = g_A m_A^3 (a h_{A1} + (1 - a) h_{A2})(v_{soma} - v_A) \qquad a = \frac{1}{1 + exp(\frac{v_{soma} - v_{aA}}{s_{aA}})} \tag{B.58}$$

$$I_{leak,s} = g_{leak,s}(v_{soma} - v_{leak}) \qquad I_{leak,a} = g_{leak,a}(v_{axon} - v_{leak}) \tag{B.59}$$

$$\frac{d[Ca]}{dt} = -c_{ICa} I_{Ca} - k_{Ca}([Ca] - [Ca]_0) \tag{B.60}$$

B.2 Synapse models

B.2.1 Electrical

$$I_{syn} = g \cdot (V_{post} - V_{pre}) \tag{B.61}$$

B.2.2 [Golowasch et al., 1999]

$$I_{syn} = I_{fast} + I_{slow} \tag{B.62}$$

$$I_{fast} = \frac{g_{fast} \cdot (V_s^{post} - E_{syn})}{1 + exp(s_{fast}(V_{fast} - V_s^{pre}))} \tag{B.63}$$

$$I_{slow} = g_{slow} \cdot m_{slow} \cdot (V_s^{post} - E_{syn}) \tag{B.64}$$

$$\frac{dm_{slow}}{dt} = \frac{k_1(1 - m_{slow})}{1 + exp(s_{slow}(V_{slow} - V_s^{pre}))} - k_2 m_{slow} \tag{B.65}$$

B.2.3 [Destexhe et al., 1994]

$$I_{syn} = g \cdot r \cdot (V_{post} - E_{syn}) \tag{B.66}$$

$$\frac{dr}{dt} = (\alpha \cdot [T] \cdot (1 - r)) - (\beta \cdot r) \tag{B.67}$$

where [T] is the concentration of synaptic transmitter. During transmitter release [T] = T_{max}, with [T] = 0 otherwise.

B.2.4 [Greenberg and Manor, 2005]

$$I_{syn} = g \cdot m^p \cdot h^q \cdot (V_{post} - E_{syn}) \tag{B.68}$$

$$\frac{dx}{dt} = \frac{x_\infty(V) - x}{\tau_x(V)} \qquad x = m, h \tag{B.69}$$

$$x_\infty(V) = \frac{1}{1 + exp(-\frac{(V - V_{1/2,x})}{k_x})} \tag{B.70}$$

$$\tau_x(V) = \tau_{x,lo} + \frac{\tau_{x,hi} - \tau_{x,lo}}{1 + exp(\frac{(V - V_{1/2,x})}{k_x})} \tag{B.71}$$

C

Publications

In this appendix, we present the list of publications that have been accomplished throughout the thesis in indexed JCR journals and international congresses, and relate them to the corresponding chapters.

C.1 Publications in indexed JCR journals

- **Andrea Navas-Olive*, Rodrigo Amaducci*, Maria-Teresa Jurado-Parras, Enrique R. Sebastian, Liset M. de la Prida. 2022. Deep learning based feature extraction for prediction and interpretation of sharp-wave ripples, *eLife* 11:e77772.**

  JIF: 8.713 (BIOLOGY, 2021)

  QUARTILE: Q1 (8/94)

  EDITORIAL: ELIFE SCIENCES PUBLICATIONS LTD

  ISSN: 2050-084X

  DOI: 10.7554/eLife.77772

  Related with **chapter** 4.

  *These authors contributed equally to this work

- **Rodrigo Amaducci, Manuel Reyes-Sanchez, Irene Elices, Francisco B. Rodriguez, Pablo Varona. 2019. RTHybrid: a standardized and open-source real-time software model library for experimental neuroscience, *Frontiers in Neuroinformatics* 13:11.**

  JIF: 2.649 (MATHEMATICAL AND COMPUTATIONAL BIOLOGY, 2019)

  QUARTILE: Q1 (13/59)

  EDITORIAL: FRONTIERS MEDIA SA

  ISSN: 1662-5196

  DOI: 10.3389/fninf.2019.00011

Related with **chapter 2**.

- **Manuel Reyes-Sanchez, Rodrigo Amaducci, Irene Elices, Francisco B. Rodriguez, Pablo Varona. 2020. Automatic adaptation of model neurons and connections to build hybrid circuits with living networks, *Neuroinformatics* 18: 377–393.**

  JIF: 4.085 (COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS, 2020)

  QUARTILE: Q2 (38/111)

  EDITORIAL: HUMANA PRESS INC

  ISSN: 1539-2791

  DOI: 10.1007/s12021-019-09440-z

  Related with **chapter 2**.

C.2 Articles under revision

- **Manuel Reyes-Sanchez, Rodrigo Amaducci, Pablo Sanchez-Martin, Irene Elices, Francisco B. Rodriguez, Pablo Varona. 2022. Automatized offline and online exploration to achieve a target dynamics in hybrid neural circuits built with living and model neurons, *Neural Networks.***

  JIF: 9.657 (COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE, 2021)

  QUARTILE: Q1 (16/145)

  EDITORIAL: PERGAMON-ELSEVIER SCIENCE LTD

  ISSN: 0893-6080

  Related with **chapter 2**.

C.3 Articles in preparation

- **Rodrigo Amaducci, Irene Elices, Pablo Sanchez-Martin, Manuel Reyes-Sanchez, Alicia Garrido-Peña, Carlos Garcia-Saura, Rafael Levi, Francisco B. Rodriguez, Pablo Varona. 2022. Robotic motion controlled by the flexible rhythm of a living CPG with sensory feedback from the robot.**

  Related with **chapter 3**.

C.4 Contributions to international conferences

- **Pablo Soëtard, Rodrigo Amaducci, Pablo Martin-Sanchez, Manuel Reyes-Sanchez, Alicia Garrido-Peña, Rafael Levi, Francisco B. Rodriguez, Pablo Varona. 2022. Dynamical principles of functional neural sequences validated in hybrid robots built with living central pattern generators. *CNS 2022 Melbourne, Australia.*** In press for Journal of Computational Neuroscience.

  Related with **chapter 3**.

- **Andrea Navas-Olive, Rodrigo Amaducci, Maria-Teresa Jurado-Parras, Enrique R. Sebastian, Liset M. de la Prida. 2022. Using 1D-convolutional neural networks to detect and interpret sharp-wave ripples.** ***COSYNE 2022 Lisbon, Portugal.***

  Related with **chapter 4**.

- **Manuel Reyes-Sanchez, Rodrigo Amaducci, Francisco B. Rodriguez, Pablo Varona. 2021. Parameter adaptation of hybrid circuits by online exploration driven by genetic algorithms.** ***CNS 2021 Online.*** Journal of Computational Neuroscience 49 (Suppl 1): S189-S190, P192

  Related with **chapter 2**.

- **Rodrigo Amaducci, Andrea Navas-Olive, Maria-Teresa Jurado-Parras, Liset M de la Prida. 2021. Using convolutional neural networks to evaluate hippocampal sharp-wave ripples in real-time.** ***ENCODS 2021 Online.*** Oral presentation

  Related with **chapter 4**.

- **Rodrigo Amaducci, Irene Elices, Manuel Reyes-Sanchez, Alicia Garrido-Peña, Rafael Levi, Francisco B. Rodriguez and Pablo Varona. 2021. Controlling robotic locomotion by a closed-loop interaction with living central pattern generators.** ***COSYNE 2021 Online.*** 2-030

  Related with **chapter 3**.

- **Rodrigo Amaducci, Irene Elices, Manuel Reyes-Sanchez, Alicia Garrido-Peña, Rafael Levi, Francisco B. Rodriguez, Pablo Varona. 2020. Hybrid robot driven by a closed-loop interaction with a living central pattern generator with online feedback.** ***CNS 2020 Online.*** BMC Neuroscience 2020, 21(Suppl 1): P207

  Related with **chapter 3**.

- **Manuel Reyes-Sanchez, Irene Elices, Rodrigo Amaducci, Francisco B. Rodriguez, Pablo Varona. 2020. Parameter exploration in neuron and synapse models driven by stimuli from living neuron recordings.** ***CNS 2020 Online.*** BMC Neuroscience 2020, 20(Suppl 1): P206

  Related with **chapter 2**.

- **Rodrigo Amaducci, Irene Elices, Manuel Reyes-Sanchez, Rafael Levi, Francisco B. Rodriguez, Pablo Varona. 2019. Robotic locomotion driven by the flexible rhythm of a living Central Pattern Generator.** ***CNS 2019 Barcelona, Spain.*** BMC Neuroscience 2019, 20(Suppl 1):P322

  Related with **chapter 3**.

- **Manuel Reyes-Sanchez, Rodrigo Amaducci, Irene Elices, Francisco B. Rodriguez, Pablo Varona. 2019. Transfer of rich living circuit dynamics to neuron models through graded synapses.** ***CNS 2019 Barcelona, Spain.*** BMC Neuroscience 2019, 20(Suppl 1): P323

  Related with **chapter 2**.

- **Irene Elices, Manuel Reyes-Sanchez, Rodrigo Amaducci, Rafael Levi, Francisco B. Rodriguez, Pablo Varona. 2019. Hybrid circuits to assess sequential neural rhythms from low dimensional observations.** ***CNS 2019 Barcelona, Spain.*** BMC Neuroscience 2019, 20(Suppl 1): P321

  Related with **chapter 2**.

- **Irene Elices, Manuel Reyes-Sanchez, Rodrigo Amaducci, Rafael Levi, Francisco B. Rodriguez, Pablo Varona. 2018. Dynamical invariants underlying robustness and flexibility in sequential neural dynamics.** ***SFN Neuroscience 2018 San Diego, United States.***

  Related with **chapter 2**.

- **Rodrigo Amaducci, Manuel Reyes-Sanchez, Irene Elices, Francisco B. Rodriguez, Pablo Varona. 2018. A cross-platform real-time model library to build hybrid neural circuits.** ***CNS 2018 Seattle, United States.*** BMC Neuroscience 2018, 19(Suppl 2):P172

  Related with **chapter 2**.

- **Irene Elices, Manuel Reyes-Sanchez, Rodrigo Amaducci, Rafael Levi, Francisco B. Rodriguez, Pablo Varona. 2018. Unveiling and characterizing dynamical invariants in central pattern generators.** ***CNS 2018 Seattle, United States.*** BMC Neuroscience 2018, 19(Suppl 2): P173

  Related with **chapter 2**.

- **Manuel Reyes-Sanchez, Irene Elices, Rodrigo Amaducci, Carlos Muniz, Francisco B. Rodriguez, Pablo Varona. 2018. Assisted construction of hybrid circuits: making easy the implementation and automation of interactions between living and model neurons.** ***CNS 2018 Seattle, United States.*** BMC Neuroscience 2018, 19(Suppl 2):O19

  Related with **chapter 2**.

- **Rodrigo Amaducci, Manuel Reyes-Sanchez, Irene Elices, Francisco B. Rodriguez, Pablo Varona. 2018. Assisted construction of hybrid circuits: making easy the implementation and automation of interactions between living and model neurons.** ***FENS Forum 2018 Berlin, Germany.***

  Related with **chapter 2**.

- **Rodrigo Amaducci, Carlos Muñiz, Manuel Reyes-Sanchez, Francisco B. Rodriguez, Pablo Varona. 2018. Assisted construction of hybrid circuits: making easy the implementation and automation of interactions between living and model neurons.** ***SENC Congress 2017 Alicante, Spain.***

  Related with **chapter 2**.

- **Rodrigo Amaducci, Carlos Muñiz, Manuel Reyes-Sanchez, Francisco B. Rodriguez, Pablo Varona. 2017. On the need for standardized real-time software technology in closed-loop neuroscience.** ***CNS 2017 Antwerp, Belgium.*** BMC Neuroscience 2017, 18 (Suppl 1):P104

  Related with **chapter 2**.

- **Manuel Reyes-Sanchez, Irene Elices, Rodrigo Amaducci, Carlos Muniz, Francisco B. Rodriguez, Pablo Varona. 2017. Automatic calibration for hybrid circuits of living and artificial neurons.** ***CNS 2017 Antwerp, Belgium.*** BMC Neuroscience 2017, 18 (Suppl 1):P281

  Related with **chapter 2**.