UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

21|22

Bachelor's Thesis

Systematic methodology for biologically controlled robot design

Pablo Ernesto Soëtard García

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Degree in Computer Science and Engineering

BACHELOR'S THESIS

Systematic methodology for biologically controlled robot design

Author: Pablo Ernesto Soëtard García
Advisor: Pablo Varona Martínez

June 2022

Pablo Ernesto Soëtard García
Systematic methodology for biologically controlled robot design

Pablo Ernesto Soëtard García
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

A mi familia y amigos

Si buscas resultados diferentes no hagas siempre lo mismo.

Albert Einstein

Agradecimientos

En primer lugar me gustaría agradecer tanto a la Universidad Autónoma de Madrid como al Ministerio de Educación por el apoyo económico percibido gracias a la beca de colaboración otorgada.

Me gustaría agradecer el apoyo, consejos y ayuda indispensable brindada por todos los miembros del Grupo de Neurocomputación Biológica de la Universidad Autónoma de Madrid, en especial a mi tutor Pablo Varona.

Y por último a mis padres, que me han apoyado y animado durante el transcurso de toda la investigación, aportando nuevos puntos de vista y valiosos consejos.

Resumen

El poder computacional de las neuronas es bien conocido desde hace décadas, es la base de la Neurociencia y puede tener impacto en otros campos como el de la Inteligencia Artificial. Este poder computacional también puede ser usado para crear controladores de sistemas robóticos bio-inspirados, este es un campo activo de estudio en ramas de la biomedicina para producir prótesis seguras y ergonómicas.

El propósito de este trabajo es elaborar un método consistente para la creación de controladores biológicos para sistemas digitales y actuadores mecánicos (es decir, un robot híbrido o Hybrot), en concreto robots bípedos humanoides que potencialmente podrían ser usados como prótesis para pacientes con lesiones de médula espinal.

Este sistema está basado en los circuitos neuronales vivos encargados de las tareas de locomoción llamados Generadores Centrales de Patrones (GCPs). Estudios recientes han demostrado relaciones robustas entre algunos de los intervalos que forman las secuencias neuronales que controlan la actividad motora rítmica. Estas relaciones forman invariantes dinámicos que pueden ser aplicados para implementar y coordinar secuencias de movimientos robóticos robustas y autónomas, adaptando su comportamiento al entorno mientras la secuencia locomotora se mantiene en marcha.

En este trabajo se ha demostrado que los ritmos generados por los GCPs son adecuados para el control de sistemas robóticos, esto es debido a su naturaleza secuencial y periódica. Además, el uso de invariantes dinámicos frente a otras relaciones de intervalos presentes en los GCPs ha demostrado ser favorable para el control de la locomoción de robots. Esto ha sido probado gracias a interacciones monodireccionales online y offline así como experimentos con Hybrots bidireccionales en un entorno online.

Palabras clave

Hybrots, Robótica, Neurocomputación, Ingeniería Biomédica, Prótesis, Generadores Centrales de Patrones, Invariantes Dinámicos

Abstract

The computational power of neurons has been known for several decades now, and is the ground base for Neuroscience and can impact other fields like Artificial Intelligence. This computational power can also be exploited to create controllers for bio-inspired robotic systems, which is an active field of study in branches of biomedicine to produce ergonomic and reliable prosthesis.

The purpose of this work is to elaborate a reliable method to create biological-based controllers for digital systems and mechanical actuators (i.e. a hybrid robot or Hybrot), in this case focused in humanoid biped robots that could potentially be used as prosthesis for patients with spine injuries.

This system is based upon the living neural circuits in charge of locomotion tasks called Central Pattern Generators (CPGs), those are present in all living beings. Recent studies have shown different constraints between the intervals that build robust neural sequences named **dynamical invariants** that could be applied to perform robust and autonomous robot movement sequences, adapting its behavior to the environment while keeping the locomotor sequence going.

In this work the use of CPG sequential periodic rhythms for the control of robotics systems has been demonstrated. Furthermore, the advantages of using dynamical invariants against other interval relationships present in CPG rhythms have been proven effective for robotic locomotion control both in mono-directional offline and online interactions, and bidirectional online Hybrot experiments.

Keywords

Hybrots, Robotics, Neurocomputing, Biomedicine, Prosthesis, Central Pattern Generators, Dynamic Invariants

Table of Contents

Lists

List of algorithms

List of codes

List of equations

List of figures

List of tables

1

Introduction

Neurons are the cells in charge of endowing the brain with intelligence, and this due to their complex computational capabilities. Since the discovery of the neuron doctrine that considered that the nervous system is made up of discrete individual cells by scientist Santiago Ramón y Cajal in 1888 [1], multitude of studies have been carried out to understand neural computation. At first the goal was to understand their properties and capabilities and later on they inspired modern computing architectures [2] and their computational capabilities have been exploited to create bio-inspired system controllers.

In biomedicine several attempts have been made to create reliable Brain-Machine Interfaces (BMI) to control digital systems and mechanical actuators [3, 4]. By analyzing the neural activity of a living being and using neural events as triggers to perform different behavioral functions, novel neurorehabiliation protocols have been built [5].

It is known that rhythmic locomotion is generally orchestrated by specific neural circuits denominated Central Pattern Generators, those circuits are in charge of autonomously generating robust rhythms without the need external stimuli [6]. Invertebrate CPGs are the best known circuits in neuroscience research since their neurons and connections can be mapped and their dynamics has been well characterized using electrophysiological experiments [7].

This work will cover the identification and application of dynamical invariants in the form of robust relationships between time intervals that cycle-by-cycle build the functional sequence of CPGs and its bidirectional interaction with robotic sensors and mechanical actuators, to design hybrid electronic-biological entities called Hybrots. The CPG used for this work will be the one found on the gastric system of the *Carcinus maenas*, this CPG generates a triphasic rhythm that will be exploited to control a biped robot named NeuroBip that has a triphasic walking locomotor sequence. The robust rhythm capabilities of CPGs combined with their temporal structures will be used to adapt the Hybrot's behavior to the environment while keeping the locomotor sequence going.

1.1 State of the Art

The term Hybrot comes from the conjunction of the words *hybrid* and *robot*. It is used to define robots that are controlled by a computational framework based on both, electronic and biological elements. These biological elements tend to be neurons, that are the only living cells that are know to provide computational-like power. The interest on the study of Hybrots started in the early 2000s, when Potter and colleagues coined this concept [8, 9], and first used the computational properties of living neural networks for robotic control.

Hybrots should not be confused with other bioengineered Biohybrid robots, being the later biological robots made up generally of biological and artificial materials, used to obtain improved performance or features that are difficult to mimic with man-made materials [10]. They are usually built with muscular tissue, either cardiac or skeletal, and an artificial skeleton. These robots have extensive applications in the biomedical field as efficient and targeted drug delivery systems [11].

The computational power of neural meshes grown in *in vitro* cultures has been evaluated in different works such as [12] and its application in robotic systems is well known among neuroscientists and biomedical engineers. On the other hand, the use of neural circuits present in living beings as robotic controllers has been studied to build BMIs. As an example, neural circuits which generate oscillatory patterns for locomotor activities, such as CPGs , which present a rich dynamical yet stable behavior, have been extensively simulated using a variety of techniques, namely the Kuramoto coupled oscillators model [13] or neural networks [14, 15], and have been used as robot controllers [16].

Researchers have recently developed real-time algorithmic digital protocols to interact with living neurons [17, 18] and have found linear dynamical invariants on CPGs [19]. This elucidates potential hybridization of brain and robotic technologies with applications in neurorehabilitation treatments and the design of biomedical systems such as exoskeletons or intelligent biomechanic prosthesis.

1.2 Objectives

To successfully develop the system proposed on this work, a list of objectives have been designed. These objectives aim to ensure that the system is reliable and reproduces the behaviour of human capabilities as close as possible.

1.2.1 Robot controllers based on Dynamical Invariants present in CPGs

In the past most biological based controllers have only taken advantage of neural behavior in a *event-trigger* like methodology, that means only taking into account neural events and performing a defined task when detected. Most of those methodologies have not taken into account the sequence and time

interactions between neurons that can be exploited to achieve powerful robust adaptive rhythms.

Recently new constraints in shaping intervals that build the Central Pattern Generator (CPG) sequence were discovered [19, 20], those linear correlations overtime denoted as dynamical invariants have been shown to be linearly dependent on the CPG rhythm through time. This can be used to adapt and coordinate the CPG rhythm as desired, giving the opportunity to create a robust rhythm that can be adapted to external conditions, in our case, positional information feedback given by sensors.

In the experiments from [19], at least two linear dynamical invariants were detected in the *Carcinus maenas'* gastric CPG. As already mentioned, those can be exploited for loco-motor controlling purposes, as they maintain a linear correlation thought time, on top of the constant rhythm generated by the CPG, this can lead to more robust biological controllers than the ones based on the *event-trigger* methodology.

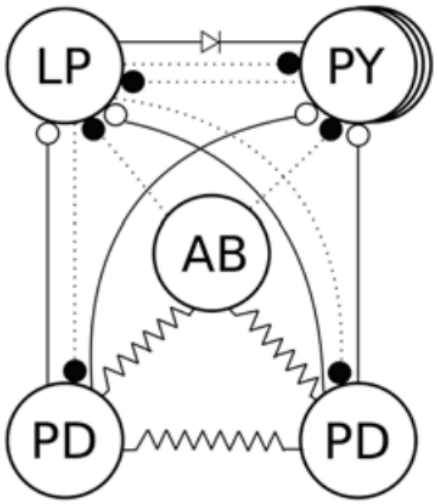

Figure 1.1: Carcinus maenas' gastric CPG circuit

As it can be seen in figure 1.1, the *Carcinus maenas'* CPG rhythm can be considered to be controlled by mainly four types of neurons *(**Lateral Pyloric (LP) , Pyloric (PY) , Pyloric Dilator (PD)** and AB)*. There are electrical connections between the *AB* and the two *PD* neurons (represented with a resistor symbol), that means that those connections follow *Ohm's law*. On the other hand, there is a monodirectional connection between the *LP* and the *PYs*, denoted by a diode symbol. All other connections are chemical inhibitory synapses, those can be described by several complex Ordinary Differential Equation (ODE) , the ones dotted are indirect connections, and the solid ones are direct connections. When monitoring the CPG with *in vitro* electrodes only three types of neurons are accessible at the GNB-UAM electrophysiology setup, those are *LP, PYs and the two PDs*. The dynamical invariants have been discovered in experiments monitoring those type of neurons and the different burst intervals between them [19].

Figure 1.2 represents the different intervals between the CPG neurons in one of its periodic cycles.

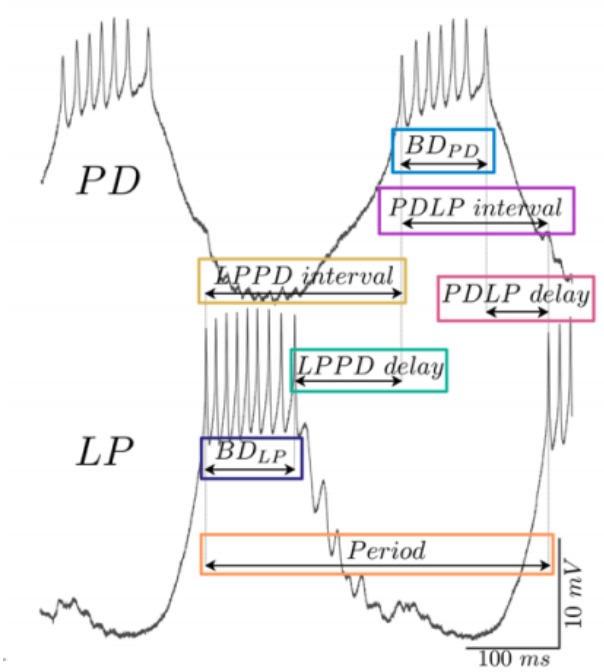

Figure 1.2: Carcinus maenas' gastric CPG neuron periods of one cycle [19].

Interval correlation	$\rho_{<Exp>}$	t-test
LPPD inter.[Period]	1.000	1
LPPD delay [Period]	0.999	1

Table 1.1: Correlations between Carcinus maenas' gastric CPG neuron burst intervals that define the two dynamical invariants [19].

Following that notation, the two dynamical invariants found are represented in table 1.1. This work targets to demonstrate the advantages of using dynamical invariants found on CPGs to achieve robust locomotor tasks, rather than using *event-trigger* like methodologies.

1.2.2 Dynamical Ivariants vs other Periods

In the study [19], correlations between all available sequence intervals of the *Carcinus maenas'* CPG were evaluated, the ones with the highest correlation were those of table 1.1, the rest have intermediate or low correlation, which made them unsuitable for locomotion activities. The intervals with the least correlation found were the ones of table 1.2.

Interval correlation	$\rho_{<Exp>}$	**t-test**
PDLP delay[BD_{LP}]	**0.764**	**0**

Table 1.2: Carcinus maenas' gastric CPG neuron burst period with the least correlation. From [19].

A series of experiments and simulations will be made to demonstrate the efficiency of dynamical invariants creating robust rhythms against other sequence interval relations such as the ones from table 1.2.

1.2.3 Reaction Time

A general milestone that could enhance the similarity with human behavior would be that the system should be capable of reacting to external stimuli in a human-like time, this time is around 250 ms Round Trip Time (RTT) or 125 ms One Way Delay (OWD) at most.

Walking sequences depend upon equilibrium feedback on most living beings, the reaction mechanisms and information flow can be seen in figure 1.3. For humans, the vestibular system located in the inner ear is the one responsible for giving spacial position feedback to the brain, the brain then processes that information and performs corrections in muscles if necessary.

The electronic system that will be crafted in this work will mimic those biological systems, the vestibular system will be replaced by a gyroscope sensor, then the electronic controller will be the interface between the sensors and actuators and the CPG, and last but not least, the muscles will be replaced by mechanical actuators, in our case servo motors.

1.2.4 System Overview

Taking into account all the objectives previously mentioned, the system of figure 1.4 should satisfy all the requirements.

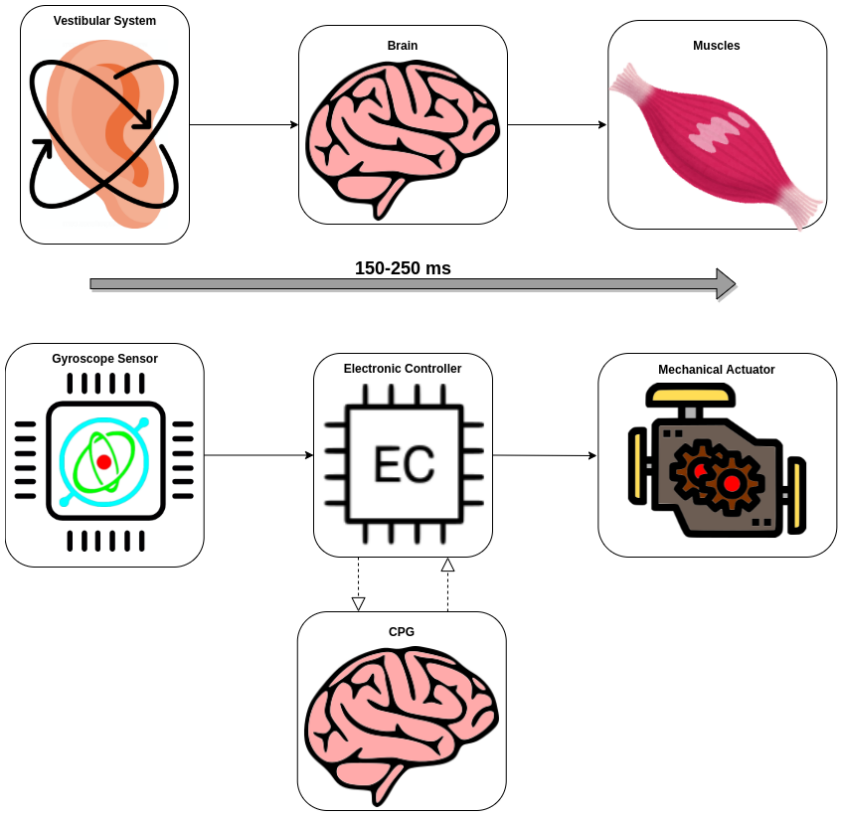

Figure 1.3: Reaction time diagram

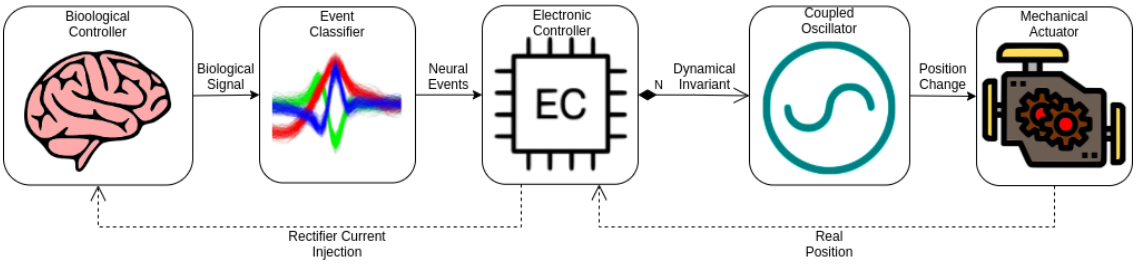

Figure 1.4: General system diagram

As showed on the diagram of figure 1.4, a general way of designing a system with a biological controller based on dynamical invariants present in biological CPGs is created by combining three main components.

Event Classifier

The event detection/classifier, uses techniques mainly known as **spike sorting**, the objective of this module is to get the required timestamps from the biological signals that are needed to compute the different time intervals and associated dynamical invariants.

Electronic Controller

The electronic controller is the component that receives the timestamps detected by the neural event classifier and calculates the dynamical invariants. It then coordinates the movements of the mechanical oscillators taking into account the calculated dynamical invariants. In addition, it sends back feedback to the CPG in order to modify its rhythm to adapt to the environment, this is done by injecting current on one of the neurons.

Coupled Oscillators and Mechanical Actuators

As mentioned in subsection 1.2.1, dynamical invariants are thought to be present in all living CPGs, this linear relationships can be used to control different coupled oscillators. If this is applied for robot locomotion purposes, those will be mechanical oscillators such as motors. In our case, the *Carcinus maenas'* gastric CPG has at least two dynamical invariants: *LP-PD, PY-PD*. These dynamical invariants will control two mechanical oscillators, one of them will be used to move the hip of a biped robot and the other one its knees.

2

Design and Development

After coming up with the desired system high-level architecture explained in section 1.2.4, the following elements have been identified as key parts of it and have been developed: a spike sorter (described in section 2.1), a physics engine based simulation for robots (described in section 2.2), and finally the hardware for the biped robot and its controller (described in section 2.3). In the following sections a detailed explanation of how they have been designed and developed will be given.

2.1 Spike Sorting

The first element that appears in the system information processing flow is a spike sorter. Spike Sorting are techniques that aim to distinguish the different neural events that are present on a biological signal. Each neuron tends to fire spikes with a particular shape, the general way of automatically differentiate them is by clustering the neural spikes by shape similarity [21]. The code developed in this section can be found at https://github.com/pabolojo/TFG/tree/master/SpikeSorter

2.1.1 Current Setup

The current experiment setup to acquire and detect the three main neurons needed to calculate the dynamical invariants on the *Carcinus maenas'* CPG requires two electrodes to be inserted into the stomatogastric ganglion. One of the electrodes is inserted in the extracellular space among neurons, the other one needs to be inserted inside one of the *PD* neurons of the CPG, the two signals coming from the ganglion are sampled at 10 KHz by a data acquisition card. After both signals have been sampled they are processed by a low-level program written in C that classifies the neural events based on different thresholds evaluated on both signals.

This method is fast and accurate, but it comes with several drawbacks. For example, it requires two electrodes to be inserted on the stomatogastric ganglion, in addition to the difficulty of finding a specific neuron, in this case the *PD*, this could be a red flag on setups were the number of electrodes available is very limited. Another downside of this technique is that the thresholds used for spike sorting need to

be fine tuned for each experiment, leading to misbehavior otherwise.

The new technique that has been designed and developed requires only one electrode to be inserted on the extracellular space, demanding less equipment and experimental time, and all the information is extracted from the extracellular activity of the CPG. Furthermore, this technique is extensible to the vast majority of experiments, not needing adjustments from one another. The main drawback of this new technique is that its accuracy is not as high and that is not as close to real-time as the threshold method, this is mainly due to the implementation being in Python, a high-level slower programming language. Those downsides could be partially solved by improving the neural sorting models and using Application-Specific Integrated Circuits (ASICs) for inference, potentially producing a significant improvement in classification accuracy and inference time.

2.1.2 Data acquisition and preprocessing

The data acquisition is performed the same way as in the previous setup, but in this case, as Python was used to implement the spike sorting algorithm, a wrapper should be created. This Python wrapper calls the system C functions that are included in the kernel after installing the drivers for the data acquisition card that records the signals from the living CPG circuit. With this Pyhton wrapper when a *DataAcquisitor* object is created it automatically calls the needed kernel functions to successfully initialize and configure the data acquisition card, this object holds all the necessary information to maintain a session open on the device with a custom configuration profile. After initialization, the *DataAcquisitor* can read/write to the data acquisition card channels on demand, it can also be configured to read/write indefinitely at a given frequency, storing/retrieving the data from a shared circular buffer.

While the *DataAcquisitor* is reading samples from the biological neurons at 10 KHz, the data stored in the circular buffer is streamed to the *Normalizer* object where it is normalized following a Z-Score distribution, this normalization allows the data to be independent on the experiment, so that no changes should be made on the Spike Sorter's parameters between different experiments. The normalized data is then streamed to the *Buffer* object, where it gets splitted in fixed window sizes interleaved by a factor of one over two as shown in figure 2.1. This method of interleaving windows has two advantages, first it reduces the sorting time, since the evaluation of each window is done in parallel, thus reducing the overall time to half the window size, the other benefit is that no spike could be missed if it lies between two windows, if the windows where consecutive, that spike would be cut in half and may not be classified well. The size of the windows has been set as the mean distance between spikes (20 ms or 200 samples in our case), this distance is weakly dependent on the rhythm of the CPG, therefore it is stable across experiments. Furthermore, it is the smallest window that can be taken with no loss of information between spikes, at least for our setup.

After the *Buffer* has created the windows, the spike on each of them is centered to facilitate inference on the classifier. This is done by simply centering the maximum point of data and setting the points

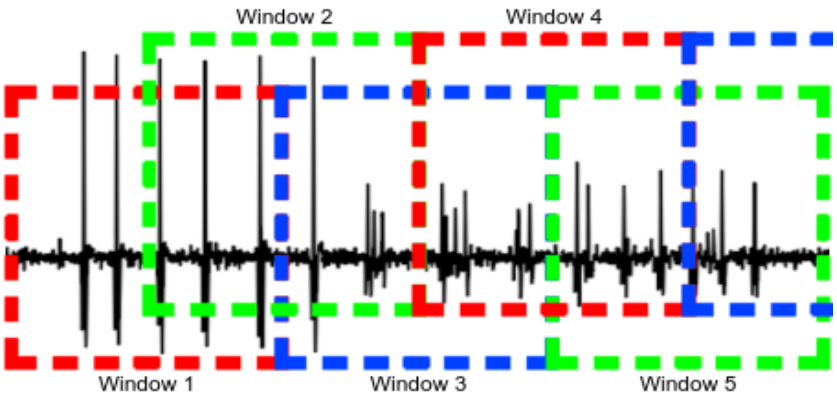

Figure 2.1: Window splitting of data

needed to fill the window to zero, normally those points set to zero are noise so there is not a significant information loss. An example of spike centering can be seen in figure 2.2.

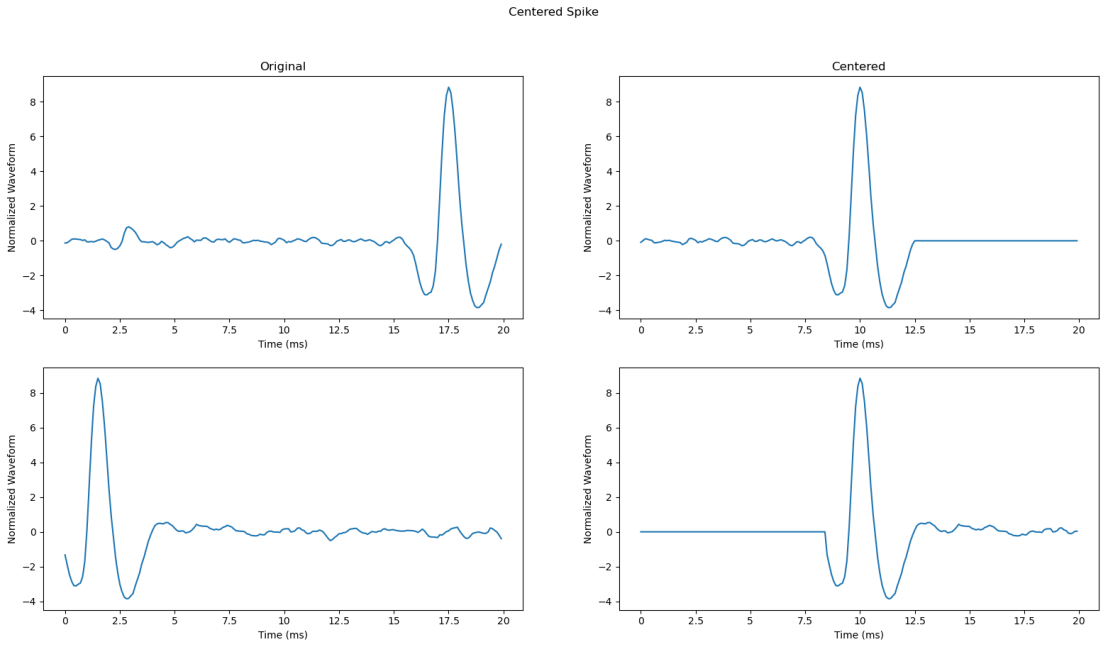

Figure 2.2: Normalized Centered Spike

2.1.3 Architecture

As previously mentioned, the new spike sorting method developed in this work only needs information from the extracellular activity of the stomatogastric ganglion. This frees one of the electrodes, that can be used to stimulate other neuron rather than recording its activity. The spike sorter is composed of three main blocks as seen in figure 2.3, the first one preprocess the sampled extracellular activity, it

normalizes and splits it in overlapping windows, as explained in subsection 2.1.2. Then it centers the spike found on each window and send those preprocessed windows for classification.

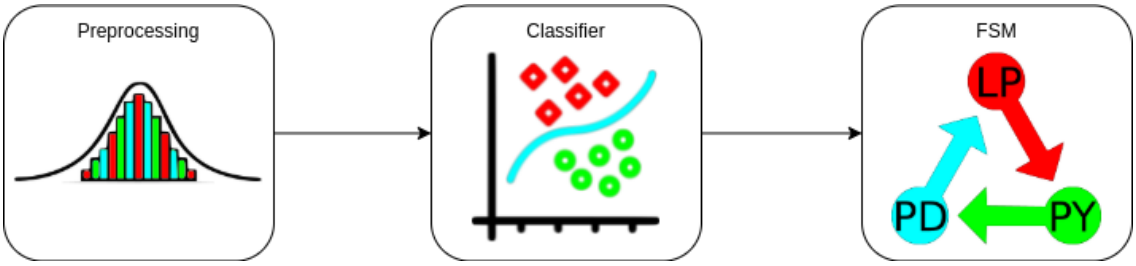

Figure 2.3: Spike Sorter Architecture

The second block of the diagram is the classifier model, given a window it predicts which neuron of the CPG: *(LP, PY or PD)* does the activity belongs to. Several supervised and unsupervised approaches have been tried to implement this classifier, all of them came down to be conformed by two parts: a feature extractor and a classifier (given the features).

The best unsupervised method relies on **k-means clusterization**, the features that should be clustered where selected manually, taking into account that they can not vary given a vertical offset (due to external potential noise) or scaling of the horizontal axis (due to stimulation). The features selected were: *Prevailing frequency, Maximum Amplitude and Energy*, those where calculated for every window. The k-means algorithm was then fed with those features, and it was trained to aim 4 clusters *LP, PY, PD and Noise*.

As it was mentioned in subsection 2.1.1, the current spike sorting technique used was very accurate, this opens the possibility to train supervised algorithms for our classifier. The feature extractor was created by training an autoencoder on the windows, then the encoder part of it was used as the feature extractor. This features where used to train a multilayer perceptron classifier, the target set was the classification inferred by the current accurate classifier.

To maximize the real time capabilities of the system, all classifiers where executed in parallel to the preprocessing block, keeping in mind that a condition must be satisfied to prevent data overflows: $predictionTime \leq bufferingTime$, and thus $predictionTime \geq windowSize * \frac{Ts}{2}$, getting a maximum overhead of $windowSize * \frac{Ts}{2}$ seconds.

The third block of figure 2.3 is a Finite State Machine (FSM) that takes advantage of the sequential behavior of the *Carcinus maenas'* CPG to get more accurate predictions. Given that the neuron spiking sequence is constant *LP-PY-PD*, the FSM checks redundancy on the output of the classifier to avoid false positive classifications that do not lie on the sequence *LP, PY, PD, LP, PY...* The FSM will break the natural sequence only given enough redundancy on the output of the classifier. This allows the system to keep in sync with the neuronal spiking pattern in case of miss classification or miss sampling. When the FSM jumps to the next state in the spike firing sequence, a new neuron activity has been detected.

2.2 Simulation

A simulation based on the **(Box2D)** physics engine has been designed and developed to facilitate the validation of several hypothesis and quickly setup experiments. Once the desired configuration for the robot has been achieved, it can be easily transferred to the physical setup designed and developed in section 2.3. The code developed in this section can be found at
https://github.com/pabolojo/TFG/tree/master/Simulation

2.2.1 Architecture

The simulation has been developed to host and emulate the behavior of a biped robot which can only perform forward walking sequential movements in a stretch line, and thus it only needs two dimensions to simulate its motion. This is the reason why the simulation motor is 2D rather than 3D, this simplifies the complexity of the calculations needed to simulate it and allows to easily implement optimisation algorithms to change the morphology of the robot and the performance of its sequential movements. The simulation was created to support three modes: *playground, genetic algorithm and experiment*.

Playground Mode

In the playground mode a single robot is created, this allows to test its mechanics and behavior in a simulated environment, before crafting a real version of it.

Genetic Algorithm Mode

The genetic algorithm mode spawns an entire population of robots and train them to obtain the best parameters that enables them to perform the desired sequential movements. This will be explained in more details in subsection 2.2.2.

Experiment Mode

And last, in the experiment mode two robots are created to test the premise exposed in section 1.2.2. One of them is controlled by the dynamical invariants and the other one with other non correlated period relations.

The robot designed in section 2.3 has only a few centimeters in height and weights no more than 300 grams. The chosen simulation is designed for large scale behavior emulation (meters, kilograms...), thus a *Custom World* had to be designed. This new environment has a *Unit Conversion* module that transforms the robot's physical characteristics to the large scale behavior of the simulation, automatically converting the units to the correct scale.

As shown in figure 2.4, the *Custom World* module hosts several *Robot* instances. The *Robot* mod-

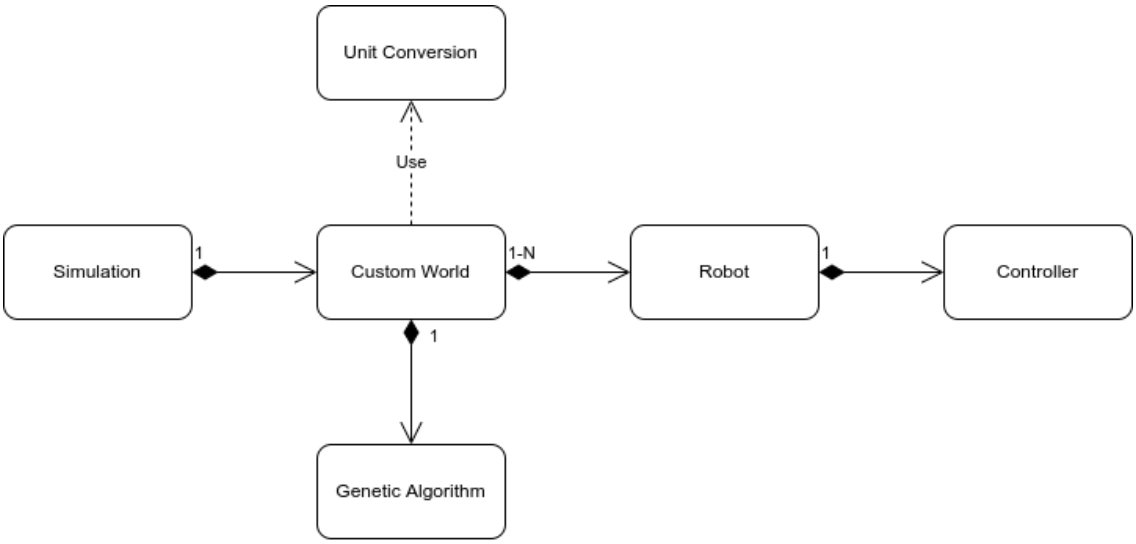

Figure 2.4: Simulation Architecture

ule creates a robot and introduces it in the simulation given its morphological description in a JSON file. The JSON file contains all the physical properties of the parts that constitute the robot, which allows to create and host robots with arbitrary morphologies. All robots are controlled by their own *Controller* module which will be explained in more details in section 2.3, as it was designed with crossover compatibility with real hardware in mind.

2.2.2 Genetic Algorithm

After the physics simulation has been successfully created and configured, robots have to be included in it to test their behavior and performance before creating a real version of them. Generally speaking, robots are gifted with locomotion abilities thanks to mechanical actuators such as motors or hydraulic mechanisms. In order to perform a desired movement, those actuators should be tweaked to attain the desired motion. Recent works such as [22] have used genetic algorithms to achieve the desired movements. In this work a similar approach is exposed to obtain the desired parameters for the actuators to achieve the desired locomotor robust sequences. These parameters will be called from now on sequential **resonance parameters**, because they are the ones that allow the system to enter in a stable sequential locomotion rhythm.

Each robot is build up of a series of mechanical coupled oscillators (motors), those coupled oscillators have to be configured with a set of parameters (oscillation frequency, amplitude, initial offset). In this work, dynamical invariants present in CPGs are used as controllers for the oscillators, those invariants are baunded in frequency. The frequency resonance parameter of these actuators are the range of frequencies that the actuator behaves in the appropriate way, thus a map from the lower frequency of the invariant to the lower frequency resonance parameter will be made. The linear correlation on the

dynamical invariant will allow the system to alter the oscillation frequencies to adapt to external inputs, in the event of getting into a non resonance state or a critical status, an electronic Proportional Integral Derivative (PID) controller could bring the system back to a resonance state.

An optimisation algorithm has been developed to determine the resonance parameters bandwidth of each of the oscillators present on the robot developed in section 2.3. A **genetic algorithm** implemented using **(PyGAD)** was used to find the optimal resonance parameters for all the mechanical oscillators of the robot, those parameters empower it with walking abilities. The initialization of the first generation of the algorithm can be done randomly or performing a kinematic analysis of the desired movements sequence, this is done to lower the number and degrees of freedom of the parameters that are being optimized by the genetic algorithm. Once those parameters have been found on the simulation, they can be easily transferred to hardware to test their behavior on a real setup.

2.3 Physical Controller

A biped robot based on mechanical oscillators was designed and developed to test in a real setup the results from the simulation. The 3D models and circuits designed and developed in this section can be found at https://github.com/pabolojo/TFG/tree/master/NeuroBip

2.3.1 NeuroBip

A biped robot structure was designed in a Computer-Aided Design (CAD) environment, the design was created with several physical features in mind. The robot should have a low center of masses, this was achieved by placing the heaviest components, in this case the batteries, in the feet of the robot, improving its stability and avoiding over oscillations while waking. The robot was designed with large hips and soles to prevent it from falling sideways, assuming that the robot cannot perform side movements simplifies the simulation by limiting the motion of it to only two dimensions. The final 3D model can be seen on figure 2.5.

The robot has four mechanical actuators, in this case servo motors (MG90S), this kind of motors were chosen because they are accurate and fast, being good candidates to perform precise oscillatory motions. They were modified to read their position and allow the controller to have feedback of the real status of the robot at any moment. In addition, visual indicators were placed on top of the hip to send light feedback of the robot. One of the Light Emitting Diodes (LEDs) indicates robot instability and the other three signal the reception of the three different neurons of the CPG that are classified by the spike sorter from the CPG sequence: *LP-PY-PD*.

The electronic components needed to build the controller were: *a microcontroller, a gyroscope and some power electronics*.

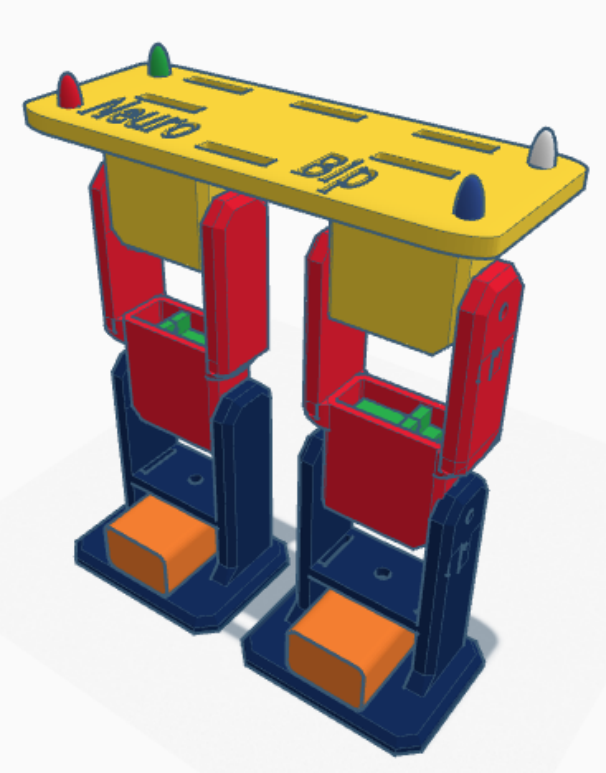

Figure 2.5: NeuroBip's 3D model developed for this work

Electronic Components

- **Microcontroller (ESP32-S2 Mini)**: This System on Chip (SoC) was chosen as the controller for the robot, it is fast and has Wifi and Bluetooth built in, this allows a variety of communication protocols to interact with the robot, in our case we will be using sockets over Wifi. Other advantage is that it supports **(MicroPython)** which will be useful for crossover compatibility of the firmware with the simulation. It has two accurate Analog to Digital Converters (ADCs) of 12 bit resolution with 10 channels each, allowing to connect all the analog sources easily to the controller.
- **Gyroscope (MMA7361)**: An analog accelerometer was chosen to measure the spacial position of the robot. This sensor automatically calibrates itself on start and sets at its outputs an analog voltage indicating the position in degrees of each spacial axis. It is fast because of its analog nature, there is no need for a digital communication protocol to transfer the desired information. In addition, a great measurement accuracy is achieved by using the high quality ADCs of the microcontroller to read its outputs.
- **Power electronics (MP2307, 1N4007, PJ-102AH)**: A mini Direct Current (DC) -DC buck converter was used to step down to 5V the voltage of the two 9V batteries in series that power the circuit through a DC connector. A rectifier diode allows to power the microcontroller from the included Universal Serial Bus (USB) and batteries simultaneously for monitoring purposes.

After all the electronic components were interconnected and tested in a prototyping board, a Printed Circuit Board (PCB) was designed and manufactured by a third party company. By designing and developping this PCB, all the electronic components are interconnected in a small space that fits at the top of the robot's hip. The PCB also includes the connectors necesary for the mechanical actuators, state indicators and batteries. The designed PCB can be seen on figure 2.6 and a final version of the

assembled robot is showcased in figure 2.7.

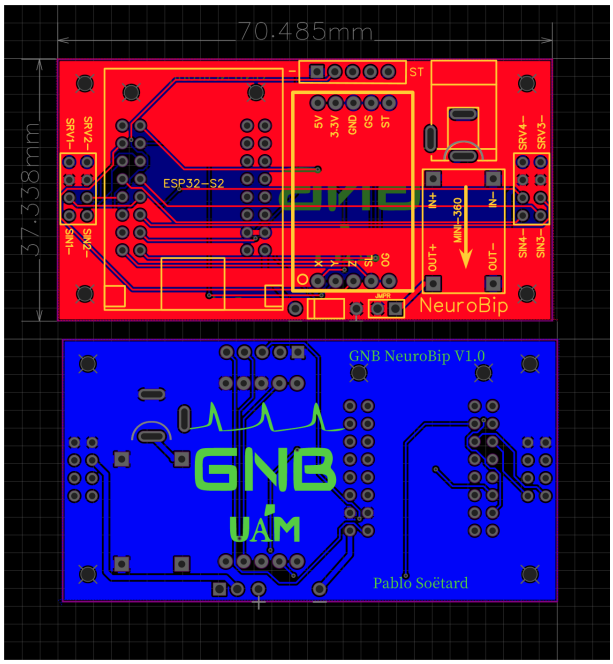

Figure 2.6: NeuroBip's PCB developed for this work

The robot and the simulation are compatible with the spike sorting output as they both use sockets to identify the neural events. As mentioned before, the controller was developed with crossover compatibility with real hardware in mind. This was achieved by flashing the MicroPython firmware on the microcontroller, this firmware enables to run Python code in the SoC. Writing the controller code in Python allows to reuse the same codebase for both the simulation and the real hardware, only requiring the implementation of a set of interfaces with real/simulated peripherals such as sensors and actuators for each of the scenarios. All of this is illustrated in diagram 2.8.

The pseudocode on algorithm 2.1 implements the behavior of Neurobip's controller. At each cycle the robot checks for new communication messages and modifies, if necessary, the invariant according to the new classified neuron of the spike sorter. Then the angles of the gyroscope and the servos are measured and sent back to the land station. A correction in the coupled oscillators angles is made by setting them to the actual servo angles. After that, the position of the robot with respect to the vertical axis is checked, if the robots is on a safe position ($|gyroAngle| \leq criticalAngle$) visual indicators are updated with the received detected neuron information and servos are set to the next angles calculated by the coupled oscillators. Otherwise, if the robot is going to fall, the instability visual indicator is turned on and the servos would be set to the angles calculated by the PID controller to save the robot from falling.

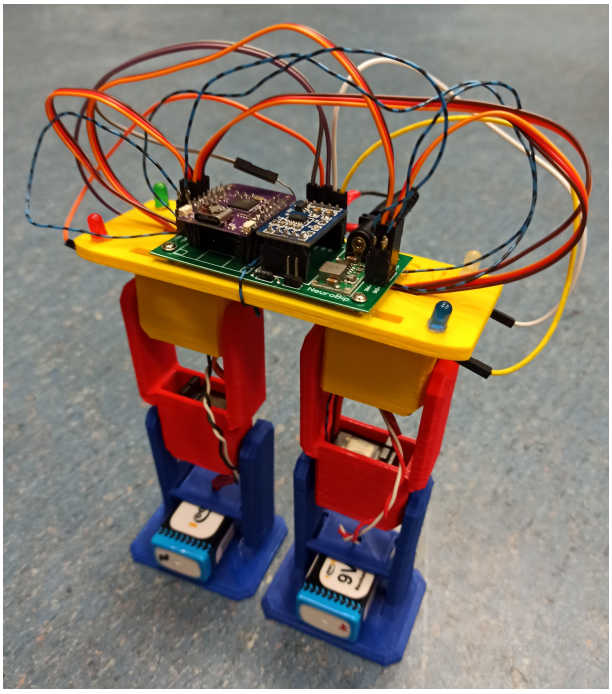

Figure 2.7: Assembled NeuroBip

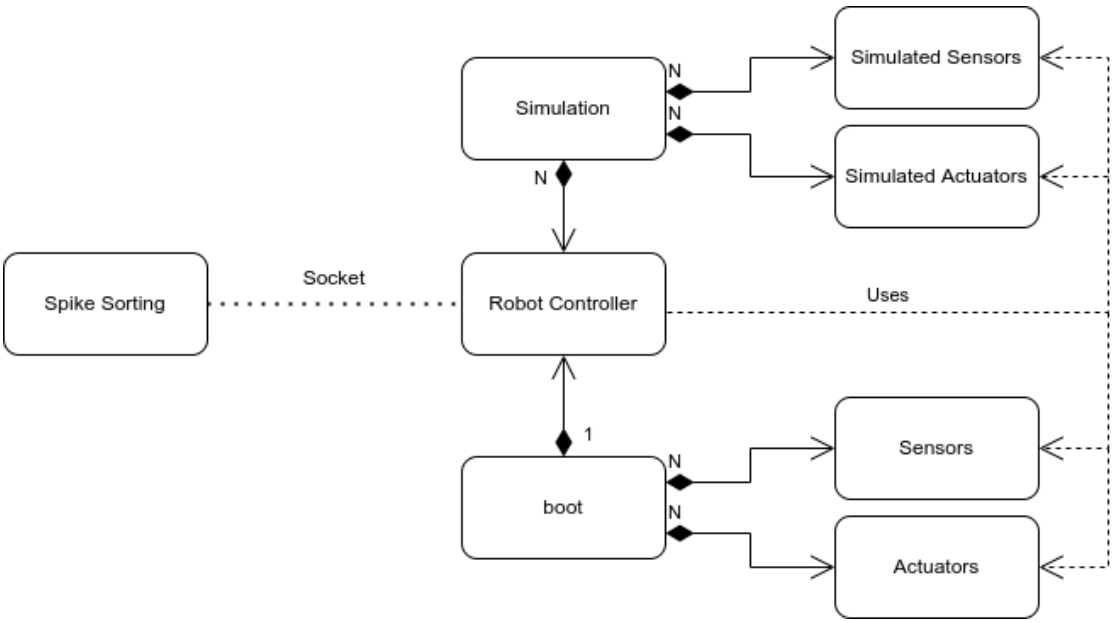

Figure 2.8: NeuroBip's Controller Interconnection Diagram

```
input: Δt in seconds

detectedNeuron, detectionTime ← communication.receive()
if detectedNeuron is not None then
    freqScale, ampScales ← invariant.calculate( detectedNeuron,
     detectionTime )
    oscillators.update( freqScale, ampScales )
end

gyroAngle ← gyro.getAngle()
servoAngles ← servos.getAngles()
communication.send( gyroAngle, servoAngles )

oscillators.setAngles( servoAngles )

if |gyroAngle| ≤ criticalAngle then
    lights.update( detectedNeuron )
    oscillatorsNextAngles ← oscillators.getNextAngles( Δt )
    servos.setAngles( oscillatorsNextAngles )
else
    lights.update( emergencyLight )
    pidAngles ← pid.save( gyroAngle, servoAngles, Δt )
    servos.setAngles( pidAngles )
end
```

Algorithm 2.1: NeuroBip main loop. This algorithm is executed indefinitely after all the sensors and actuators had been initialized during setup. It receives a Δt parameter, which is the time in seconds that has elapsed since last execution of it.

2.3.2 Coupled Oscillators

Coupled Oscillators have been extensively used to simulate CPGs and control robot locomotor sequences [13, 23]. The reason why they are used is because they offer smooth gate transitions when their parameters are changed, being more stable against abrupt changes. In this work, a set of coupled oscillators were designed to control the walking sequence of the robot developed in subsection 2.3.1. The parameters of this coupled oscillators are modified according to the invariant, and real positional feedback of the mechanical actuators is given to it to sync the oscillators with the real motion of the robot. The differential equations that govern the behavior of the coupled oscillators can be seen in equations 2.1a to 2.1d.

$$\dot{\phi}_i = \omega_i + \sum[\omega_{ij} r_j sin(\phi_j - \phi_i - \varphi_{ij})] \tag{2.1a}$$

$$\ddot{r}_i = a_r(\frac{a_r}{4}(R_i - r_i) - \dot{r}_i) \tag{2.1b}$$

$$\ddot{x}_i = a_x(\frac{a_x}{4}(X_i - x_i) - \dot{x}_i) \tag{2.1c}$$

$$\theta_i = x_i + r_i sin(\phi_i) \tag{2.1d}$$

Where:

- θ_i is the output angle of each oscillatory centre
- ϕ_i, r_i and x_i are the state variables that encode the variation in time of every phase, amplitude and offset
- The control parameters for each oscillator are ω_i (natural frequency), R_i (target amplitude) and X_i (target offset)
- a_r and a_x are constant positive gains for Eqns. 2.1b and 2.1c. In our case: $a_r = a_x = 2\pi\ rad/s$
- Finally, ω_{ij} and φ_{ij} are respectively the coupling weights and phase biases that determine how oscillator j influences oscillator i

Three oscillators have been defined, one for the hip and the other two for the knees. The one from the hip controls the two mechanical oscillators that are situated in the top part of each leg, one is set to the $angle$ given by the oscillator and the other to the $-angle$ (being 0º the vertical axis). The two oscillators of the knees are coupled to the main hip oscillator with a factor of 0.5, in addition, they oscillate at the same frequency and have the same amplitude, but may differ in phase.

The differential equations that describe the oscillators behavior are continuous, they were implemented in our digital (discrete) controller using **Euler's method**. The code implementation of the equations can be seen in appendix A.

3

Experiments and Results

3.1 Extracellular Spike Sorting Accuracy

As mentioned in subsection 2.1.2, a wrapper written in Python for the data acquisition card had to be developed. In order to evaluate its efficiency compared to the real time C version of it, a set of time delay measurements were performed. A square wave at 100 Hz and 50 % duty cycle was generated by a function generator for one minute and, using the Python wrapper, it was read from one of the channels of the data acquisition card and forwarded into an output channel of the same card. Using another data acquisition card and the real time C recorder, both signals, the one from the function generator and the one emitted by the Python wrapper via the other acquisition card were recorded. The number of samples delayed between both signals acquired at 10 kHz was calculated and its illustrated in figure 3.1, it can be seen that the delay of each sample is in most of the cases real time, having and averaged delay of 0.21 samples.

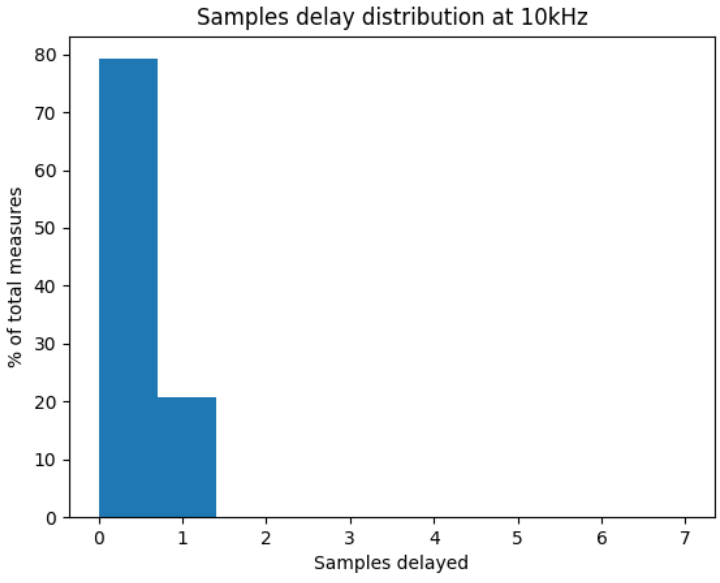

Figure 3.1: Samples delay distribution of python data acquisition wrapper

In subsection 2.1.3 the architecture of the spike sorter was designed and developed, this architecture was constituted by three elements: *preprocessing unit, classifier and FSM*. The data recorded from the data acquisition card is streamed to the preprocessing unit, were it is normalized, splitted in windows and spikes are centered on each of the windows. The windows are then passed to the classification model for inference. Two models have been tested, one unsupervised *(k-means clusterization)* and one supervised *(Encoder + Perceptron)*.

K-Means Clusterization

As stated in subsection 2.1.3, the features that define the coordinates of the points that have to be clustered were selected manually. Those features were chosen taking into account that they can not vary given a vertical offset (due to external potential noise) or scaling of the horizontal axis (due to stimulation). The features selected were: *Prevailing frequency, Maximum Amplitude and Energy*. After having calculated those features for every window of a low noise record of the extracellular activity of the *Carcinus maenas'* CPG, the k-means clusterization algorithm was run on them, obtaining the clusters (represented by different colors) and centroids of them (represented by stars) shown in figure 3.2.

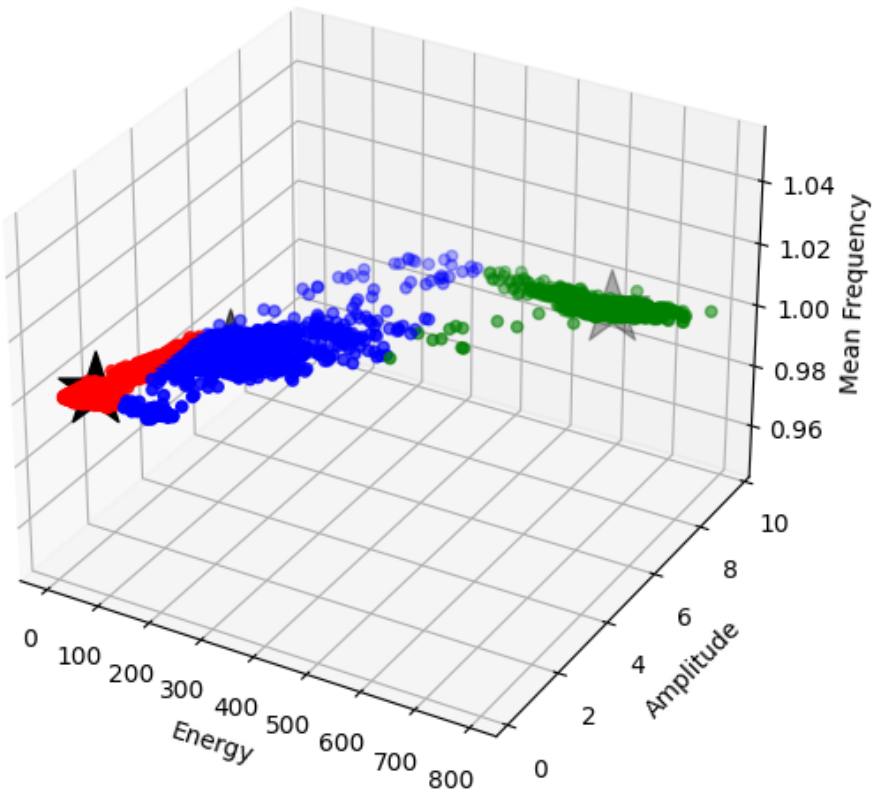

Figure 3.2: K-Means manual features clustering example

When the k-means clusterization algorithm has been trained, it can start performing infernece and classify neural events. An example of the classification of the model already trained can be seen in

figure 3.3, the sequence *LP-PY-PD* is being detected and colored in *blue-red-green*, although there are multiple false positives, most of this noise is removed by the FSM.

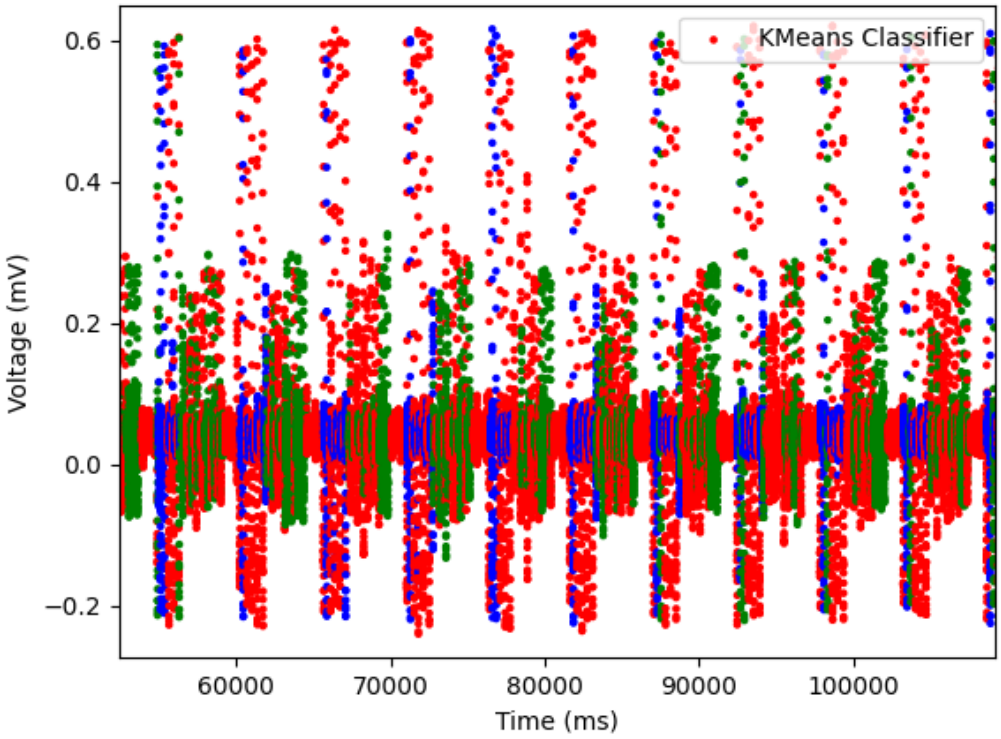

Figure 3.3: K-Means spike sorting

Encoder + Perceptron

In this case the feature extraction was performed by an encoder composed of 200 input neurons (same length as the window size), a bottleneck of 32 neurons and 200 output neurons. This encoder was extracted from an autoencoder trained on a set of preprocessed windows from a low noise extracellular record. To ensure the functionality of the trained encoder, it was tested against opposed snapshots of the same spike. The premise was that the inference should yield the same features for both snapshots, as they represent the same spike. Both snapshots were fed into the encoder and the euclidean distance and dot product (alignment) of the resulting features vector were calculated. As seen in figure 3.4, the distance between the two features vectors is 0 and the alignment is 1, therefore, as expected, the two feature vectors are the same.

The output of the encoder is then attached to a multi-layer perceptron classifier. In this case of supervised learning, the outcome of the accurate threshold classifier was used as the target values for the perceptron training. Due to potential differences between experiments, the train and test sets were made up of different experimental records. A classification example of this supervised architecture

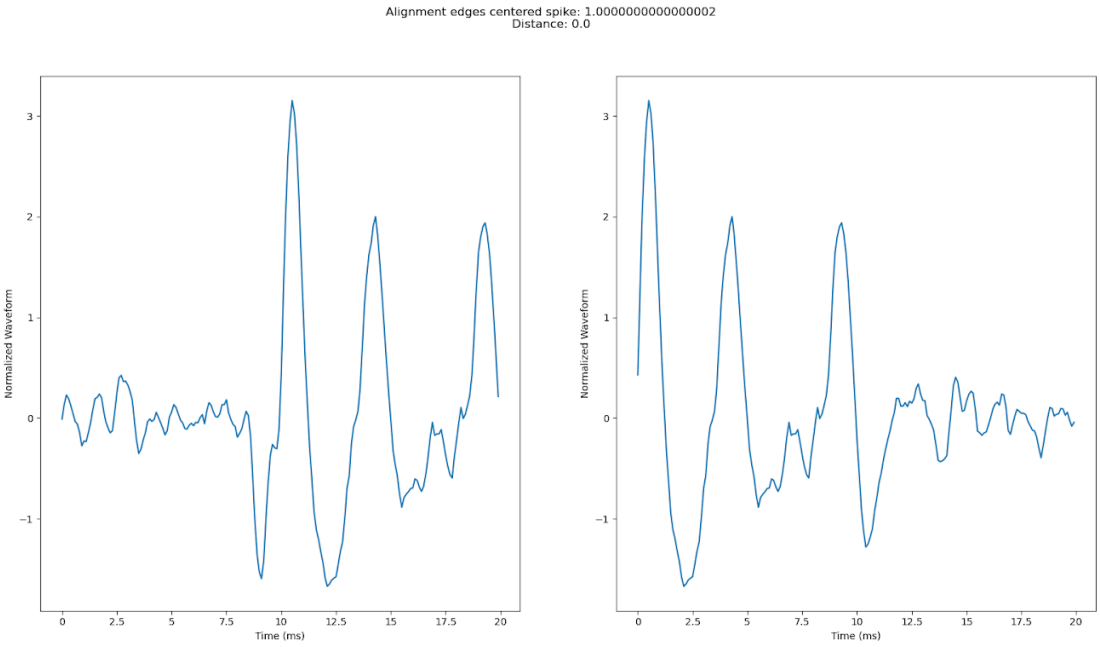

Figure 3.4: Affinity comparison of spike snapshotted in opposed instants

already trained can be seen in figure 3.5. The sequence *LP-PY-PD* is being detected and colored in *blue-red-green*, we can appreciate less number of false positives in this example than in the one of k-means clusterization, once again, this results will improve after applying the sequential rectifier FSM.

To test the accuracy of the models exposed above, the averaged **Jaccard index** of each possible neuron classification was used. The *Jaccard index* of two sets is calculated by dividing the intersection over the union of them, as described in equation 3.1. In our case, the sets are the detection periods of a neuron given by the assessed classifier and the ground true threshold classifier. Therefore, we will get a *Jaccard index* for each of the possible classifications *(LP-PY-PD)*, leaving room for possible improvements of the model neuron by neuron.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

	Accuracy pre-FSM (%)	Accuracy post-FSM (%)	Inference time (s)
K-Means clusterization	39.79	60.64	0.1067
Encoder + Perceptron	41.72	67.29	0.1800

Table 3.1: Spike sorters accuracy comparison

Table 3.1 shows the resulting *Jaccard index* for each model. Those *Jaccard indices* were calculated by averaging the *Jaccard index* of each of the possible classifications *(LP-PY-PD)* given by the models. It can be seen that the accuracy of both models increases substantially after going trough the sequential

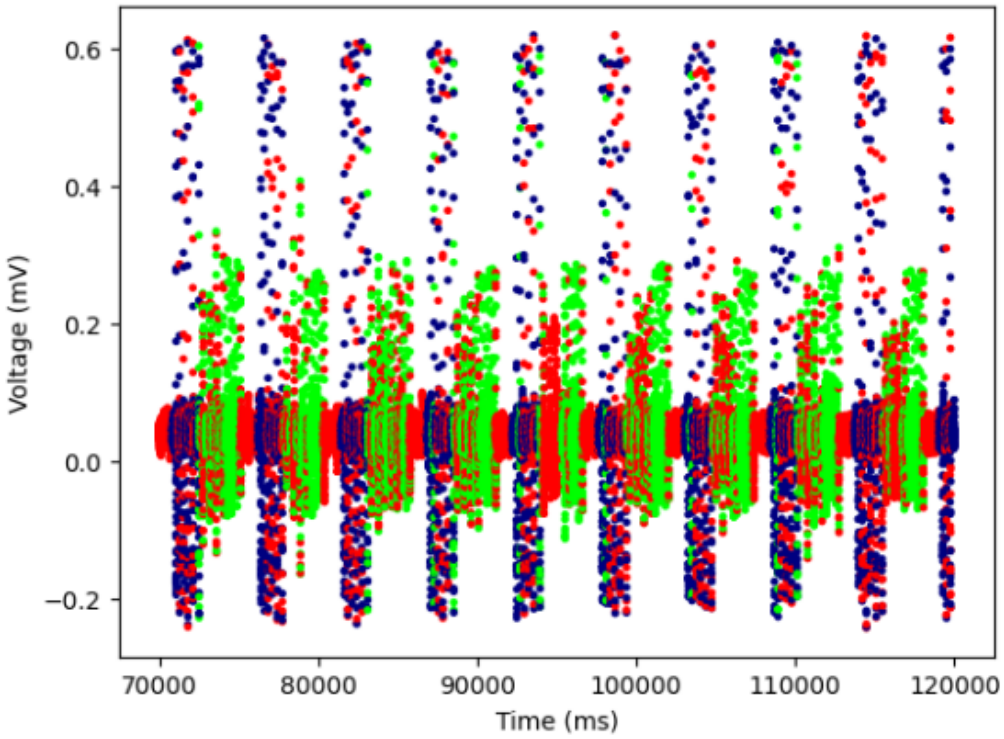

Figure 3.5: Encoder + Perceptron spike sorting

rectifier FSM, reaching an average of almost 70 % the accuracy of the ground truth classifier.

3.2 Resonance Parameters

Once the simulation environment has been created and configured to host robots, it was time to include a simulated model of the biped robot developed in section 2.3. In this work, locomotor abilities are given by mechanical coupled oscillators, those oscillators have 3 parameters each: *frequency, amplitude and phase (relative to other of the coupled oscillators of the system)*. In case of *NeuroBip*, it is composed of 4 servo motors, therefore 4 oscillators should be needed to drive them, one per motor. As said before, each oscillator has 3 parameters and we have 4 of them, that means that up to 12 parameters have to be tuned.

By analyzing the problem of a biped robot, we can see that, if the vertical axis is taken as the origin (0º), the two oscillators in the hip in fact are the same with negated angles. In addition, we can assume that both knees have a similar behavior, sharing their joint amplitude and frequency, but not necessarily their phase. As the phase is relative to another coupled oscillator, we will only have to tune $\#oscillators - 1$ phases. With all those simplifications due to the nature of this particular problem, we have reduced the number of parameters to be tuned from 12 to only 6.

As stated in subsection 2.2.2, a genetic algorithm has been used to find those parameters. The initial population genome of the genetic algorithm can be selected randomly, but this could lead to slow training and may not result in optimal parameters. A better choice would be to perform a kinematic analysis of the movements that should be mimicked by the robot. In our case, after analyzing the dynamics of hip and knee joints from [24], we obtain the relations and conditions that the parameters should satisfy, those are summarized in table 3.2. We can see that knee joints are only allowed to bend backwards, this is because humans use their arms to balance the inertia caused by their walking sequence. In our case, *NeuroBip* does not have any stabiliser mass, this is the reason why the amplitude of its knees was set to rotate further than humans, to allow stable behavior of the biped robot while walking.

Parameter Conditions
$kneesFrequency = 2 * hipFrequency$
$0\ Hz < hipFrequency < 2\ Hz$
$amplitudeHip < amplitudeKnees$
$0\ rad < amplitudeHip < \frac{\pi}{4}\ rad$
$0\ rad < amplitudeKnees < \frac{\pi}{4}\ rad$
$-\frac{\pi}{2}\ rad < phaseKnee1 < \frac{\pi}{2}\ rad$
$-\frac{\pi}{2}\ rad < phaseKnee2 < \frac{\pi}{2}\ rad$

Table 3.2: Genetic Algorithm Parameter Conditions

We can see in table 3.2 that, by relating the knees and the hip frequencies, we can further reduce the number of parameters needed to 5. Therefore, the genome we are looking for will consist of the following parameters *[hipFrequency (Hz), hipAmplitude (rad), kneesAmplitude (rad), phaseKnee1 (rad), phaseKnee2 (rad)]*. As stated above, we can initiate them randomly, following the conditions on table 3.2, or we can perform a kinematic analysis as the one in [24] for human walking. Looking at the gait graphs for hip and knee joint angles in [24], we can craft the following initial genome for the first population of the genetic algorithm $[0.5, 15 * \frac{\pi}{180}, 20 * \frac{\pi}{180}, -\frac{\pi}{2}, -\frac{\pi}{2}]$.

Using that genome as the initial seed for the first population of the genetic algorithm, the algorithm was trained during 60 generations, each generation executed for 60 seconds and having 100 individuals (population size), getting the train graph shown in figure 3.6. Each of the 100 individuals in the population is executed in parallel as shown in figure 3.7, their fitness function, that the algorithm is trying to maximize, is the maximum distance travelled by the individual. The 10 best individuals of each generation are included in the next generation and their offspring will make up the other 90 individuals of the population. That offspring is generated following a uniform crossover between the parents and mutations are performed on their genes with a probability of 0.2 per gene. A visual example of the evolution of the training during four generations can be seen in figure 3.7, the link to a video showcasing the evolution is included in the caption. In addition, the code snippet with the initialization of the genetic algorithm configuration in appendix A.

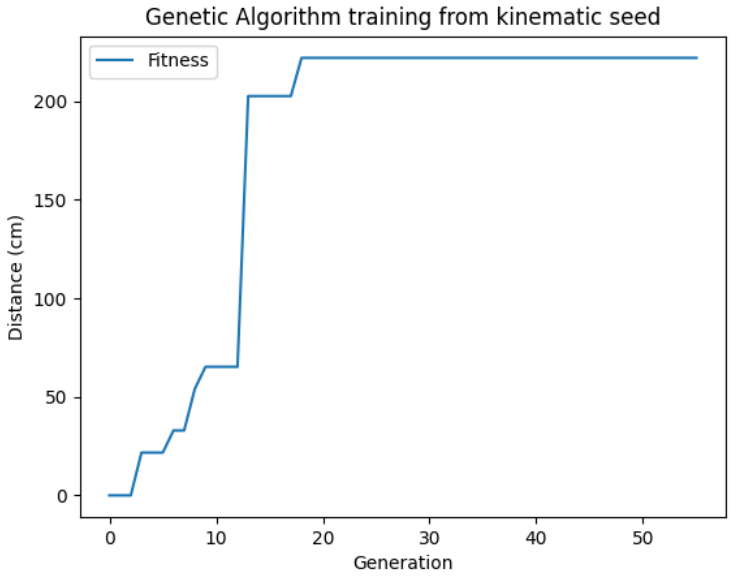

Figure 3.6: Genetic Algorithm training from kinematic seed

After the first training has finished, the genes of the best individual could be reintroduced as the initial seed for the first population and retrain the genetic algorithm, obtaining a finer tuning of the parameters. When the obtained genes have been tested genuinely good on the playground mode of the simulation, they can also be tested on real hardware, taking advantage of the crossover compatibility

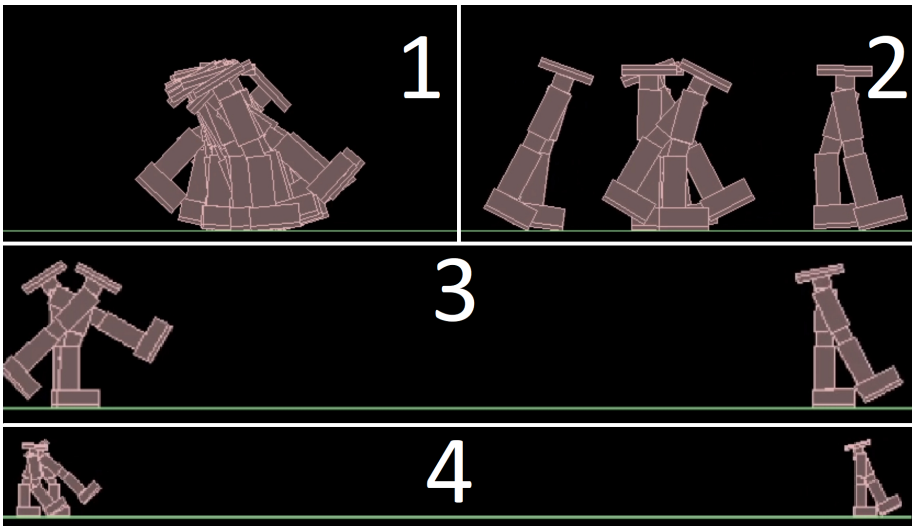

Figure 3.7: Genetic Algorithm training of robots from kinematic seed, a video of the evolution of the first 5 generations can be seen at **https://github.com/pabolojo/TFG/tree/master/videos/genetic_algorithm_5_generations.mp4**

of the controller program.

3.3 Dynamical Invariant stability vs other interval relationships

To test the hypothesis stated in subsection 1.2.2, offline and online experiments were performed to demonstrate the robustness of the rhythm created by the CPG and the advantages of dynamical invariant based controllers against other interval relationships.

As shown in section 3.2, the four coupled mechanical oscillators of *NeuroBip* are controlled by 5 parameters: a frequency, 2 amplitudes and 2 phases. The dynamical invariant will control the 3 first parameters, while the phases between legs will be kept constant.

The cycle interval *(LP-LP)* is mapped to the frequency resonance parameter *(hipFrequency)* of the actuator, increasing or decreasing all the oscillator's frequencies simultaneously on a linear way when one of the neurons is excited by an electrode for control feedback purposes. On the other hand, the two other intervals that form the dynamical invariants with respect to the cycle interval, *(LP-PD and PY-PD)*, are used to modulate the amplitudes of the oscillators *(hipAmplitude and kneesAmplitude)* respectively.

At the beginning of the transmission of the CPG sequence, some cycles are taken to calibrate the linear factor mapping between the CPG intervals and the robot parameters. After the calibration is finished, the new incoming intervals (multiplied by a calibration factor), control the parameters of the robot, and thus its locomotion.

Offline Experiments

These offline experiments were conducted to asses the efficacy of dynamical invariants for the control of robots in a flexible and easily replicable environment, in this case the simulation. Two types of offline experiments were designed, both of them involved 2 robots, one will be receiving the intervals that form the dynamical invariants and the other intervals with no correlation between them at all.

The first type of experiment indirectly measures the stability of the rhythm by storing the distance travelled and the time it took for the robot to arrive there before it falls. Both robots receive the neural events synced, when both of them have fallen, they are respawned at the origin of the simulation and the process starts again. In the end the mean time and distance of the robots during the simulated experiment is calculated.

The second type of experiment measures the stability by counting the number of times each robot has fallen. In this scenario, as soon as a robot falls it is respawned at the origin of the simulation, not needing to wait for the other robot to fall too. In the end the number of times both robots have fallen during the experiment is calculated.

Offline experiments were performed by directly streaming the output of a pre-spike sorted low noise record to the two robots in the simulation. One of the robots was receiving the intervals involved in dynamical invariants, whose correlations can be seen in figure 3.8. We can see the presence of strong linear correlations between the intervals *LP-PD and PY-PD* with the cycle interval (LP-LP) there. The second robot was controlled with other intervals, being one of them not correlated at all with the cycle interval, as seen in figure 3.9, where the *PD burst* shows no correlation with the cycle interval.

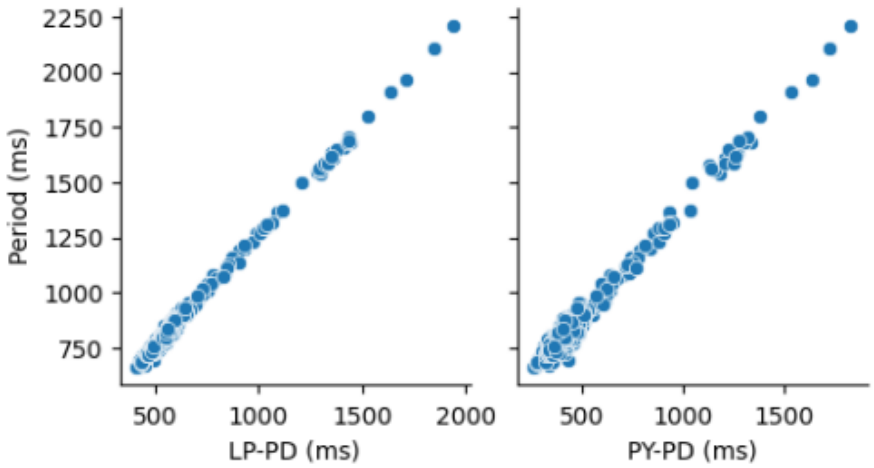

Figure 3.8: Dynamical Invariant intervals correlations

After 30 minutes of neural activity being sent to the simulation, the results of both types of experiments can be seen in tables 3.3 and 3.4 respectively. A significant difference in the performance of both robots is appreciated, doubling the stability of the dynamical invariant controlled robot to the one

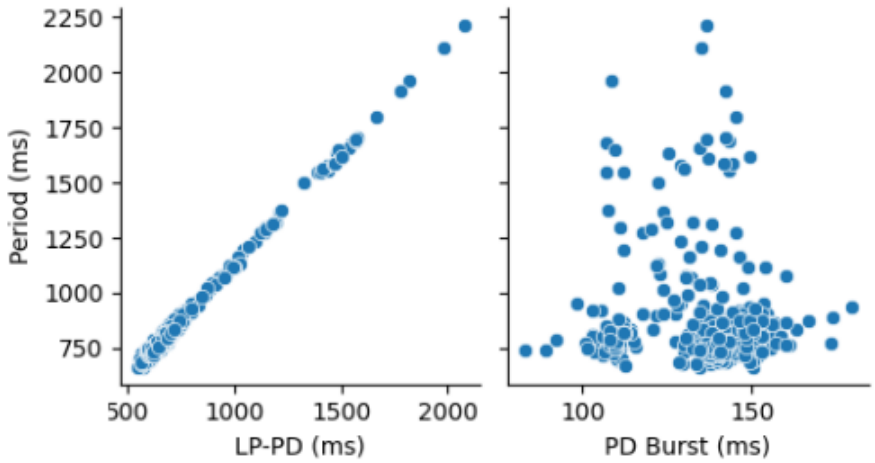

Figure 3.9: Other intervals correlations

controlled by other intervals.

The same experiment was performed in real hardware, but an issue with the robot's soles friction was discovered, *NeuroBip* was unable to walk due to the low friction its soles had with the floor. Despite that, the robot successfully performed the desired sequential movements and proved to be stable when dynamical invariant intervals were applied.
This can be seen at **https://github.com/pabolojo/TFG/tree/master/videos/offline_neurobip.mp4**

	Mean Time (s)	**Mean Distances (cm)**
Dynamical Invariant intervals	39.68	209.36
Other intervals	17.28	81.82

Table 3.3: Offline experiment distance and time results

	Number of Falls in 30'
Dynamical Invariant intervals	47
Other intervals	106

Table 3.4: Offline experiment number of falls results

Online Experiments

The same setup as in offline experiments was reproduced, but in this case live biological signals from living CPGs were used as the spike sorter input. In addition, real time feedback was given to the living cells by injecting current in one of the PD neurons present in the CPG. This feedback current was injected each time the robot started to destabilize, and proved to be effective to stabilize the robot again by relaxing the CPG rhythm and thus the dynamical invariants relations too. The results from the experiment can be seen in the video linked at the caption of figure 3.10, where a real time evolution

of the CPG neural activity, the spike sorter detections and the interaction of the simulated robot with the CPG are illustrated. As already mentioned, the real hardware *NeuroBip* could not be used due to friction issues.

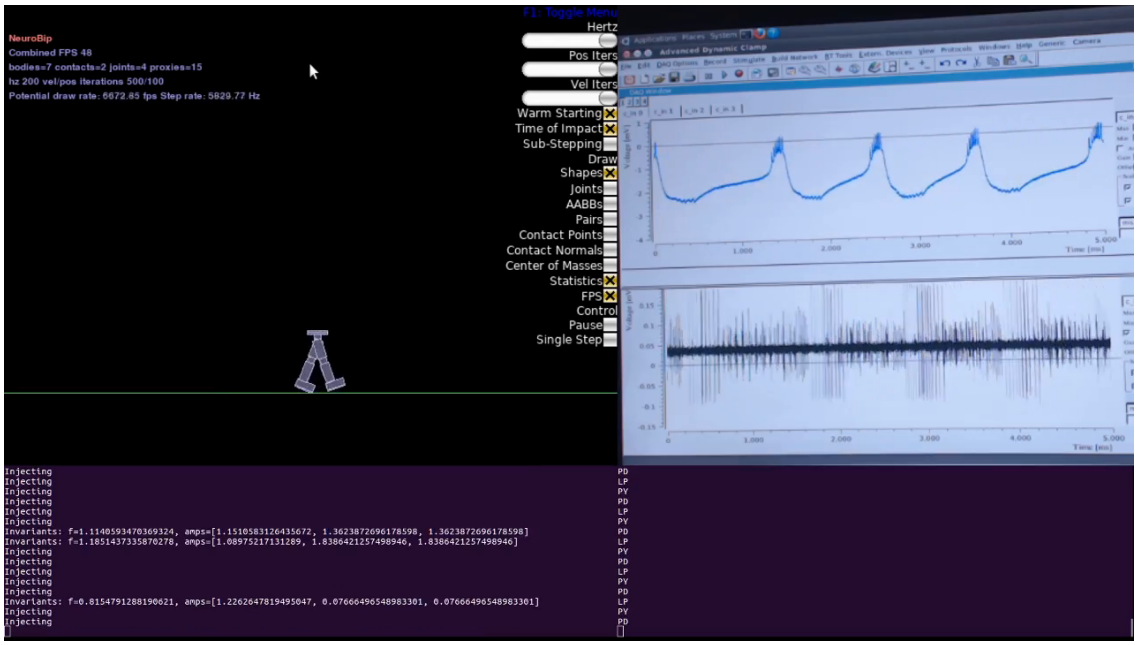

Figure 3.10: Online bidirectional Hybrot experiment, a complete video of the bidirectional interaction between the robot and the CPG can be seen at **https://github.com/pabolojo/TFG/tree/master/videos/online_experiment.mp4**

3.4 Total reaction time

The last general milestone that was exposed in subsection 1.2.3 is the reaction time of our system. To identify possible bottlenecks in the system exposed in subsection 1.2.4, the reaction time of the whole system was measured in two phases. At first the spike sorter delay was measured as seen in table 3.1, the lower inference time model was used on this results. Then the communication and processing time of the controller until the mechanical actuators react to the incoming signal was measured.

To do so, a digital pin was configured in the controller to turn on when the mechanical actuators are being reconfigured with the received new dynamical invariant data. A reference signal was generated when the packets were sent to the controller, both the reference signal and the digital pin output of the controller were measured simultaneously by a data acquisition card. This measurements resulted in an average actuation delay of $0.035\ s$, the distribution of delays can be seen in figure 3.11.

The final results of reaction times of both parts of the system can be seen in table 3.5. We can see that the overall time required for the neural events to affect the mechanical actuators is $0.145 \pm 0.031\ s$ (Spike sorter and Transmission + Controller), this would be the $CPG \rightarrow Controller$ OWD. We can

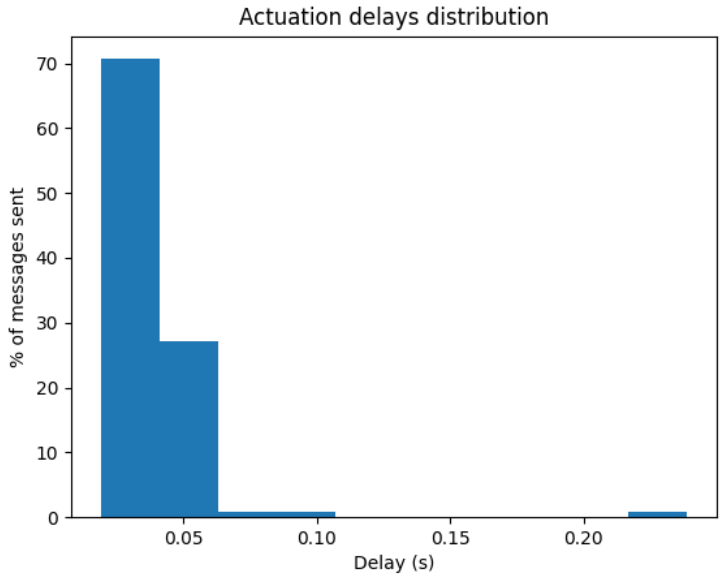

Figure 3.11: Time delay distribution of robot reaction time

	Time (s)
Spike sorter	0.11 ± 0.03
Transmission + Controller	$0.035 \pm 1e-3$
Transmission + Injection	$0.035 \pm 1e-3$
Total	0.18 ± 0.03

Table 3.5: System reaction times

safely assume that the time that the controller feedback takes to arrive to the CPG, that would be the $Controller \rightarrow CPG$ OWD (Transmission + Injection), is the same as the *Transmission + Controller* delay. And thus we have a final reaction time or RTT of $0.18 \pm 0.03\ s$ or $180 \pm 30\ ms$, which is smaller than $250\ ms$, that was the maximum acceptable reaction time stipulated in subsection 1.2.3.

4

Conclusions and Future Work

4.1 Conclusions

In this work, a reliable and scalable architecture for biological based controllers for robotic systems has been designed and developed. This architecture has proven human-like reaction times, as described in section 3.4, only being limited by the reaction time of the biological system. This elucidates potential hybridization of human brain and robotic technologies with applications in neurorehabilitation treatments and the design of biomedical systems such as exoskeletons or intelligent biomechanic prosthesis.

We have shown that Hybrots are powerful tools that allow the validation of hypothesis on the functional dynamics of CPGs. In this case, the restrictive flexibility imposed by the dynamical invariants in building the sequence intervals of the CPG functional rhythm has been proven.

As demonstrated by this work, CPG rhythms are suitable for locomotion tasks, as their present a sequential rhythm that repeats over time indefinitely. Furthermore, the constrains and correlations between intervals that build robust neural sequences called dynamical invariants have been assessed and an advantage over other type of biological signals for the control of robotics systems has been proven in section 3.3. During the experiments performed, the robots that were controlled by dynamical invariants had a more robust rhythm, this was reflected in the distance they travelled and the number of times they fell in comparison with their non-dynamical invariant controlled counterparts.

New techniques specifically targeted towards sequential biological signal detection, like the ones present in CPGs, were developed in section 2.1 and shown effective against simple biological systems such as the *Carcinus maenas'* gastric CPG, as demonstrated in section 3.1. In addition, thanks to the creation of a physical robot simulation developed in section 2.2, the parameters needed for the control of a robotic system by the dynamical invariants relations can be obtained easily with optimisation algorithms, as shown in section 3.2.

4.2 Future Work

A generalization of the spike sorting technique for noisier and more complex biological signals could be developed, enhancing the classifier models and taking advantage of the FSM for robust rhythm preservation. As shown in table 3.5, the main bottleneck of this system regarding delays is the spike sorter. The architecture presented in this work can be is easily implemented in ASICs, this would boost the time performance of this element substantially, creating a system closer to real-time reaction times.

For larger and more complex systems, the robot controller and simulation should be generalized to handle an arbitrary number of oscillators. This will allow the creation of more complex robots with more degrees of freedom and a wider locomotion spectrum.

Given the slow rhythm generated by the *Carcinus maenas'* CPG, an emergency electronic controller could be incorporated in the system, as mentioned in subsection 2.3.1. This electronic PID controller would be activated when the robot falls into a critical state that can not be solved by the CPG (due to its low reaction time). The electronic PID controller would restore the robot to a stable state.

Regarding the interface with the biological system, a PID controller could be created to automatically adapt the current injected to the CPG depending on the values that are being read and the type of neuron on which the feedback is being injected. This would potentially reach faster and more accurate response of the CPGs, increasing the reaction time of the overall system.

The methodology exposed in this work could be generalized to other CPGs beyond the pyloric circuit of *Carcinus maenas*. If adapted to the ones present in the human nervous system, this could be used for the development of exoskeletons.

For simplicity, in our experimental setup a straightforward feedback inhibitory stimulation has been used. Feedback involving both excitation and inhibition (implemented with electrical currents or chemical microinjection) on multiple cells could potentially result in extended Hybrot functionality.

Bibliography

[1] S. R. y Cajal, *Les nouvelles idées sur la structure du système nerveux chez l'homme et chez les vertébrés*. Reinwald, 1894.

[2] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, no. 2, pp. 245–258, 2017.

[3] L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain computer interfaces, a review," *sensors*, vol. 12, no. 2, pp. 1211–1279, 2012.

[4] X. Gao, Y. Wang, X. Chen, and S. Gao, "Interface, interaction, and intelligence in generalized brain–computer interfaces," *Trends in cognitive sciences*, vol. 25, no. 8, pp. 671–684, 2021.

[5] A. Kubota and L. D. Riek, "Methods for robot behavior adaptation for cognitive neurorehabilitation," *Annual review of control robotics and autonomous systems*, p. 10285172, 2021.

[6] A. I. Selverston, M. I. Rabinovich, H. D. I. Abarbanel, R. Elson, A. Szücs, R. D. Pinto, R. Huerta, and P. Varona, "Reliable circuits from irregular neurons: a dynamical approach to understanding central pattern generators," *Journal of Physiology-Paris*, vol. 94, no. 5-6, pp. 357–374, 2000.

[7] A. I. Selverston, "Invertebrate central pattern generator circuits," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 365, no. 1551, pp. 2329–2345, 2010.

[8] A. Potter, T. B. DeMarse, D. J. Bakkum, M. C. Booth, J. R. Brumfield, Z. Chao, R. Madhavan, P. A. Passaro, K. Rambani, A. C. Shkolnik, *et al.*, "Hybrots: hybrids of living neurons and robots for studying neural computation," *Proceedings of brain inspired cognitive systems*, pp. 1–5, 2004.

[9] S. M. Potter, D. A. Wagenaar, R. Madhavan, and T. B. DeMarse, "Long-term bidirectional neuron interfaces for robotic control, and in vitro learning studies," vol. 4, pp. 3690–3693, 2003.

[10] L. Sun, Y. Yu, Z. Chen, F. Bian, F. Ye, L. Sun, and Y. Zhao, "Biohybrid robotics with living cell actuation," *Chem. Soc. Rev.*, vol. 49, pp. 4043–4069, 2020.

[11] Z. Hosseinidoust, B. Mostaghaci, O. Yasa, B.-W. Park, A. V. Singh, and M. Sitti, "Bioengineered and biohybrid bacteria-based systems for drug delivery," *Advanced Drug Delivery Reviews*, vol. 106, pp. 27–44, 2016. Biologically-inspired drug delivery systems.

[12] M. Val-Calvo, J. R. Álvarez Sánchez, J. Alegre-Cortés, F. de la Paz-López, J. M. Ferrández-Vicente, E. Fernández-Jover, and I. Val-Calvo, "Frequency variation analysis in neuronal cultures for stimulus response characterization," *Neural Computing and Applications*, vol. 32, pp. 5027–5032, 5 2020.

[13] C. Garcia-Saura, "Central pattern generators for the control of robotic systems," *Arxiv*, p. 1509.02417, 9 2015.

[14] D. Gutierrez-Galan, J. P. Dominguez-Morales, F. Perez-Peña, A. Jimenez-Fernandez, and A. Linares-Barranco, "Neuropod: a real-time neuromorphic spiking cpg applied to robotics," *Neurocomputing*, vol. 381, pp. 10–19, 2020.

[15] F. Dzeladini, N. Ait-Bouziad, and A. Ijspeert, "CPG-based control of humanoid robot locomotion," *Humanoid Robotics: A Reference*, pp. 1–35, 2018.

[16] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review.," *Neural Netw*, vol. 21, pp. 642–653, may 2008.

[17] M. Reyes-Sanchez, R. Amaducci, I. Elices, F. B. Rodriguez, and P. Varona, "Automatic adaptation of model neurons and connections to build hybrid circuits with living networks," *Neuroinformatics*, vol. 18, pp. 377–393, 6 2020.

[18] R. Amaducci, M. Reyes-Sanchez, I. Elices, F. B. Rodriguez, and P. Varona, "RTHybrid: a standardized and open-source real-time software model library for experimental neuroscience," *Frontiers in Neuroinformatics*, vol. 13, p. 11, 2019.

[19] I. Elices, R. Levi, D. Arroyo, F. B. Rodriguez, and P. Varona, "Robust dynamical invariants in sequential neural activity," *Scientific Reports*, vol. 9, 12 2019.

[20] A. Garrido-Peña, I. Elices, and P. Varona, "Characterization of interval variability in the sequential activity of a central pattern generator model," *Neurocomputing*, vol. 461, pp. 667–678, 2021.

[21] H. G. Rey, C. Pedreira, and R. Q. Quiroga, "Past, present and future of spike sorting techniques," *Brain research bulletin*, vol. 119, pp. 106–117, 2015.

[22] B. Zahn, J. Fountain, T. Houliston, A. Biddulph, S. Chalup, and A. Mendes, "Optimization of robot movements using genetic algorithms and simulation," in *Robot World Cup*, pp. 466–475, Springer, 2019.

[23] F. Herrero-Carrón, F. Rodríguez, and P. Varona, "Bio-inspired design strategies for central pattern generator control in modular robotics," *Bioinspiration and Biomimetics*, vol. 6, p. 16006, mar 2011.

[24] X. Xu, R. W. McGorry, L. S. Chou, J. hua Lin, and C. chi Chang, "Accuracy of the microsoft kinect™ for measuring gait parameters during treadmill walking," *Gait and Posture*, vol. 42, pp. 145–151, 7 2015.

TERMINOLOGY

dynamical invariant constraints in shaping intervals that build the CPG sequence, these linear correlations overtime have been shown to be linearly dependent on the CPG rhythm through time.

Euler's method Euler's method is a first-order numerical procedure for solving ODE with a given initial value..

genetic algorithm A genetic algorithm is a search heuristic inspired by Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

k-means clusterization K-Means Clusterization is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

resonance parameter Set of parameters that allows the robot system to successfully perform the desired sequential movements.

spike sorting Spike sorting is a class of techniques used in the analysis of electrophysiological data.

Acronyms

ADCs Analog to Digital Converters.

ASICs Application-Specific Integrated Circuits.

BMI Brain-Machine Interfaces.

CAD Computer-Aided Design.

CPG Central Pattern Generator.

CPGs Central Pattern Generators.

DC Direct Current.

FSM Finite State Machine.

GCPs Generadores Centrales de Patrones.

LEDs Light Emitting Diodes.

LP Lateral Pyloric.

ODE Ordinary Differential Equation.

OWD One Way Delay.

PCB Printed Circuit Board.

PD Pyloric Dilator.

PID Proportional Integral Derivative.

PY Pyloric.

RTT Round Trip Time.

SoC System on Chip.

USB Universal Serial Bus.

APPENDICES

A

Code Snippets

Code A.1: Discrete implementation of the ordinary differential equations for the Kuramoto model using Euler's method.

```
def getNext(self, deltaTime):
    output = [0] *self.N
    new_sA = [0] *self.N
    new_sr = [0] *self.N
    new_sx = [0] *self.N
    new_sr_d = [0] *self.N
    new_sx_d = [0] *self.N

    for i in range(self.N):
        sA_d = self.tW[i]
        for j in range(self.N):
            sA_d += self.couplingWeights[i][j] *self.sr[j] *sin(self.sA[j] -self.sA[i] -self.phaseBiases[i][j])
        new_sA[i] = self.sA[i] + deltaTime *sA_d

        sr_dd = self.ar *((self.ar / 4) *(self.tR[i] -self.sr[i]) -self.sr_d[i])
        new_sr_d[i] = self.sr_d[i] + deltaTime *sr_dd
        new_sr[i] = self.sr[i] + deltaTime *self.sr_d[i]

        sx_dd = self.ax *((self.ax / 4) *(self.tX[i] -self.sx[i]) -self.sx_d[i])
        new_sx_d[i] = self.sx_d[i] + deltaTime *sx_dd
        new_sx[i] = self.sx[i] + deltaTime *self.sx_d[i]

    for i in range(self.N):
        self.sA[i] = new_sA[i]
        self.sr[i] = new_sr[i]
        self.sx[i] = new_sx[i]
        self.sr_d[i] = new_sr_d[i]
        self.sx_d[i] = new_sx_d[i]
        output[i] = self.sx[i] + self.sr[i] *sin(self.sA[i] + self.initialPhases[i])

    return output
```

Code A.2: Genetic Algorithm setup for resonance parameters.

```
def __init__(self, generations, gen_time, population_size, callback_generation, callback_fitness,
    initial_genes = [0.25, 15*pi/180, 20*pi/180, -pi/2, -pi/2]):
    global time_per_gen, cb_generation, cb_fitness

    cb_generation = callback_generation
    cb_fitness = callback_fitness

    num_generations = generations
    num_parents_mating = population_size // 10

    population = np.array([])
    gene_space = [{'low': 0, 'high': 2}, {'low': 0, 'high': pi/4}, {'low': -pi/4, 'high': pi/4}, {'low': -pi/2,
        'high': pi/2}, {'low': -pi/2, 'high': pi/2}]
    for _ in range(population_size):
        population = np.concatenate((population, np.array(
            initial_genes)))

    population = population.reshape(population_size, len(initial_genes))

    parent_selection_type = "rank"
    keep_parents = -1
    crossover_type = "uniform"
    mutation_type = "random"
    mutation_probability = 0.2

    self.ga = None
    self.ga_instance = pygad.GA(num_generations=num_generations,
                                num_parents_mating=num_parents_mating,
                                fitness_func=fitness_func,
                                initial_population=population,
                                parent_selection_type=parent_selection_type,
                                keep_parents=keep_parents,
                                crossover_type=crossover_type,
                                mutation_type=mutation_type,
                                mutation_probability=mutation_probability,
                                on_generation=generation_func,
                                delay_after_gen=gen_time,
                                gene_space=gene_space)
```


UAM
UNIVERSIDAD AUTONOMA
DE MADRID