Hybrid Bio-inspired architecture for walking Robots through Central Pattern Generators using Open Source FPGAs

Author: Julián Caro Linares

Director: Antonio Barrientos

Date: February, 2020

INDUSTRIALES
ETSII | UPM

FPGA wars

POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES
UNIVERSIDAD POLITÉCNICA DE MADRID

José Gutiérrez Abascal, 2. 28006 Madrid
Tel.: 91 336 3060
info.industriales@upm.es

www.industriales.upm.es

Julián Caro Linares

05 TRABAJO FIN DE MASTER

INDUSTRIALES

TRABAJO FIN DE MASTER

HYBRID BIOINSPIRED ARCHITECTURE FOR WALKING ROBOTS THROUGH CENTRAL PATTERN GENERATORS USING OPEN SOURCE FPGAS

TRABAJO FIN DE MASTER
PARA LA OBTENCIÓN DEL
TÍTULO DE **MASTER EN AUTOMÁTICA Y ROBÓTICA**

FEBRERO **2020**

Julián Caro Linares

DIRECTOR DEL TRABAJO FIN DE MASTER:
Antonio Barrientos

INDUSTRIALES
ETSII | UPM

POLITÉCNICA

Abstract

In this Master Thesis we present an alternative robotics control approach inspired in animal nervous systems like the human one. The architecture implements the binomial *Brain* (*CNS*) - *Peripheral nervous system* (*PNS*) combining the use of *microprocessors* as the *high-level* control or brain, and the use of *FPGAs* as the *low-level* control or nervous system. Thanks to the *Open-Source FPGA* tools developed in the last years, we are able to use them in the robotics field in new ways that were impossible before.

In this work, we are going to demonstrate that our hybrid bio-inspired architecture, is not only capable of controlling robots using a bio-inspired approach, but it also can go further and create robots with the ability of changing their own nervous system, designing, building and uploading the circuits needed for the *low-level* system in real time, on demand, and without human intervention.

Acknowledgments

First of all, I would like to thank my professor and tutor of this master thesis, Antonio Barrientos, for the opportunity of doing the thesis that I wanted to do. Thanks to his infinite patience, belief in my work, dedication, encouragement, and level of demand. It has been an honor to create this work with a professor like him. Thanks a lot.

To the *FPGA Wars* community, especially Juan González Gómez, thanks to his invaluable and continuous support, enthusiasm, inspiration, and help. Thanks for teaching me day by day to love and enjoy the things that I do. A big part of this work is yours. Also thanks to Jesús Arroyo, for his incredible work developing some of the amazing tools used in this work, and to Eladio Delgado, for designing the heart of this project: the Icezum Alhambra.

To the Maker movement, especially in Spain, for their infinite help, support, and encouragement. This project would have been three times harder, and ten times more boring without you. You are amazing. This project is yours.

Grazie mille a i miei amici Nuno, Maria, Ilaria e Charlotte che dopo tanti anni siete ancora qui, mostrandomi che il mondo è grande, diverso, meraviglioso e, sopratutto, migliore con voi. Vi voglio bene.

To colleagues and extraordinary friends like Roberto Barahona, Beatriz Fernandez, David Estevez, Pablo Brazell, Sergi Consul, Enric Mayas, Luis Diaz Fernandez and especially to Sara Alvarellos. They support, love and advise me day by day.
Thanks for keeping me sane.

To my family, for everything. With them even the impossible is possible.

I love you

Summary

Every time that we stuck in the progress of a field, humanity has always come back to its roots: nature. In this master thesis we present a Robotics Hybrid bio-inspired architecture with the aim of creating robots inspired in nature controls, like the human one. Robots with a brain, and more importantly, a whole nervous system.

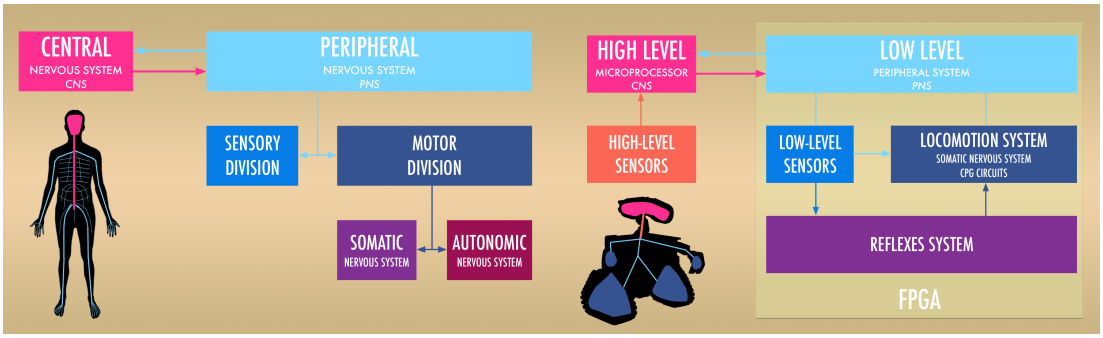


Figure 1: Human Nervous system and the proposed bio-inspired robotics architecture.

The proposed bio-inspired control architecture imitates the common nervous system of some animals and humans. The human control is divided in two main parts: The *Central Nervous system (CNS)*, formed by the brain and the spynal cord, and the *Peripheral nervous system (PNS)*, distributed along the whole body. The idea is to imitate this natural architecture creating a *high-level* system, or a brain, responsible of the high level and more abstract orders, like tasks planning, machine learning, advanced sensors and, in general, complex tasks that require a lot of computer power. The *low-level* system, or *Peripheral Nervous System*, is responsible of the most specific orders, like motors control, basic sensors, and even reflexes actions. In general, the *low-level* system will be responsible of all the repetitive, simple and concise actions, freeing up computer power in the *high-level* system to be focused in more relevant tasks.

To create this architecture, the *high-level* system is implemented inside a microprocessor and the *low-level* inside an *Open Source FPGA*. The idea is to bring together the best of two different worlds: the software created in the *high-level*, and the hardware created in the *low-level*. The microprocessor, the brain, is an excellent general-purpose machine capable of doing complex tasks thought the creation of programs. However, and despite the use of threads and multiple cores, it is not capable of doing real parallelization, losing performance when there are a lot of tasks working at the same time.

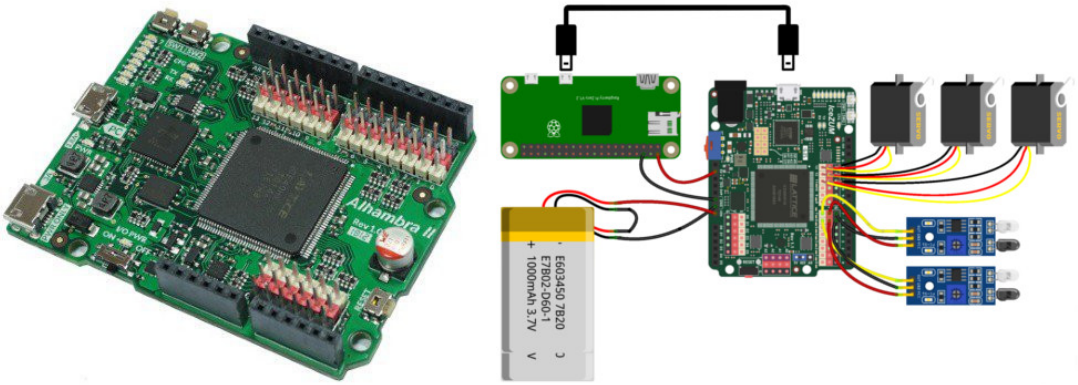


Figure 2: The Icezum Alhambra Open Source FPGA and the hardware implementation of the proposed architecture on one of the robots developed in this work: *Doodle*.

The *low-level* or nervous system, on the other hand, is based in digital circuits implemented inside an *Open Source FPGA*. Although the design of hardware requires in general more effort than software, there are very interesting advantages. The circuits created inside an *FPGA* are *real* circuits with an impressive speed performance compared with their equivalent in software. Moreover, the problem of parallezation in hardware is just trivial. All the circuits or different *low-level* tasks are executed in parallel, being capable of adding all the new circuits that we want until reach the maximum size of the *FPGA* without losing performance. The *Open Source FPGAs*, like the *Icezum Alhambra* used in this work, represents an interesting new approach in the robotics world. For the first time we can use *FPGAs* to do things that were not allowed by the manufacturer, like the robots of this work, where the *high-level* system can create, verify, sinthetize and upload new circuits for the *low-level* system in real time and without human intervention. Having robots that are able to change their own nervous system to adapt it to new tasks and situations.

To experiment and develop the proposed architecture, different robots and experiments have been done. But there are two of them that have been crucial for the development of the architecture: the robots *Doodle* and *Crabdle*.

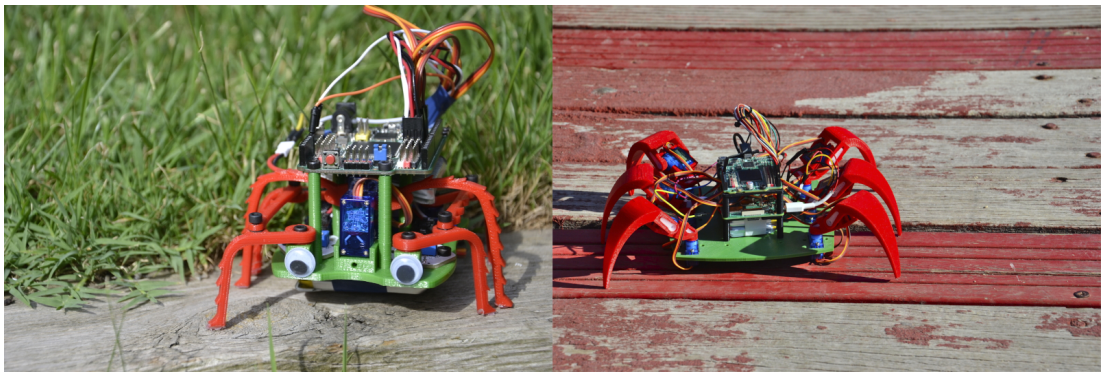


Figure 3: Doodle (left) and Crabdle (right). Two robots developed from scratch that use a Raspberry Pi and an Open Source FPGA to implement the proposed architecture.

Doodle is a robot created from scratch with the purpose of testing the full bio-inspired architecture of this work avoiding most of the mechanical, electronic, design, and locomotion problems that more complex robots have. *Doodle* is a relatively simple and small hexapod with only three degrees of freedom and two small infrared sensors to follow lines and avoid falls. But despite its simplicity *Doodle* has all the fundamental elements of the control architecture. *Doodle* has been designed to be an educational robot, easy to build, and as cheap as possible. With these features, *Doodle* is perfect for focusing on the development of the tools and technologies needed to create the bio-inspired control architecture avoiding the most common and complex technical problems that we have to face in most advanced robots.

Crabdle, on the other hand, was designed from scratch as a result of the learned lessons developing the bio-inspired architecture in *Doodle* and other robots. *Crabdle* is a bigger modular hexapod with twelve degrees of freedom and a complex locomotion, a more complex power management and design, and the fully bio-inspired architecture implemented using a *Raspberry Pi* as the brain or high-level and the *Icezum Alhambra* as the nervous system of *low-level*. *Crabdle* and *Doodle* use *Central Patter Generators* or *CPG* implemented inside *Open Source FPGAs* to achieve great soft, efficient and continuous movements from their legs. *CPGs* are present in most of the animals and

insects and are a great tool in robotics to achieve better locomotions and more complex movements with less effort than using other traditional control methods.

As we said, the *low-level* system has been developed in the form of circuits inside an *Open Source FPGA*. To describe those circuits, the program called *Icestudio* has been used. *Icestudio* it is an *Open Source* program that combines graphical blocks and wires with *Verilog* code, being perfect for prototyping and testing the *low-level* system.

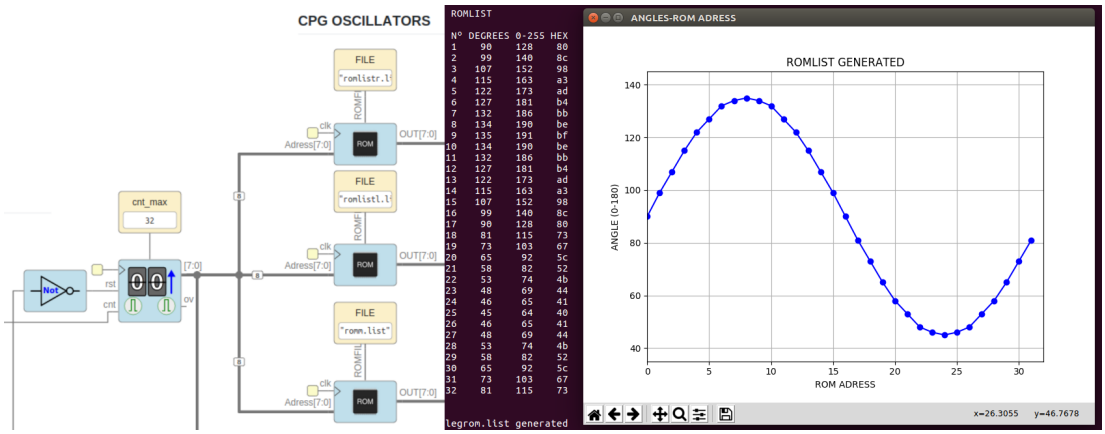


Figure 4: CPGs implementation in Icestudio and CPG sinusoidal signals automatically generated by the high-level system for the movement of the motors.

To create the *CPG* generators inside the *low-level* hardware different lookup tables in combination with *RAM/ROM* memories have been used. Once the hardware modules have been built and fully tested using *Icestudio*, they have been ported to normal *Verilog* code. Inside the computer of the *high-level* system, a program called *Python says* has been created. This program is able to automatically generate new circuits for the *low-level* and upload them to the *FPGA* in almost real time, being able to modify the whole *low-level* nervous system based on the needs of each situation and robot.

The *high-level* system, responsible of the most abstract orders and tasks, like task and path planning, computer vision or complex sensorization can talk with the *low-level* system to interchange information and high level orders. With this information the *high-level* system can decide to remove the circuits related with a broken leg, or describe new *CPG* signals for changing the locomotion of the robot. It can also upload to the nervous system pre-designed circuits for specific tasks or download the circuits from a hardware repository from the cloud. All of this can be done without human

intervention having robots that can change their own body nervous system thanks to the *Open Source FPGA* tools.

Thanks to this architecture, it is possible to create bio-inspired robots that combines the best of the software and hardware world. While the traditional *high-level CPU* is very powerful and requires relative little effort to implement new features, is not great executing tasks in real parallelization, losing performance in robots with complex tasks and a lot of sensors and actuators. The proposed *low-level* system, on the other hand, requires more effort and expertise to implement new tasks describing hardware inside the *Open Source FPGA*, but the execution of multiple tasks in parallel is trivial and the performance of the tasks are far superior from their equivalent in software.

The robots created for this project have been a crutial part in the development of the proposed control architecture. *Doodle* has demonstrated the full viability of the control system, while other little robots and experiments based in the tools developed with *Doodle* have tested the scalibility of the solution. Being able to use the same hardware base in totally diferent robots with little effort in the adaption to a new robot and even implementing this bio-inspired control architecture as a *ROS* module and test it in bigger robots like *Tiago* from *PAL Robotics*. *Crabdle* has been the last robot developed in this work, the purpose of *Crabdle* was to successfully implement the proposed architecture in a more complex robot taking into account the lessons learned during the development of the previous robots and having a great robot base to expand and create new robots based on this work.

The use of the new *Open Source FPGAs* and its tools has been key to create a very interesting *low-level* bio-inspired control, where the flexibility of *CPG* signals allow us to create complex locomotions with a very high performance, experimenting and implementing also reflex actions inside the developed robots.

During the development of this project, part of this master thesis was presented and published as a paper in the *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018).*

KEYWORDS: robotics, bio-inspired, fpga, open-source, hybrid-control
UNESCO: 3304, 330412, 3311, 331101, 331102

Contents

List of Figures

1

Introduction

1.1 Motivation

Robotics is hard. A robot is a combination of a lot of different disciplines, like physics, mechanics, electronics, programming, and design. In the end, a robot has to interact with the real world, that is, despite our efforts to avoid it, a chaotic system difficult to predict.

Some problems that for we humans are easy to solve, like walking, avoid obstacles, climbing or just grab an object or open a door, are a little miracle that involves hundreds of muscles and many sensors. Moreover, it requires a control able to work in perfect coordination to reach a goal that we have learned to achieve through the years of our childhood and adult life.

All of these common tasks involves a complex implementation in a robot that, in the end, is just a machine with a bunch of actuators and sensors that it is excellent resolving math problems: "A washing machine with legs." That is the reason why nowadays we have many robotics competitions oriented to different common tasks. Open doors, climb stairs, or walk on irregular terrain are common trials seen in the Darpa robotics challenge (Fig. 1.1) (1). Other competitions are more urban-oriented with scenarios that involve how to navigate at homes, identify everyday objects and perform human interaction tasks like picking a package from the postman or take coffee orders that are present in challenges like the *European Robotics League* (2) and *Sciroc* (3).

As we can see in these competitions, failure is often the most common result of months, even years, of hard work. Nature terrain is irregular, unexpected obstacles

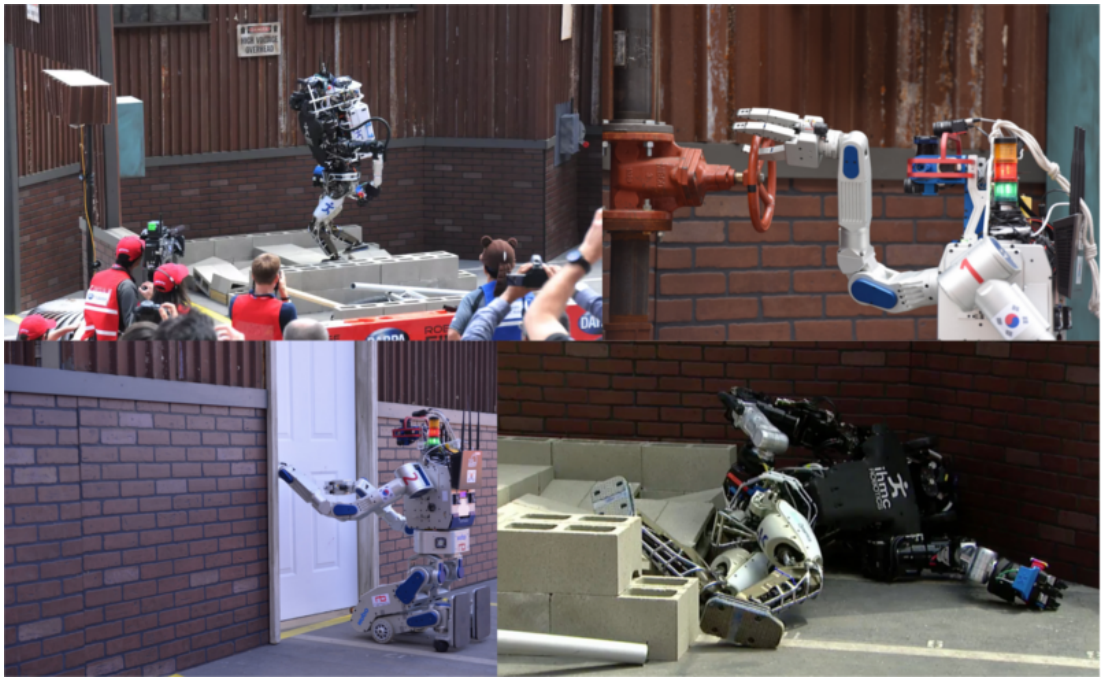


Fig. 1.1: Robots performing the frequent tasks necessary to operate in an industrial disaster zone like Fukushima.

are everywhere, and human-made objects requires hand-eye coordination and a series of complex and coordinated movements that are fairly easy for a child, but requires a team of specialized researchers to achieve in a robot.

That is the reason why some basic locomotion problems have not been totally solved nowadays. If we want to create robots that are able to move in the real world, where there is a combination of natural and human environments, we are going to need legs.

However, leg locomotion, especially human locomotion, is complicated. It requires a perfect knowledge of the dynamics and kinematics of the robot and a very robust control system able to react in real time to the changes in the terrain or the mistakes in the leg positions.

Due to this complexity, one of the best approaches is trying to imitate the animal locomotion creating bio-inspired robots with the purpose of discovering new ways to solve the problems, and improve future robots based on the observed results. One example of this, is the famous robot *Spotmini* from *Boston Dynamics* (Fig. 1.2) (4).

Fig. 1.2: The famous robot Spotmini from the company Boston Dynamics. This robot has an excellent performance in its movements thanks to the strong inspiration in the quadruped animals. The robot is usually teleoperated.

To achieve this goal is necessary not only to create robots that mimic the shape and mechanisms of an animal or a human, but it is also important to try to imitate and simplify the control architecture and movement behaviours that are present in most of the animal nervous systems. The work of *Auke Ijspeert et al* at the *Biorob laboratory at Laussane* (5) is an inspirational example of how to do a bio-inspired control for a robot (Fig. 1.3).

Fig. 1.3: Robots created at the Biorob laboratory. All the robots are bio-inspired and try to imitate not only the shape and mechanism of the animal but also its control system.

The aim of this work and the paper presented in the conference *IROS 2018* (6), is to explore the field of bio-robotics and the robotics control architectures. We propose an hybrid control architecture that mimics the basic animal nervous system present in animals like the human being, where there is a *high-level* control, responsible of the *high-level* orders, conscious decisions and complex processing of information, and a *low-level* control, responsible of the *low-level* orders like voluntary and involuntary movements, basic sensing and reflex movements (Fig. 1.4).

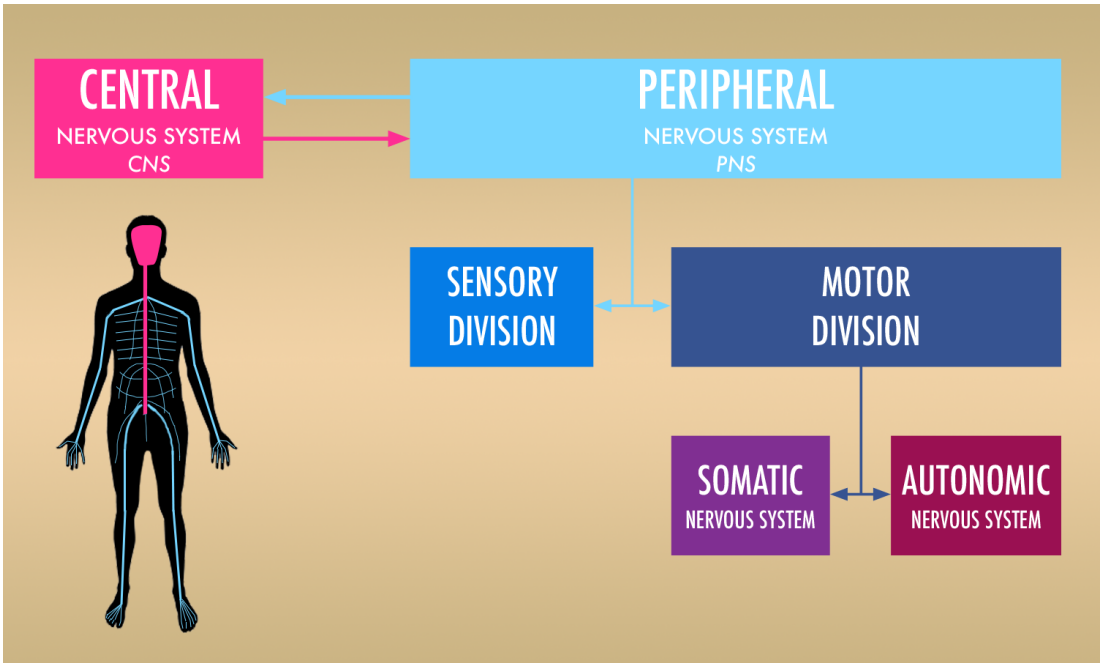


Fig. 1.4: Human nervous system. The *CNS* is responsible of the high-level orders, like *moving the arm* or *walking*. The *PNS* is responsible for the basic human senses and the voluntary and involuntary movements.

To create an approximation of this control structure (Fig. 1.5), we propose the use of a microprocessor or microcomputer as the *high-level* control system or *CNS*. This microprocessor will be responsible for traditional *high-level* orders in robotics, like tasks planning, computer vision processing, external communications, and much more.

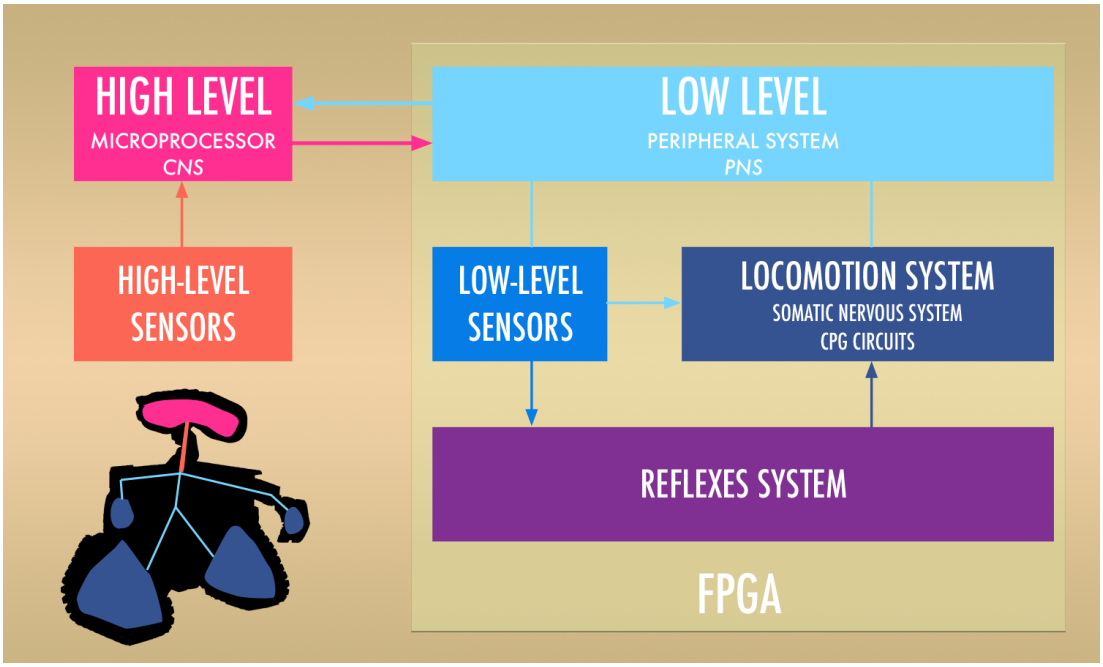


Fig. 1.5: Proposed bio-inspired architecture. The main idea is to use a traditional microprocessor as the *brain* or *CNS*, and an open-source FPGA as the low-level system or *PNS* to liberate the high-level control of repetitive and automated tasks.

In the case of the *low-level* control system, we will implement concepts like the use of *Central Pattern Generators* (*CPG*), and the creation of custom hardware circuits synthesized inside an *open-source FPGA* that are faster than their equivalent in a microprocessor and can work in real parallelization. Thanks to the *open-source FPGAs* tools, the *high-level* system will be capable of reconfiguring the *low-level* circuits without human intervention and in almost real time.

In summary, the main goal of the project is to create the presented hybrid control architecture that will be implemented in different *open-source* bio-inspired robots, including new robots created for this work, with the purpose of discovering the advantages and disadvantages that offers this kind of control systems.

1.2 Objectives and goals

As we said in the last section, the main goal of this project is to implement the proposed hybrid bio-inspired architecture in real robots with the purpose of discovering if this architecture, and the technologies used on it, can be an interesting alternative of the actual robotics control designs and in which circumstances this control will be better. In this section, we are going to enumerate the proposed objectives and goals that the present work should reach:

- **Developing a High level-Low level control architecture:** The idea is to implement the bio-inspired architecture in robots where the *high-level* will be controlled by a microprocessor that sends *high-level* orders to a *low-level* system with a high-speed response. Taken into account this architecture, the *low-level* system can be implemented creating basic circuits inside an *FPGA* for locomotion, basic sensing, and much more. Thanks to this control, the system will free up processing time at the *high-level* that can be used for new tasks.

- **Bio-inspired solutions:** Due to the strong inspiration of the proposed control system in the animal nervous system, it seems necessary to investigate the actual state of the art of the bio-robotics field.

- **FPGA solutions in robotics:** With the use of *open-source FPGAs* for the low level, there is the necessity of making a study of the state of the art in the use of *FPGAs* in the robotics field, taking special attention in the use, advantages and differences found in the use of open-source *FPGAs* tools against classic *FPGAs*.

- **Justification and possible uses of the proposed control:** It is necessary to analyze the proposed architecture to discover the pros and cons of this control against other robotics architectures, and conclude the cases and under which circumstances this control can be an advantage in the development of a robot.

- **Implementation of the architecture in different robots:** To research the utility of the proposed control architecture, it will be implemented in different robots and small mechanisms. A simple bio-inspired robot will be created to demonstrate that all the technologies and structures that are present in this work can be implemented into more complex robots.

- **Implementation of *CPGs*:** It will be necessary to experiment with *Central Pattern Generators* using this architecture to control the locomotion of the different robots. The purpose is creating a more like animal locomotion and compare this bio-inspired locomotion with others.

- **Experiments with other bio-inspired mechanisms:** Another goal will be exploring other bio-inspired mechanisms, like reflex movements, to see if they can be implemented in different robots.

- **Creation of a final robot prototype:** To demonstrate all the work that has been done, a final robot prototype should be created. This robot should have a bio-inspired modular and scalable locomotion. The objective with the creation of this robot is to demonstrate how the use of the proposed hybrid control architecture can be a serious advantage for robots where the use of multiples degrees of freedom or sensorization, working together and at the same time, can be a critical point in their performance.

- **Improvement of locomotion and other systems thought Machine learning:** It would also be desirable to experiment with machine learning techniques that improve the circuits, *CPGs*, and locomotion of the robot. To do this, the *high-level* system can implement machine learning techniques that modify the circuits of the *low-level* and their *CPGs* and, therefore, create robots that are able to modify their own control and *nervous system*, learning about the new situation in almost real time.

1.3 Structure of the document

The structure of the document is the following:

First, we are going to talk about the state of the art of the different fields that this master's thesis is about. The state of the art will talk about the bio-inspired robots and their different uses and applications, the use of the *Central Pattern Generators* or *CPGs* in bio-inspired robots, and the actual use of *FPGAs* in the robotics field.

After the state of the art, we will talk about the different technologies used in this work, putting special attention in the technologies related with the *low-level* system of the proposed architecture: the *HDL* languages, and the new *Open-Source FPGAs*, talking about their features, tools, actual state, and the reasons to use them in robotics applications.

Once we have talked about the used technologies, the proposed bio-inspired control system of this master thesis is presented. We will talk about the animal nervous system and their different parts, and how this structure has served as inspiration to develop the bio-inspired control architecture, talking about its two mains parts and how they work: the *high-level* system and the *low-level* system, and some tips and advice to correctly understand and implement the architecture.

After explaining the proposed bio-inspired system, the different robots and other machines created to experiment with the architecture are explained. We will talk about *Doodle*, a small robot created to see if it is possible to really implement or not the architecture. As well as other robots and experiments like *Randofo*, mechatronic eyes, or even a *ROS* module developed for using the bio-inspired architecture inside robots that use the *ROS* framework.

As a final project, we will talk about *Crabdle*, a relative complex *hexapod* robot created to implement the whole bio-inspired architecture. We will talk about their purpose, hardware and *CAD* design and, more importantly, the implementation of the *high-level* and *low-level* system in the robot, how works the communication between them, and how the *high-level* is able of creating new circuits and upload them in real time, and without human intervention, in the *low-level* system.

The document will finish with the conclusions and results obtained after the development of this project, talking about the advantages and disadvantages of the proposed bio-inspired robot architecture, and the future work to expand it.

2

State of the art

In this chapter, we will introduce the state of the art that has been served as the base for this work. We will talk about the state of the art of the bio-inspired robots, the use of *CPGs* in robotics and its relationship with the research field of biology, and how the *FPGAs* have been implemented so far in different robots. These three sections will be essential in the development of the proposed architecture of this project.

2.1 Robots inspired by nature

The field of bio-robotics is just a direct consequence of the way the human being thinks and acts. At the beginning of the 21st century, where we can use centuries of studies, knowledge, and wisdom, there are many problems that we are not able to solve yet. Moreover, those problems are holding back our progress. Every time that we stuck in a field of study, humankind has always come back to its roots: nature. Every past and actual species are the consequence of thousands of years, sometimes millions, of natural selection and mutations, adapting their bodies to each problem they find in their environment. Therefore, it seems logical to take inspiration from nature to solve the problems that robotics is facing nowadays.

The bio-robotics is the field of knowledge that studies the build of biologically inspired robots. Covering topics like the design, materials, mechanics, locomotion, actuators and sensors, processing of signals and control systems.

2.1.1 Robots that mimic big animals

Nowadays, there are a lot of great robots and projects inspired by real animals. The most common are the robots that try to mimic big quadrupeds like dogs, pumas, and other mammals. That is the case of *Spotmini* (4) and its big brother *Alpha Dog* (7), by *Boston Dynamics* (Fig. 2.1).

Fig. 2.1: Spotmini (left) and its big brother Alphadog (right). Both robots have an impressive locomotion inspired in big quadrupeds.

Both robots are inspired by medium and big animals. *Alphadog* was created as a workhorse or mule for the USA Army. It uses a gasoline engine and can load hundreds of kilograms. *Spotmini*, on the other hand, was designed more like a dog and is totally electric. Its legs have inverted knees for a great front and lateral movement and locomotion. These robots have a profound work study about the mechanisms and phases of the real locomotion in quadrupeds and are a great example of how this kind of robots will be used soon for many tasks.

Another interesting example of bio-inspired robots based on big animals is the ostrich-like planar locomotion robot from the *IHMC Institute* (8) (Fig. 2.2). This robot can reach velocities of 20 Km/h with only one engine. The interesting thing about this design is that it does not need a complex control system to run at these velocities. All the stability in the locomotion is thanks to the inspiration in the locomotion mechanics of the real Ostrich. This idea, the use of a very good mechanical design inspired by real animals to avoid complex controls, is one of the most important contributions that bio-robotics has to offer to the robotics field.

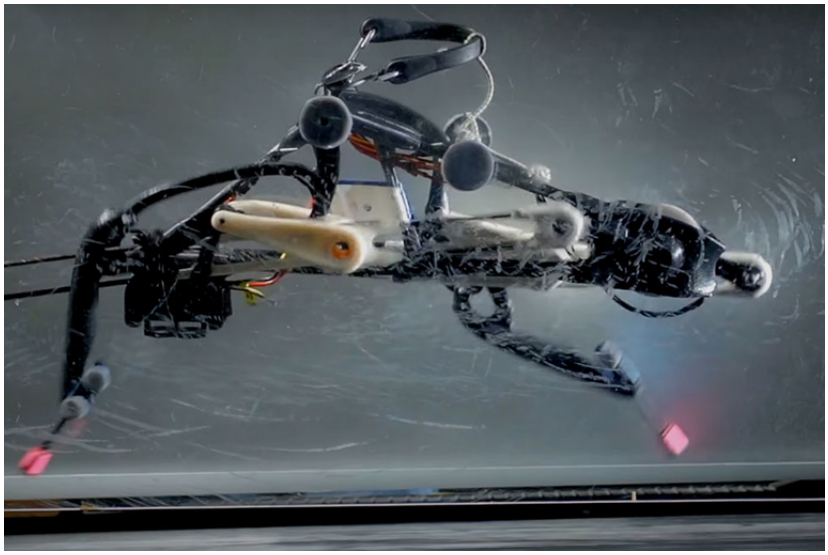


Fig. 2.2: A bio-inspired Ostrich robot that can reach high velocities with only one motor and without a sophisticated control thanks to its mechanic design.

We not only have robots inspired by animals, but we also have humanoid robots that are trying to improve their results using natural tricks. At the moment of the writing of this document, the most famous, and maybe, the most advanced, it is the robot *Atlas* by*Boston Dynamics* (9). This robot has a strong inspiration in nature to create the legs, arm, and chest of its design. The robot can walk, run, pick boxes and do some impressive tricks like static jumps and even backflips!

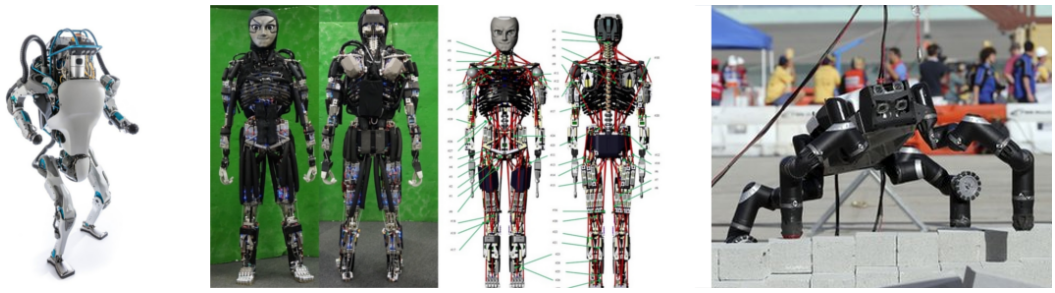


Fig. 2.3: Robot Atlas (left), Kegoro robot, inspired in the human skeleton and able to sweat (center), and Nasa's RoboSimian (right).

However, there are other humanoid robots that take advantage of the natural world (Fig. 2.3). Being a humanoid robot does not mean that you always have to walk using two legs. Primates in nature are a great example of how to use two or four extremities

depending on the context. In this sense, the primate inspired *RoboSimian* from *NASA Jet Propulsion Laboratory* (10) is an excellent example of how to take advantage of the shape of an animal to increase the performance in environments like natural disasters. Another great example of the lateral thinking that the bio-inspiration can offer is the humanoid robot *Kengoro* (11), that is able to sweat in a similar way a human does. This ability allows the robot to do more heavy tasks like more push-ups thanks to a better refrigeration.

2.1.2 When materials are key- Soft robots

In a world where most of the organic living forms, obviously without their shells or other protections covers, are soft and easy to harm, it seems ridiculous to try to create animal robots using only rigid materials like aluminium or rigid plastic like *ABS*. Soft robots are the future of robotics and, like always, materials will accelerate or brake the future that we will see.

Soft robots are complicated. A soft actuator can have, virtually, infinite degrees of freedom. The rules of locomotion, kinematics, and mechanics of the traditional robotic field cannot always be translated with this strange new world where even the most simple actuator can have a totally different behaviour depending on the temperature or direction of movement. Moreover, implant sensors, to measure positions, for example, are sometimes complicated due to the flexible nature of the robot. There is a lack of sensors for soft robots, and that implies that most of the robots are controlled in an open loop. Fabrication of soft materials can also be tricky. That is the reason why the *soft robotics toolkit* has been created (12) with the idea of incentivizing and making growth a soft robots community.

There are dozens of great examples of the state of the art in soft robotics. The most important, or at least iconic, is the called *Untethered robot* from *Harvard University* (13). This robot is one of the first soft robots that does not need external power or pneumatic tube. The robot can be trampled by a car, even burned. The only problem: the electronics, including batteries, are not flexible (Fig. 2.4).

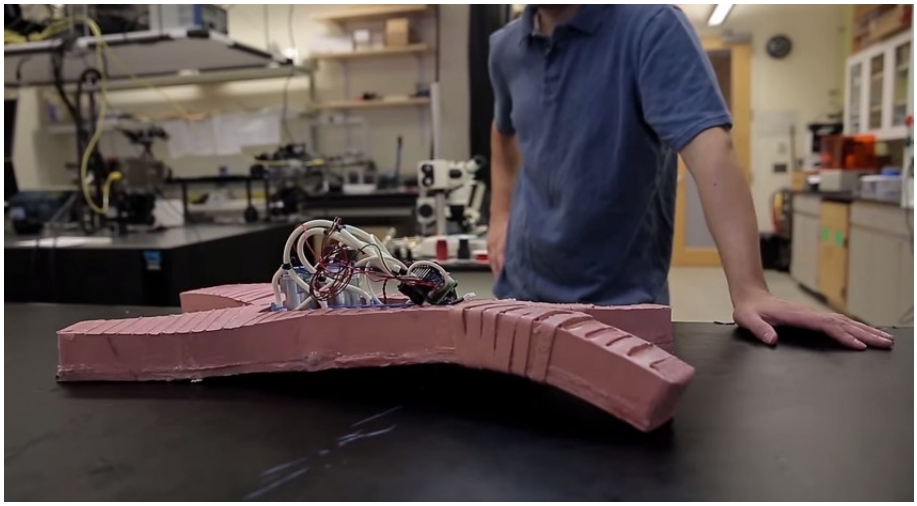


Fig. 2.4: Most of the actual soft robots use pneumatic tubes to work. This one does not need an external cord to work. A small but crucial step in its field.

The evolution of this untethered robot are robots like *Octobot* (14). This small robot is made using a combination of 3D printing soft lithography and molding techniques where even its electronics are flexible. A chemical reaction powers the robot.

Fig. 2.5: Octobot. An independent small octopus robot that is totally made using soft materials and without conventional electronic components.

Another great example are robots for underwater environments. This kind of robots are normally octopus, or jellyfish and fishes. The robotic jellyfish *Aquajelly* (15), and the fish *iTuna* created by *Claudio Rossi et al* (16), are good examples of this kind of robots (Fig. 2.6).

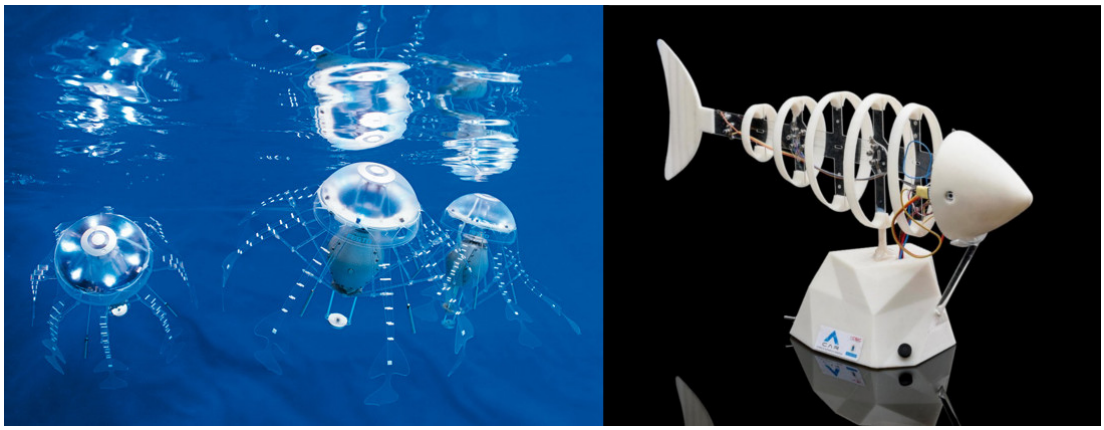


Fig. 2.6: Aquajelly fish and iTuna. The iTuna robot uses SMA actuators to mimic the real swimming oscillations in the body of a fish.

But soft robots can also do more things like walk or swimming. Soft robots can also have the impressive ability of flying. That is the case of some of the amazing robots of *Festo*, like the *BionicFlyingFox* (17), or the *Bat robot* created by *Alireza Ramezani et al* (18) (Fig. 2.7).

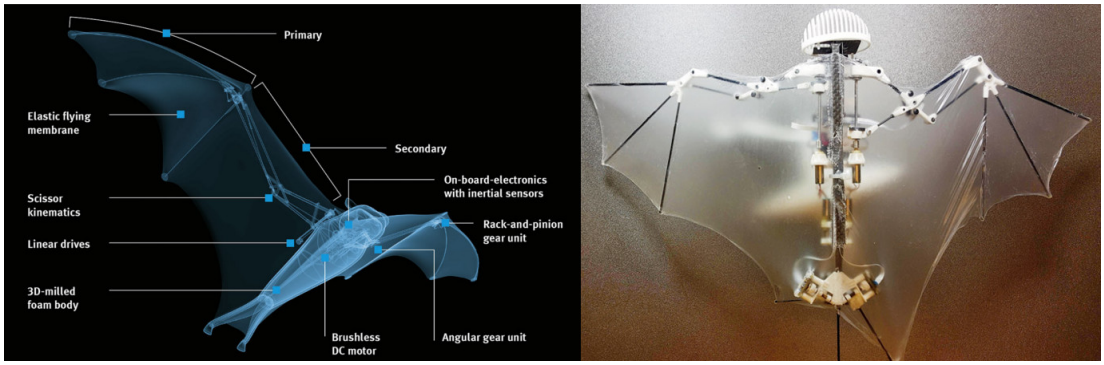


Fig. 2.7: Festo flying fox (left), and Bat bot (right). Two robots that mixed soft and rigid light materials to achieve the impressive challenge of flying.

2.1.3 They will conquer the world - Insect robots and Arthropods

In an amazing world with extraordinary and diverse nature, arthropods, including insects, represents more than the 80% of the living species. They have numerous articulated legs that allow different kinds of complex locomotions and a lot of fascinating characteristics that can be useful in a bio-inspired robot.

Arthropods are everywhere on planet Earth. They have reached a high level of specialization. The study of these animals has an incalculable value to the fields of robotics and biology and represents the approximation of two different fields of study that, in some years, will be more closer than we thought.

However, robotics inspired in arthropods or insects are difficult to do. The main problem: size. We do not have yet the right technology to do the actuators, sensors, and electronics to fit the real size of an ant, but there are some projects that want to experiment with these interesting swarm animals, like the *BionicAnt* created by *Festo* (19) (Fig. 2.8).

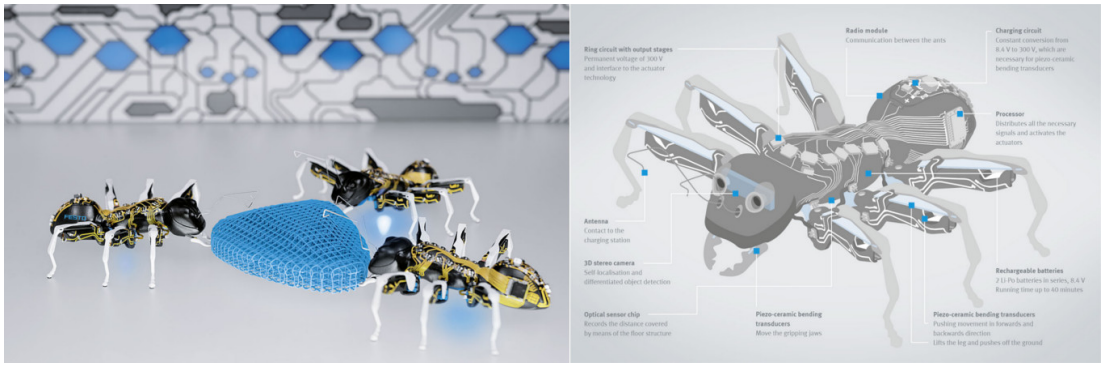


Fig. 2.8: Bionic ants that can collaborate to do some tasks. One impressive thing about these robots is that they have their circuits integrated in 3D in their own shape. Unfortunately, they have the size of a palm.

Other interesting robots are robotic insects that try to fly. As it was mentioned before, a good mechanic bio-inspired design can be better than hundreds of sensors and actuators in a perfect control loop. That is the case of the *Bumper bee* robot (20), a small robot that tries to recreate the extremely elastic wings of the bees. Real bees do not have a perfect fly control to avoid obstacles. Many things can happen when you are in the air, and bees do not care too much about bumping things while flying thanks to their soft body and incredible flexible wings. This represents a paradigm shift in the

control of robots. We are not doing industrial machines anymore. The control does not have to be perfect, it has to be resilient (Fig. 2.9).

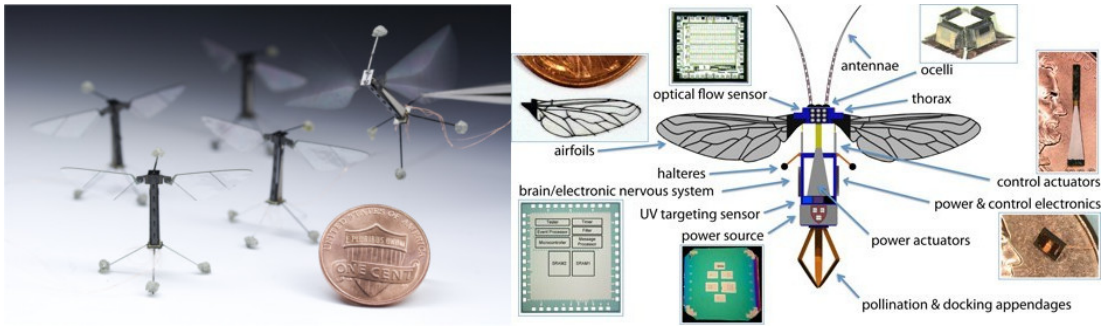


Fig. 2.9: Robobees. A robot that can fly like real bees and bump with objects without damaging its wings.

Hexapods are probably one of the most common bio-inspired robots. A lot of research has been done trying to imitate their decentralized locomotion and create a robot able to change its locomotion depending on the number of legs. There are a lot of great examples. One of the most interesting is *Snapbot* by *Disney Research* (21). The robot is completely modular (Fig. 2.10), and it uses a central body and legs that can have different degrees of freedom using magnetic *pogo* connectors.

Fig. 2.10: Snapbot. An impressive modular hexapod robot that can change its degrees of freedom and locomotion in just seconds.

2.1.4 Beyond the natural constraints

As humans, we are very good doing abstractions with the purpose of simplifying a system and connect different ideas together to create new things and solutions.

In the case of the bio-inspired robots, the word *"inspired"* and not *"mimic"*, means that your robot can imitate a real animal, but it does not have to be exactly the same. It can have new legs, new senses, wings... or just combine one animal with another. Everything that it is within our reach to take advantage of the solutions that we found in nature.

That is the case of the robot *VelociRoACH* (22). A robotic cockroach where *Carlos S.Casarez and Ronald S.Fearing* have added a tail that does not exist in real cockroaches. This tail resolves one of the problems of cockroaches, the difficulty of turn back of their legs after a flip over (Fig. 2.11).

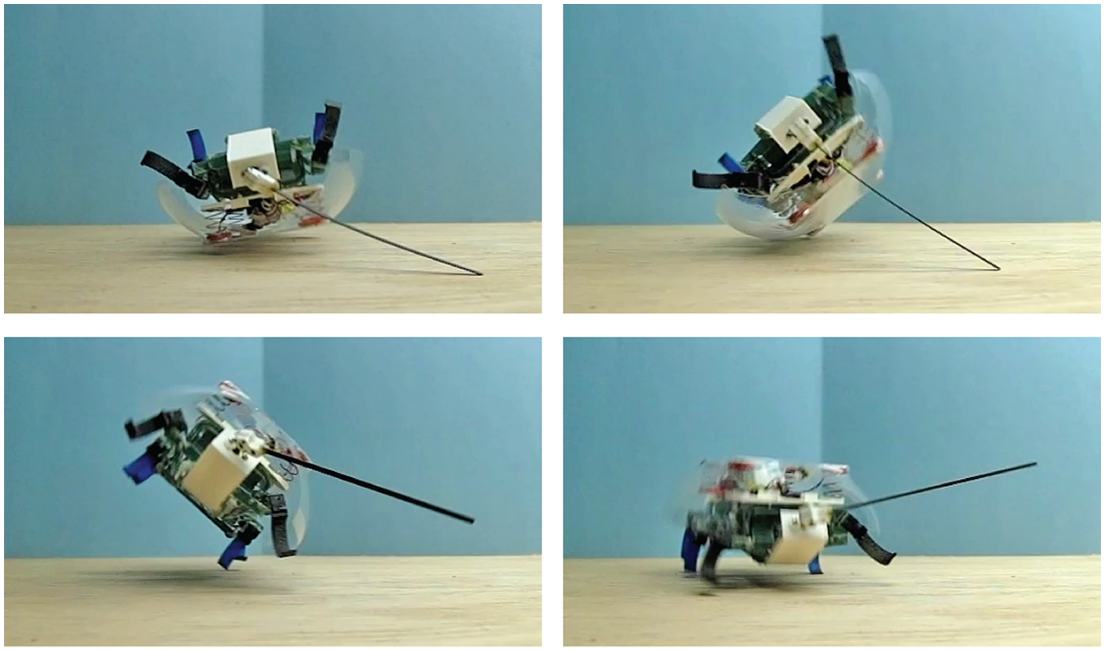


Fig. 2.11: VelociRoACH, a robotic cockroach that can flip their body thanks to an added tail that does not exist in its natural homonym.

Maybe, you prefer to add new abilities to existing animals, or just create devices and mechanism to control them. Taking apart the ethics debate that this kind of projects provokes, there are some works related with the control of animals as robots

using electronics devices that represent the evolution of the well know experiment with frog legs of *Luigi Galvani*.

Some excellent examples are the *RoboRoach*, an educational *DIY* kit to control the movement of a real cockroach thought a wireless antenna (23) or the work done by *H.Sato et al.* (24) to control the flight of a big beetle. Another interesting experiment, and probably less invasive, is related to the manipulation of the behaviour of a swimming turtle to reach a goal (25) (Fig. 2.12).

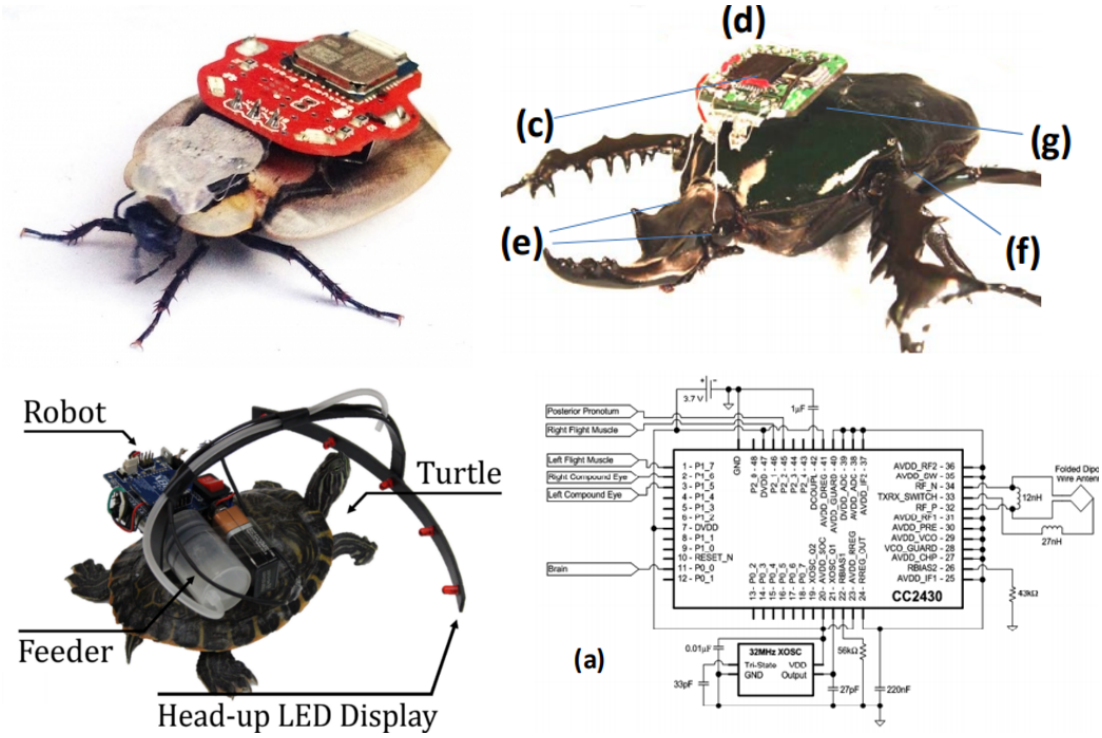


Fig. 2.12: RoboRoach (left), a beetle with its flying control teleoperated using electronics (right up and down) and a turtle controlled using LEDs and a food feeder (left down).

2.1.5 To help and entertain- Social robots

It would seem that bio-inspired robots are not used in a social context where the interactions with humans are essential, but is not true. Humans need social interactions with other humans and other livings forms, but sometimes, those interactions are far from being easy.

Bio-inspired robots can help to solve new problems in this field. One of the most

successful cases is the social robot *Paro* (26). *Paro* is a baby seal able to interact with people with dementia and other diseases or situations. This robot can be used as a great tool in many treatments and works so well thanks to its bio-inspired appearance and behaviour of a baby animal that people generally loves (Fig. 2.13).

Fig. 2.13: Paro robot. A bio-inspired robot used to interact in a medical and social context like, for example, people with dementia.

2.2 Oscillating robots: The Central Pattern Generators (CPG)

The *Central Pattern Generators*, or *CPGs*, are a fascinating subject of study inside the biology field. A *CPG* is a neural circuit able to generate rhythmic signals **without a rhythmic input**. *CPGs* do not process or transform a rhythmic signal; they are the origin of the signal, the central pattern generator.

A formal definition of a *CPG* can be found in the *Encyclopedia of Life Sciences. John Wiley & Sons* (27):

> "Central pattern generators (CPGs) are biological neural circuits that produce rhythmic outputs in the absence of rhythmic input. They are the source of the tightly-coupled patterns of neural activity that drive rhythmic motions like walking, breathing, or chewing."

As the definition says, these analog signals are the origin of a lot of vital actions, like the locomotion in mammals and insects. These oscillators signals are crucial in the execution and automation of repetitive movements, like different kinds of walking, gaits, or other locomotion patterns like swimming.

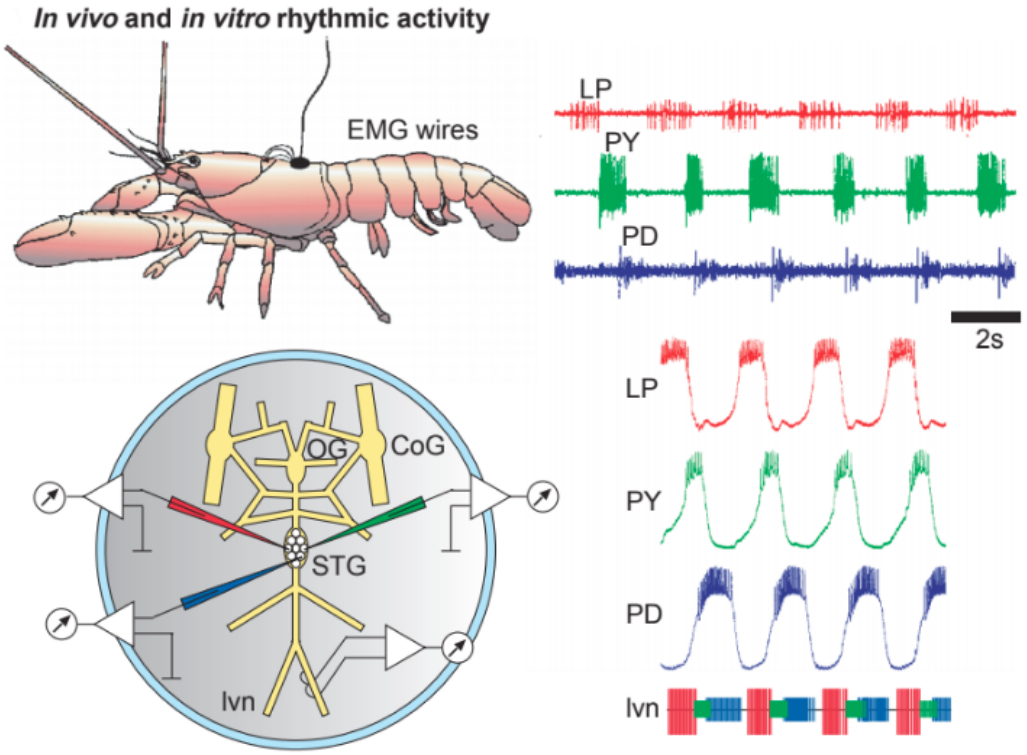


Fig. 2.14: A classic example of the CPGs signals produced in the stomach of a lobster. The recorded signals *in vivo* (up) are very similar to the recorded ganglia nervous system recorded *in vitro* (down) and in the absence of sensory inputs.

But they do not only control these locomotion movements. *CPGs* control also a lot of very important and normally automated activities like breathing, heartbeats, swallows and much more (28) (Fig. 2.14).

Although the own definition of a *CPG* means that a *CPG* does not depend on rhythmic inputs to create its waveforms, *CPGs* usually have inputs and feedback mechanism that modifies the modulation properties of the signal. These properties of the *CPGs* are really interesting and useful in the field of robotics. Thanks to the studies that neurobiologist are doing, we can apply simplified control systems that use *CPGs* as a way to imitate animal movements and achieve better controls and locomotions. A good example is the famous and useful worm *C.elegans*. This little worm of a size of 1 mm, has a very basic digestive track and a small brain. The whole nervous system is formed only by 302 neurons. This very simple nervous system is useful for a lot of scientifical

studies related with diabetes, obesity or even Alzheimer, and can be replicated in a robot using a very basic microcontroller like an *Arduino One* (29) (Fig. 2.15).

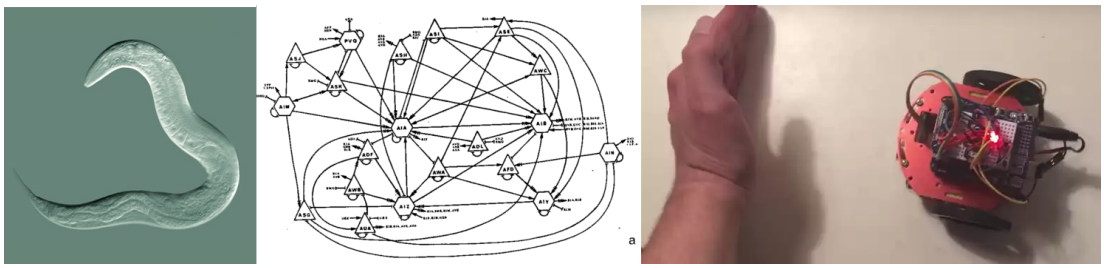


Fig. 2.15: C.elengans worm. Part of its basic nervous system and its equivalent robotic project called Nematoduino.

This collaboration between robotics and biology is very useful in the progress of the two disciplines. Robots like the *Robobee* or the *RoboRoach*, mentioned before (20) (23), imply a lot of laboratory and natural observation of these living animal forms. Sometimes, robots, electronics and *FPGAs*, and artificial *CPGs* can be used as a substitution for an animal form to do research lab. That is the case of the work of *S.Joucla et al* that generates *CPGs* injecting the signals in the isolated spinal cord of a rat (30) (Fig. 2.16).

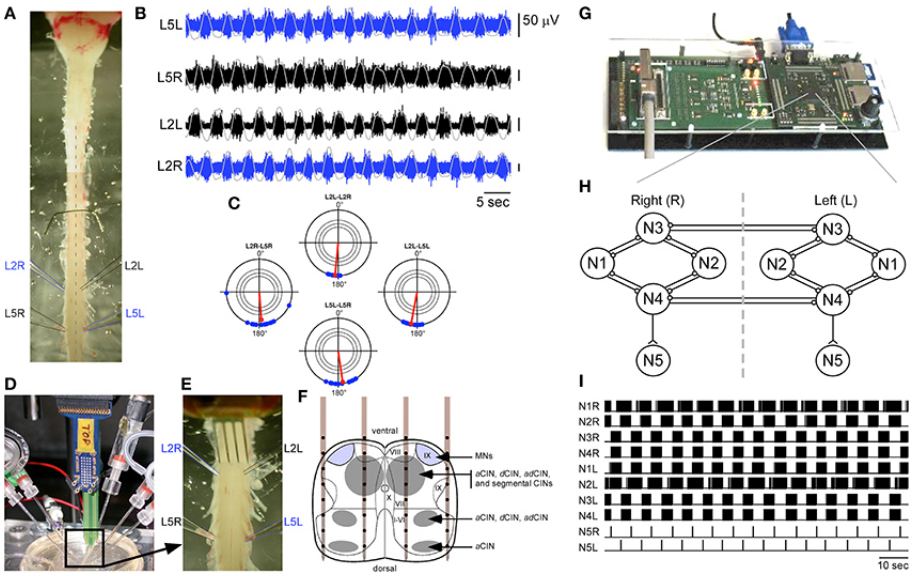


Fig. 2.16: Thanks to the approximation between electronics and biology, researchers can understand better how it works the spinal cord of a rat generating artificial *CPGs*.

2.2.1 Auke Ijspeert and the Biorob lab

One of the most fascinating groups of research about the use of bio-inspired robots with *CPGs* is the *Biorobotics laboratory of Lausanne (Biorob)* (5). This research group is lead by *Auke Ijspeert* (31), and in the last years has been working on exciting projects that have been served as an inspiration base for this work.

2.2.1.1 The Lamprey

Lampreys are a kind of jawless fish that nowadays are considered as a pest in many lakes around the world, especially in North America (Fig. 2.17).

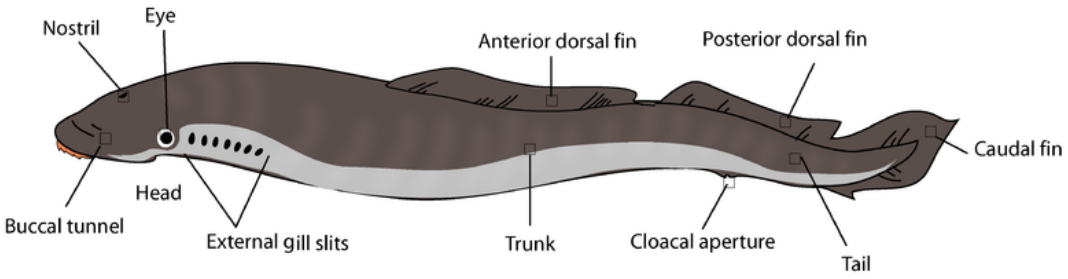


Fig. 2.17: Anatomy of a Lamprey.

Lampreys have elongated bodies and can reach one meter of longitude. There are a lot of different species, but the parasites carnivores that suck the blood of other fishes are the most common. That is the reason why they have an impressive mouth full of teeth.

However, the most interesting thing about Lampreys is their utility in research. Lampreys are very energy-saver swimmers. And this is thanks to the generation of different *CPGs* in their bodies (32). Its nerves system is relatively simple and can be used as a *model organism* to study the principles of motor control in vertebrates and synaptic transmissions. (Fig. 2.18) (33).

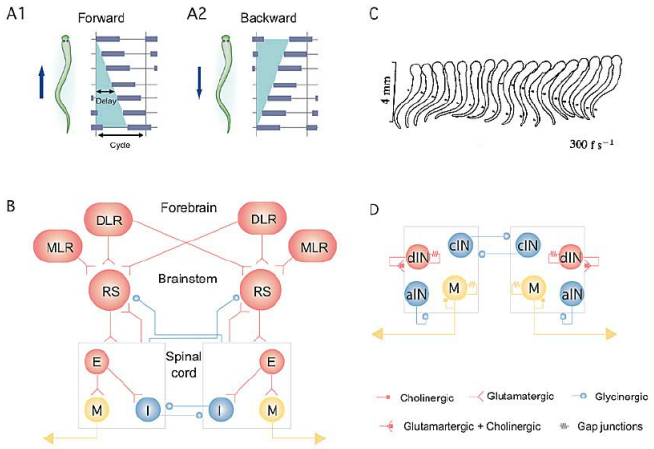


Fig. 2.18: Lamprey nervous system and the CPGs generation.

Ijspeert et Al have a long trajectory studying and creating robots that uses *CPGs*. In the work developed in (34) and (35), *Ijspeert et Al* try to recreate the locomotion of the Lamprey using simulations, genetic algorithms and neural networks. (Fig. 2.19).

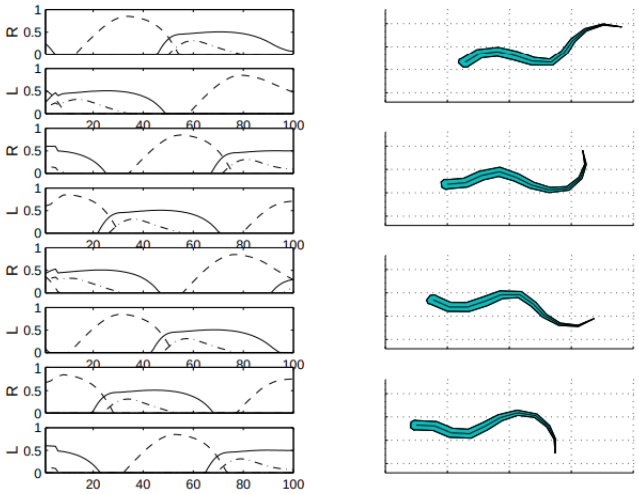


Fig. 2.19: Lamprey CPGs locomotion simulations with genetic algorithms and neural networks.

2.2.1.2 A snake robot able to swim

These previous works were the base to build the *Amphibot* (Fig. 2.20). A modular snake robot able to swimming and crawling where *Ijspeert and Alessandro Crespi* developed a snake-like locomotion system using *CPGs* inspired in the Lamprey (36).

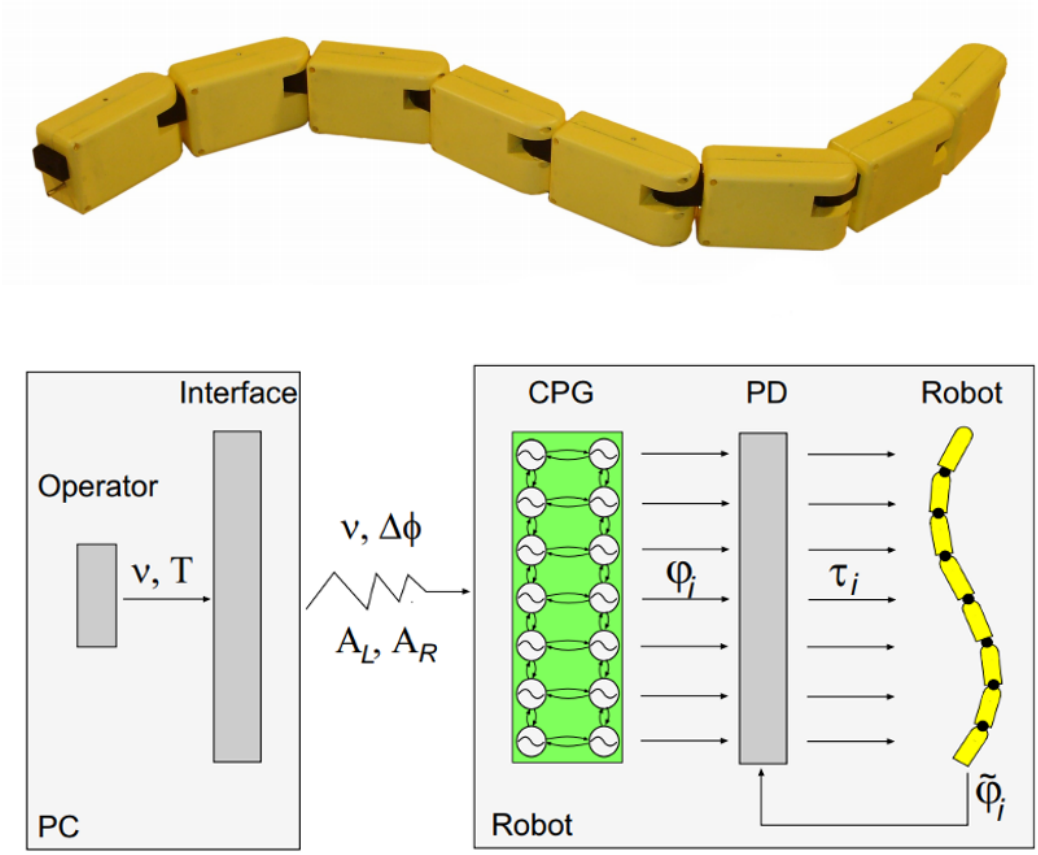


Fig. 2.20: Amphibot version II and their different locomotion controller parts. Each circle with a sine symbol it is a CPG.

The control is based in a *CPG* that uses amplitude-controlled phase oscillators. It has a double chain of oscillators controlled by differential equations that control the amplitude and phase of each chained module. In fig. 2.20 we can see that the robot needs two interfaces. The human interface, that left the operator to control the robot using only two commands that are converted to the four parameters necessary for the *CPGs* equations, and the PD interface, that converts the *CPGs* signals in the appropriate commands for the drivers of the motors.

2.2.1.3 A robotic Salamander

Like in the natural process of evolution, the process of creating and controlling the *Amphibot* robot automatically leads to the next step: legs. After the crawling movement and swimming, adding legs to move better on the terrain seems the most logical idea. Based on the previous work, they created the *"Salamandra Robotica"* (37).

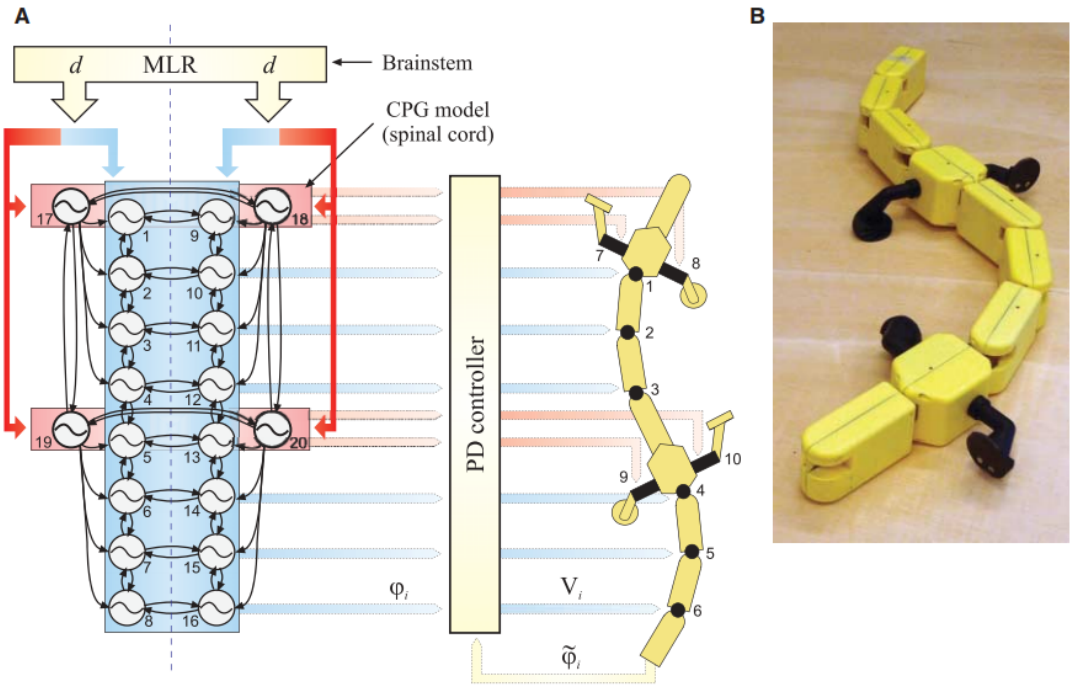


Fig. 2.21: The first version of the "Salamandra robotica". Two pairs of legs have been added to the previous snake robot with their new CPGs.

As it can be seen in fig. 2.21, four new *CPGs* have been added to the control system to command the legs. The most interesting thing about this robot is the incredible ability to switch between swimming to walking with a simple change in the frequency of the oscillations. When the robot is working at low frequencies, legs can fully rotate (the real animal does not have this full rotation in their limbs) and the wave of the oscillations is moving along the body of the robot, just like the videos of locomotion movement in real salamanders recorded for this research (Fig. 2.22).

When we speed up the frequency, the movements of legs *CPGs* begin to be so fast until the legs cannot be moving at the desired velocity, saturating and stopping their oscillations. That is the transition between walking and swimming. With the legs

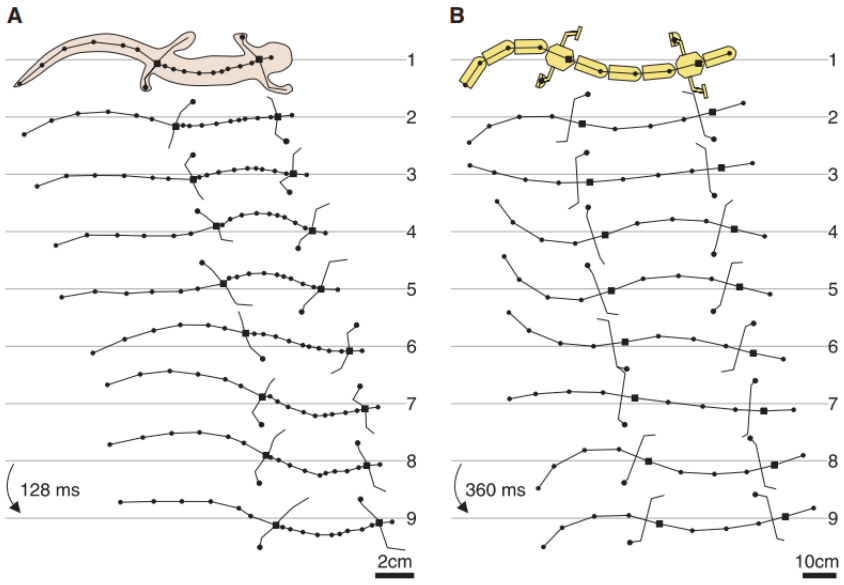


Fig. 2.22: Results of the motion capture of the gait movements in a real salamander and its robotics equivalent. Times in the gait are different, but the way of locomotion are similar enough.

blocked, but the rest of the body still oscillating at high frequencies, the robot can swim without the legs limitations.

This idea of controlling different kinds of gaits for different situations or environments, just changing a couple of parameters thanks to the use of *CPGs*, and a very good design directly inspired in nature, it is a very powerful concept that changes the way we design and control robots.

2.2.1.4 Improving the Salamander: Pleurobot

After all the evolution in research and robots developed in the amphibious branch of the *Biorob lab*, and all the experience gained developing this kind of bio-inspired robots. It seemed clear that it was necessary to create a robot that summarizes all the knowledge acquired. This robot was *Pleurobot* (Fig. 2.23).

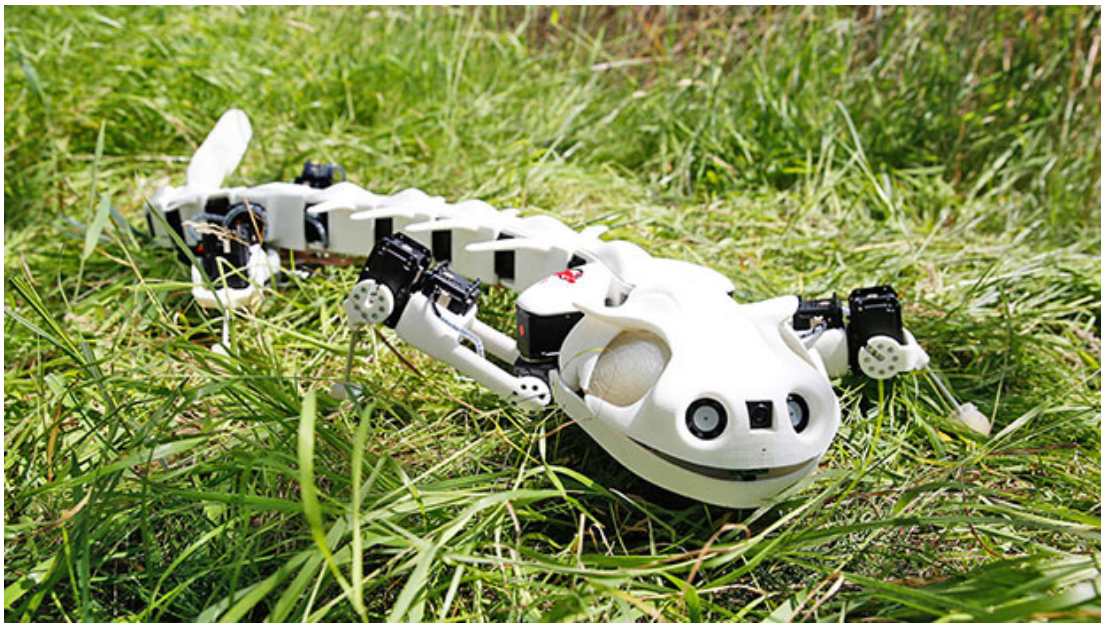


Fig. 2.23: Pleurobot. A polished evolution of all the work done in the Biorob lab in the last years.

Pleurobot (38) is the formal evolution of the job done in the last years in the robotic salamander. The spinal cord of the robot follows the same principles of modularity and control using *CPGs* signals that move along the body of the robot doing oscillating movements. Legs have been improved. Now they do not have the full rotation movement and are made by three degrees of freedom with the addition of an excellent directional pressure sensor at the end of each leg. All the robot, especially the head, has been remade to seem more like a real salamander, adding new sensors in the front-head.

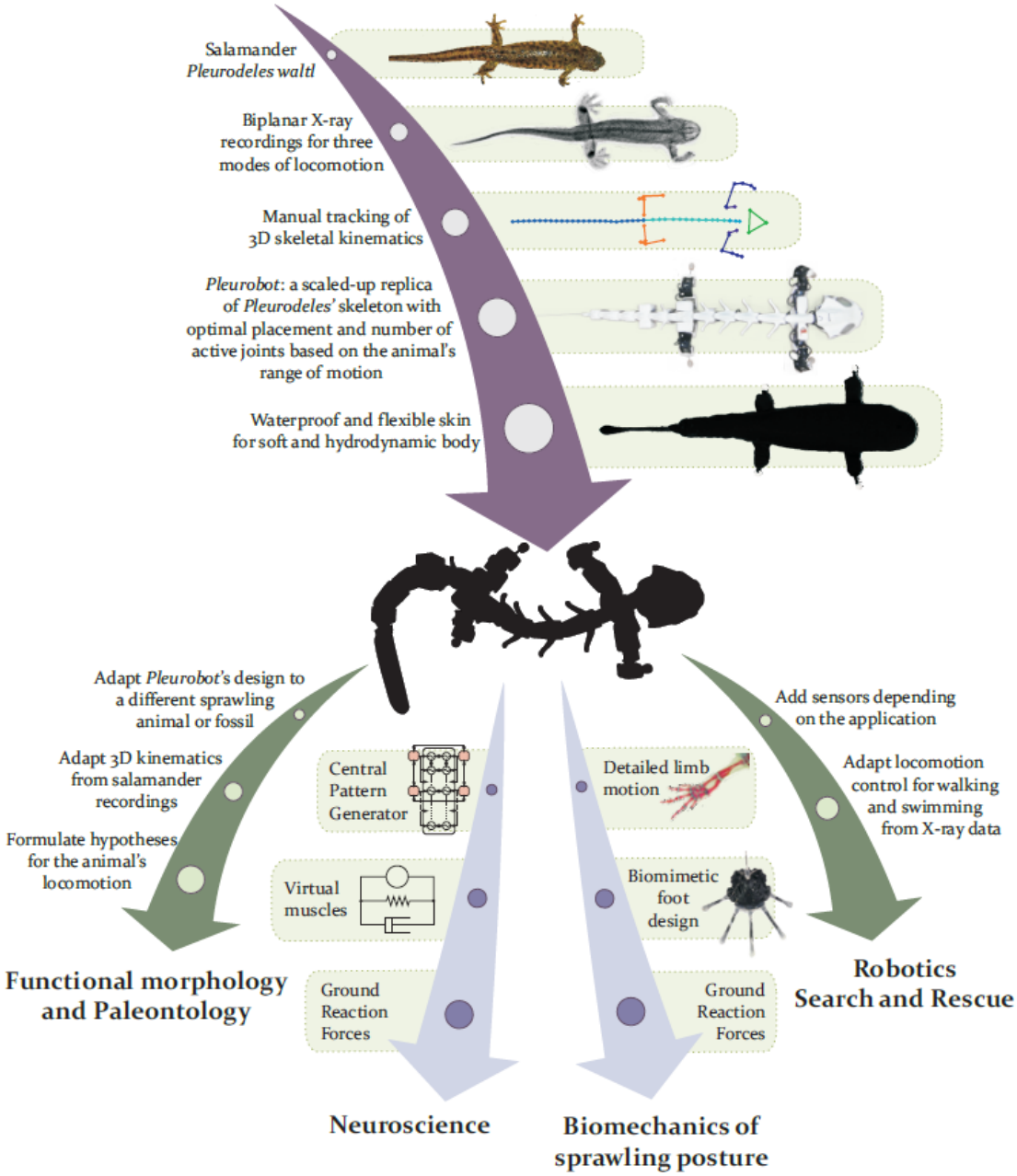


Fig. 2.24: The process of creating a bio-inspired robot developed at the Biorob lab.

However, the most interesting thing about *Pleurobot* is the methodology of work developed through the years at the *Biorob lab* in the *University of Laussane* (Fig. 2.24).

This process is very interesting. The technique of *cineradiography* (38) has been key in the development of a robot that imitates the real mechanism, system control,

and movements of a real salamander. The new data captured with this technique in conjunction with other ones, allows the designer to create a better robot that mimics nature thanks to an excellent mechanism design and a robust system control based in the use of *CPGs*.

Recently and thanks to the use of this methodology of work, they have developed, in collaboration with an interdisciplinary team at *Humboldt-Universität zu Berlin*, the *Orobot*. A research robot focused on the locomotion of the extinct animal *Orobates*. They developed a simulation where is possible to change some parameters like the spine *CPG* scaling (39), and a real robot, based in a modified *Pleurobot* morphology, that mimics what fossils and experts say the extinct animal should be (40) (Fig. 2.25).

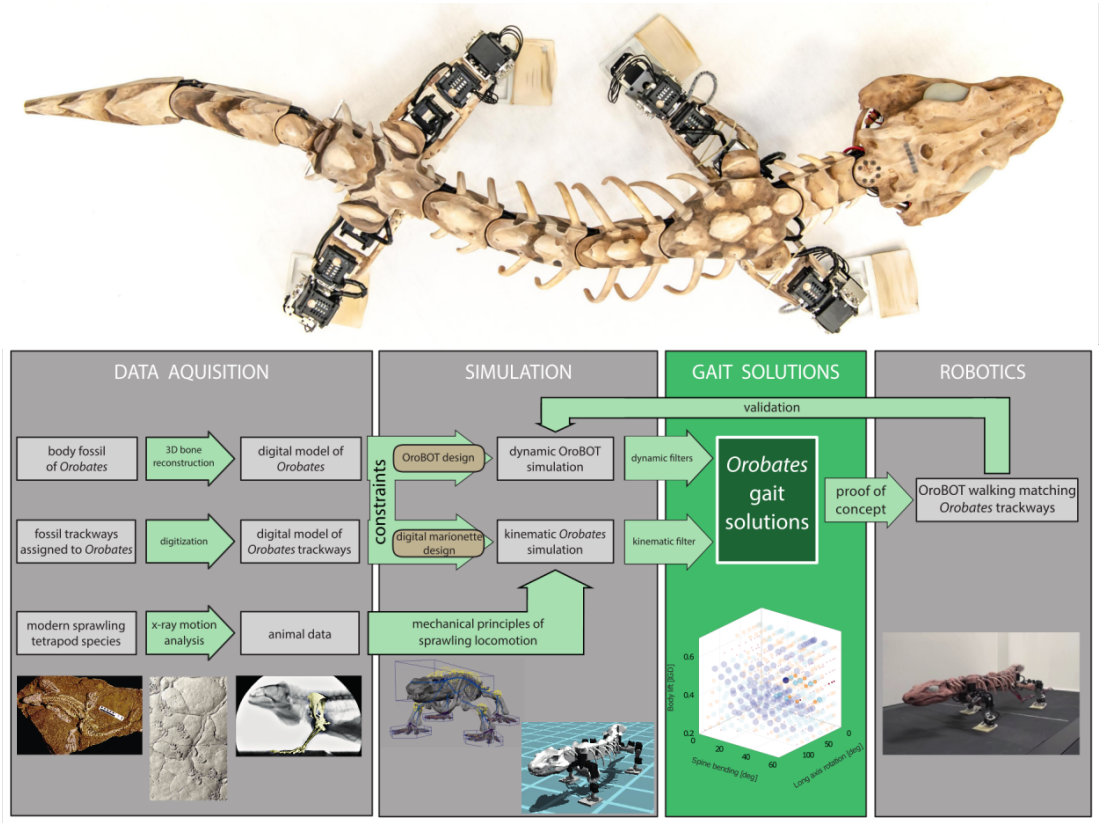


Fig. 2.25: The process of recreating the locomotion of an extinct robot using an interdisciplinary team.

2.2.2 Oscillators are the answer

Usually, the generation of real bio-inspired *CPGs* are complex. However, the idea of creating robots that take advantage of the different approaches that the *CPGs* can offer, like simple oscillators, is a very powerful idea that in general simplifies and improve the control locomotion in a lot of robots. Oscillators are so useful that, in some cases, can reduce the work of creating a new locomotion from weeks to just days.

That is the case of the modular snake robot created by *Juan Gonzalez-Gomez et Al* (41) where each module has a simple *CPG* controlled by a sinusoidal signal. Changing only a couple of parameters can change the robot's locomotion to go further, turn, or just rolling by its main axis (Fig. 2.26).

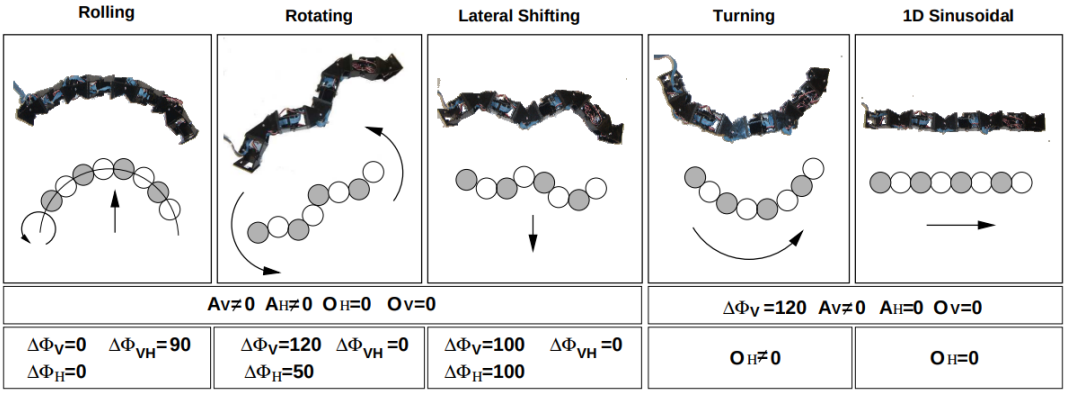


Fig. 2.26: A modular snake robot that can be controlled in different locomotions just changing the parameters of their sinusoidal CPGs.

Regarding snake-worms robots, there is another interesting approach for this kind of robot with oscillators. Instead of using one motor for each degree of freedom, they use the traditional approach used in classical *Automatas* based on clever mechanisms to generate fascinating oscillating movements with just a couple of motors.

That is the case of the worm created by *Alexander S. Boxerbaum et Al* (42). That recreates the peristalsis motion of a worm using just one motor and two wires along the body of the worm. Another similar and interesting case, is the *Saw* robot (43) from the *University of Negev* (Fig. 2.27).

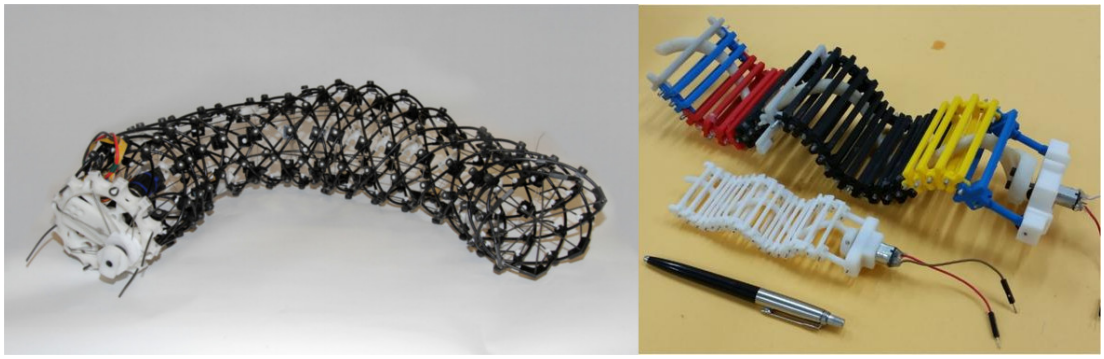


Fig. 2.27: Worm robot by Boxerbaum et al. (left), and Saw robot by the University of Negev (right).

But there is much more in the use of *CPGs* that worms, snakes and salamanders. In general, all the robots that use legs can, and should be, benefit from the use of oscillators. A great example of this is the *Cheetah-cub* robot (44). A robot that walks and runs like a cat with a very clever mechanical design with compliance spring legs in the form of a pantograph, and imitates the movements of the real cats. Thanks to the use of *CPGs* and this clever design, the robot can run in a very stable way avoiding little obstacles in an open loop without falling (Fig. 2.28).

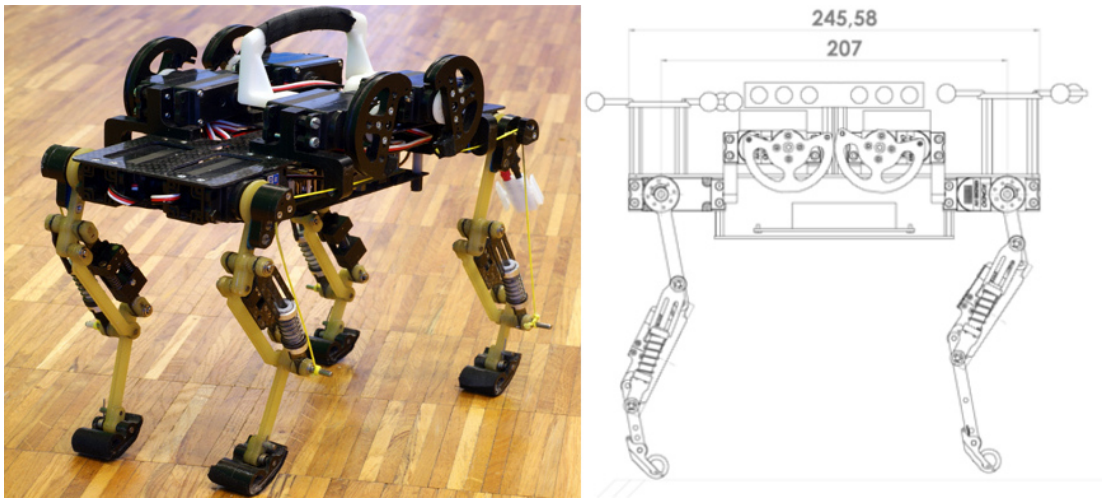


Fig. 2.28: Cheetah-cub. A robot with the locomotion and CPGs of a cat.

Even commercial toys robots and little hexapods and humanoids can benefit from the use of *CPGs* oscillators. That is the case of some of the robots developed by *Javier Isabel* (45), like the commercial and *DIY* robot *Zowi*, the quadruped robot *Kame*, or some competition humanoid robots like his robot *Raider* (Fig. 2.29).

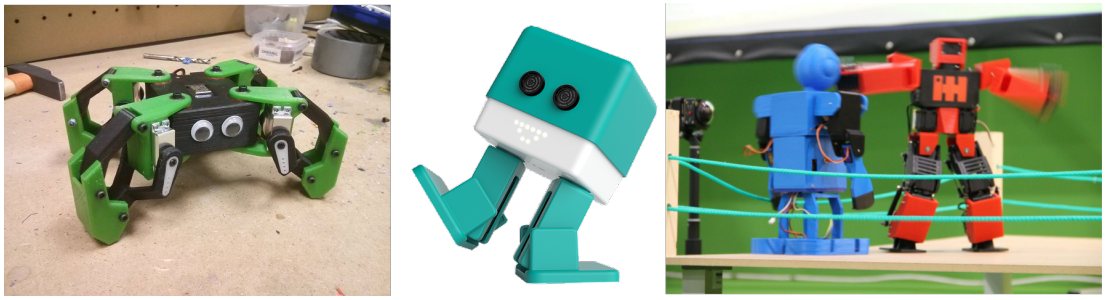


Fig. 2.29: Robots created by Javier Isabel that use CPGs oscillators for all the complex movements.

2.3 FPGAs and robotics

The *Field Programmable Gate Array* or *FPGA* is a technology that has been around for a long time. In summary (and simplifying) an *FPGA* is a special chip full of logic gates and flip-flops that can be connected between them in any way that we want.

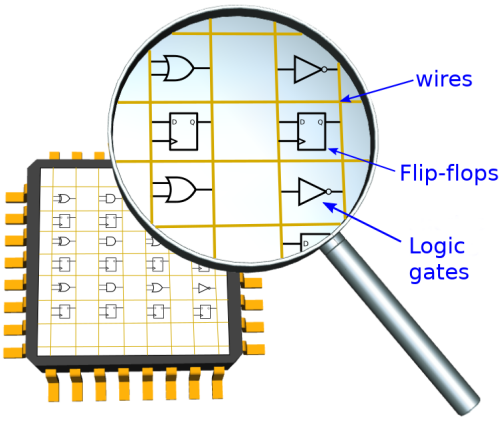


Fig. 2.30: What is inside an FPGA. By Juan Gonzalez Gomez.

This allows us to create and use different circuits inside the *FPGA*, testing and using our own hardware without the need for manufacturing the physical board. An *FPGA* is hardware, not software. We are not creating a simulation of a circuit, we are creating circuits connecting different digital components inside the chip, and we can change these circuits in a matter of minutes. Popular knowledge says that *Hardware is hard*, but is also very fast and, like the *traditional physical* circuits, all the circuits can be executed in parallel and in real time.

These properties, the ability to create our own circuits without the manufacturing process, and change them whenever we need, the fast velocity operations, and the real parallelization of the circuits that we have done, are very interesting advantages that have been using for decades in robotics.

There are tons of research and robotics projects that use the *FPGAs* as a way to implement their own hardware without the need and the cost in money and time of manufacturing. A good bio-inspired example is the use of *FPGAs* in a rat-like robot for the implementation of a sensor system that imitates the whiskers, and their rich and complex sensor system, of a real rat (46).

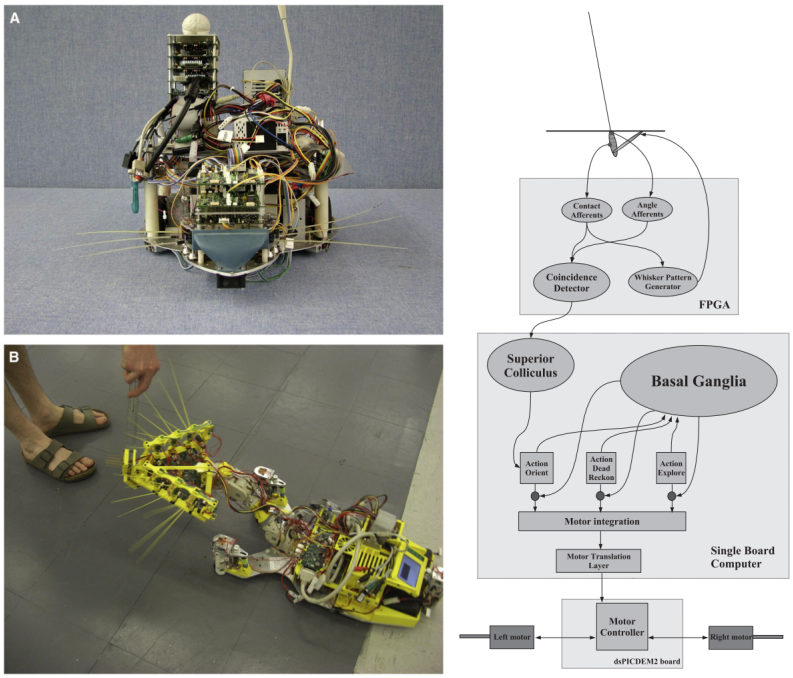


Fig. 2.31: Whiskerbot. A robot that imitates the sensor behaviour of a rat implementing its really rich and complex whiskers sensor system inside an FPGA.

As you can see in fig. 2.31, *Whiskerbot* is a robot that imitates the sensor system of a rat. Rats are well known for their excellent orientation ability, even in total darkness. Their complex whisker system, allow rats and other rodents to navigate in very narrow and dark environments where so few predators can operate. The sensory system of these artificial whiskers, including their *CPGs*, has been implemented in an *FPGA*. Thanks to the use of an *FPGA*, the robot can receive and process a lot of data for every single one artificial whisker very quickly and at the same time.

Another interesting example is the use of *FPGAs* in many kinds of different modular robots. The use of *FPGAs* allows to the researchers to create and test their own hardware all the times that they need before the final manufacture of the *PCBs* for all the modules of the robot, where usually, the most number of modules available for the same budget, the better.

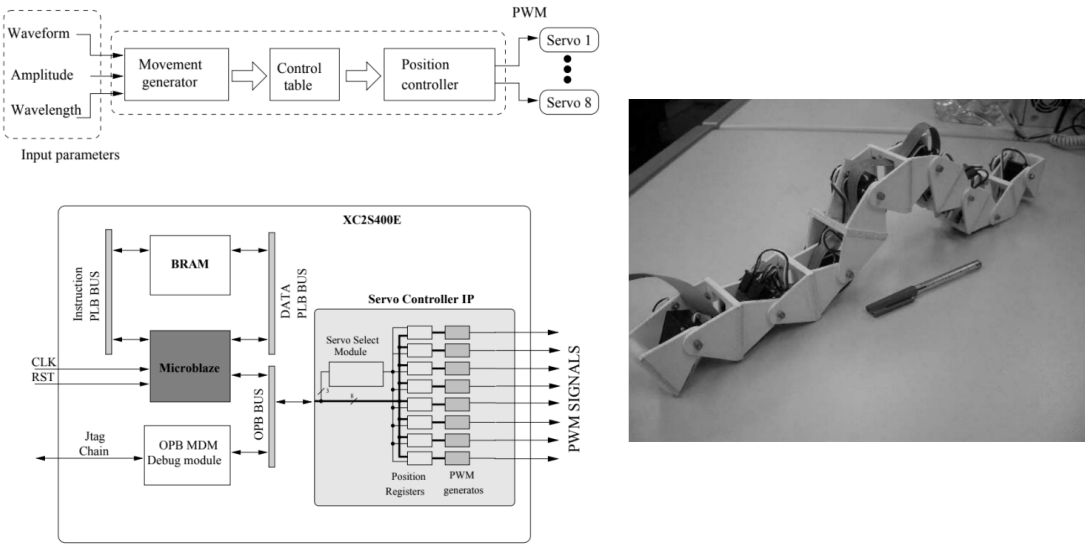


Fig. 2.32: The general locomotion scheme, and the locomotion architecture created for the snake-like robot. Thanks to the FPGA, creators can add or change things in this architecture in a matter of hours.

As we mentioned before (section 2.2.2), the modular snake created by *Juan González Gómez et Al*, is the perfect example of how *FPGAs* can help in the field of the modular robots (47). The robot can be as long as we want and it is possible to add new modules in different configurations. Just like the original project inspiration, the *Polybot* (48) that today is also the base for a lot of educational modular robots (49), this robot looks

for the maximum degree of modularity. The possibility of changing the conventional microchip with an *FPGA* allows the creators to implement new hardware architectures and faster algorithms (Fig. 2.32).

Another interesting case that combines the use of *FPGAs* and *CPGs* is the work developed by the *University of Guanajuato, Mexico* (50). They developed a locomotion system that uses *Spiking Neural Networks* (*SNNs*) that can generate *CPGs* gaits for robots with two, four or six legs, all of that implemented in an *FPGA* (Fig. 2.33).

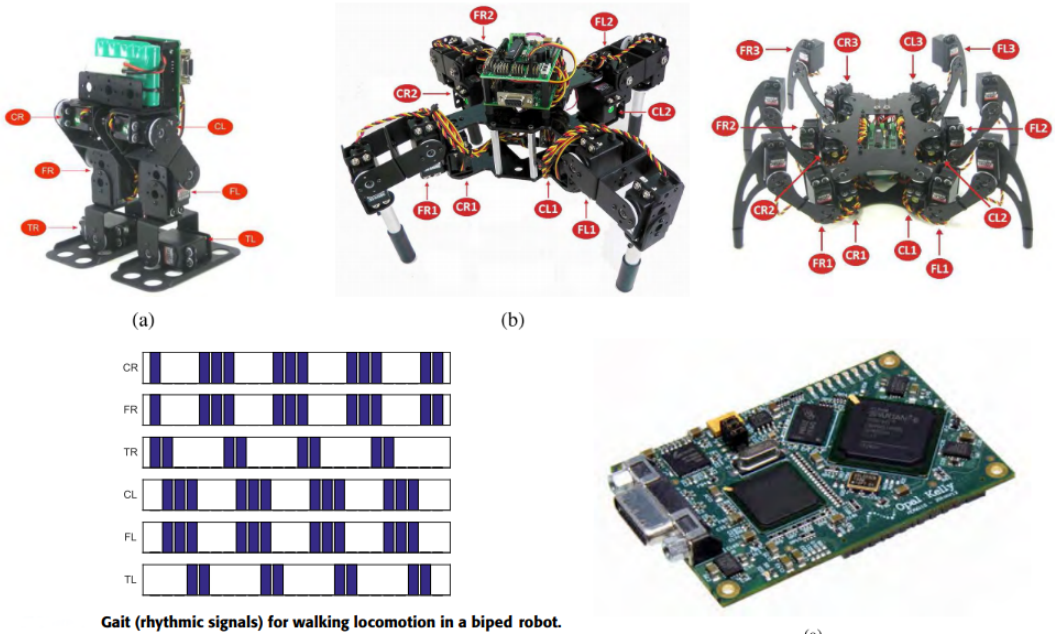


Fig. 2.33: With the generation of SNNs or Spiking Networks inside an FPGA, it is possible to develop a locomotion system that can be easily adapted to robots with two, four or six legs in a short period of time.

As we said before, *FPGAs* has been key in the development of custom hardware for different applications around the world. The robotics field has not been an exception, and for researches and even for designing products, *FPGAs* are going to play a key role in the development of new robots and *AI* systems in the following years.

The use of *FPGAs* in the field of artificial intelligence is going to be dramatically increased in a couple of years. The actual *AI* methods require the use of vast amounts of complex data and a high computational cost that is not easy to manage in an autonomous machine, like a robot, with limited resources. The use of the *Cloud*, and

the services that offer companies like *Amazon*, *Microsoft* and *Google* seems a solution for most of the cases nowadays. There is a lot of advantages of using *cloud robotics* for the artificial intelligence in robots, and one, but significant, disadvantage, the need of a very good and stable internet connection. However, in reality, we are not totally solving the problem; we are just moving away the problem to a server located on the other side of the world.

Companies like *Microsoft* are always looking for new ways to improve the performance of their servers. In the last years, *Microsoft* has been implementing neural networks to improve their server's performance (Fig. 2.34) using a combination of traditional microprocessors and *FPGAs* in their fully operational project *Catapult* (51).

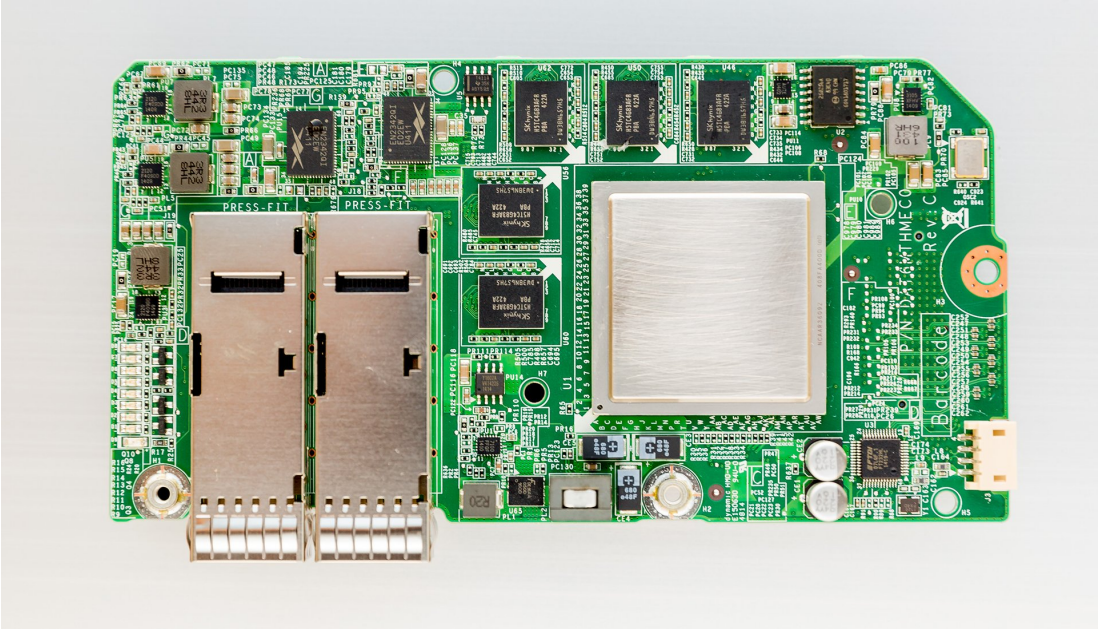


Fig. 2.34: One of the servers developed by Microsoft for the Catapult project. The idea is to implement the neural networks inside the FPGA part, improving the performance of their services with the very important and additional ability of changing the neural network inside the server FPGA without any additional cost.

One of the most evident uses of the *FPGAs* in the robotics field is the use of computer vision algorithms. If you have dedicated hardware to perform all the cost vision algorithms, the performance of the robot will drastically increase.

Using specialized hardware able to run in parallel and faster than a standard *CPU*, or even *GPU*, seems an excellent idea for computer vision. There are many works to

illustrate the use of computer vision with *FPGAs* in robotics. A good example is the work developed by *Clément Farabet et al* (52). They created a biologically inspired convolutional network (aka *ConvNets*), that can be used for multiple computer vision tasks, using an architecture that is implemented inside an *FPGA* (Fig. 2.35). To test their architecture, they use the classic face detection algorithm with very good results in a very low-power and lightweight hardware. Resulting in an advantage for drones and other kind o little robots. An ideal thing for a bio-inspired robot.

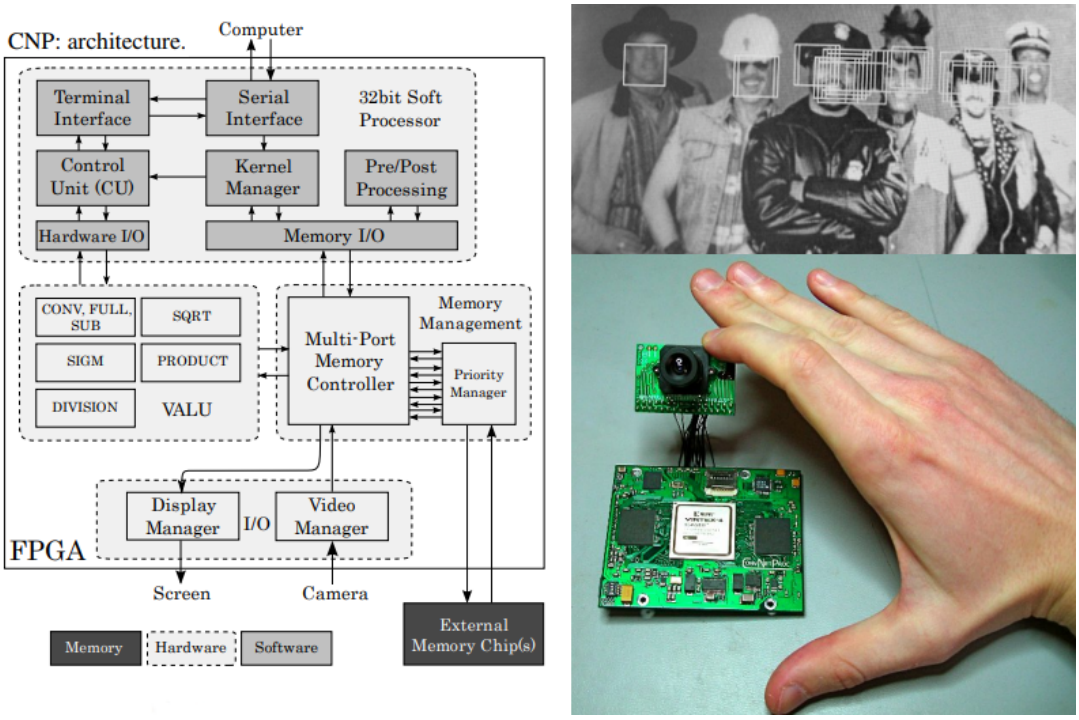


Fig. 2.35: A CNP architecture for general use in computer vision implemented in an FPGA. This architecture can occupy a minimal area of the robot working in parallel with the main microprocessor and doing all kinds of computer vision tasks like face recognition.

3

Used technologies

As we explained before in this document. The goal of this project is to create a Robotics Hybrid control architecture inspired in animal nervous systems, especially vertebrates and humans.

Humans have a very complex nervous system that can be simplified, or split into two main parts: The high-level system or *Central Nervous system* (*CNS*) and the low-level system or *Peripheral Nervous system* (*PNS*). The *CNS* is the brain and the central spinal cord and is responsible for the high level and more abstract orders, on the other hand, the *PNS* is responsible for the low level and more detailed orders.

In this chapter, some of the technologies that have been used to create a robotic version of this architecture will be briefly explained. Some of them are common knowledge nowadays, and others are quite new and experimental. Creating robots implies a lot of different tools that involves digital manufacturing, hardware development, and software development. We are going to categorize and explain all of these tools in three main sections. The tools that have been used to imitate the high-level system, the tools used for the low-level system, and the cross-tools that are necessary for the general development of the architecture and the different robots.

3.1 Technologies used in the high-level system

The high-level system is responsible for the more complex and abstract orders, for example, task planning, path planning, complex sensing, and human interaction. These kinds of tasks require multipurpose machines able to run different and multiple programs, in other words: CPUs.

3.1.1 The brain of all the beasts: The CPU

The *CPU* or the *Central Processing Unit* seems, in some way and making lots of simplifications, the artificial equivalent of a human brain. The *CPU* is the central and more important part of a computer. It is responsible for running all the instructions and logic operations of the computer programs, manage the available memory, and control the different inputs and outputs.

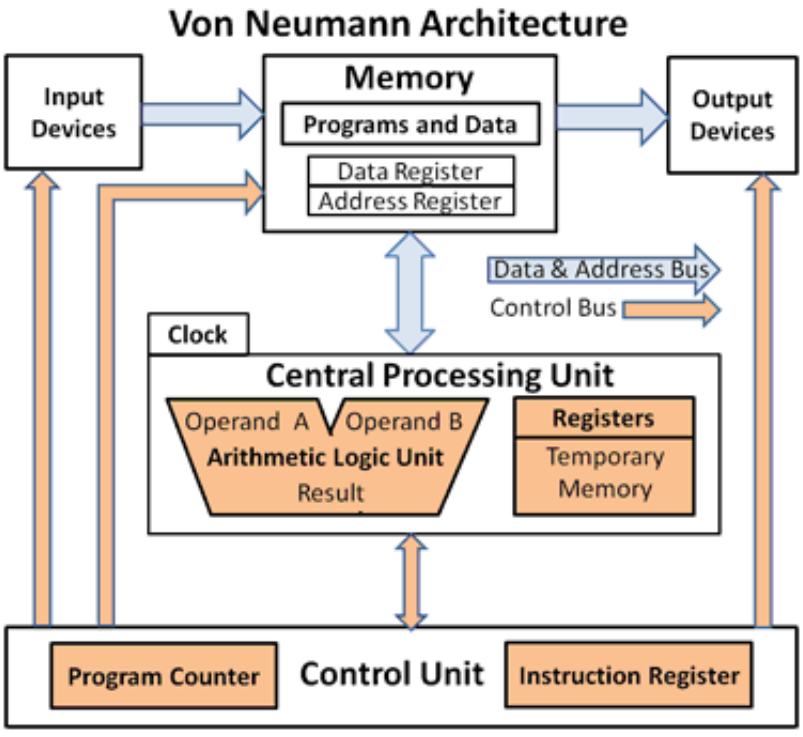


Fig. 3.1: The well known Von Neumann architecture, the architecture that rules the computer world from the beginning.

If we talk about *CPUs* we can talk about fascinating subjects like the first automated machines, like the automated looms or the first *automatas*, and famous figures like *Alan Turing* and his *Universal Turing Machine*, the Von-Neuman architecture (Fig. 3.1), used in all the modern *CPU*, and so on. However, the purpose of this chapter is not to explain what is and how works a *CPU*. A *CPU* is a very good well-known machine used every day and everywhere. Like the car or the fridge, is part of our ordinary life and actually, each one of us have one inside its pocket. Furthermore, it seems more important to talk about how we are using *CPUs* inside our robots.

What is the difference between a washing machine, your cellphone, and a robot? Of course, there are many differences, but all of them are created and programmed using the same element, a microprocessor or *CPU* using the well know *Von-Neuman architecture*. At the control level, their technology is practically the same, and there is not a real difference between programming the spin cycle of a washing machine and the decision planning of a social robot.

The *Von-Neuman* architecture was developed for a multipurpose machine that, on the basics, it does not run different programs in parallel and at the same time. Nowadays our *CPUs* with the help of other elements in the computer like the *GPU* or the sound cards, is able to perform multiple tasks at the same time, or at least, that is what it seems. The reality is that *CPUs* are constantly changing between programs and different process, using a lot of clever tricks to have most of the clock cycles busy doing useful work. It seems enough for most of the jobs but not for all. The problem is that you are like a juggler that have a lot of balls in the air, or programs, and you are constantly switching to ball to ball as fast as possible to keep all of them in the game. What happens when you start to add more and more balls? In the end, your velocity and all the tricks that you have been using are not enough, and balls start to fall. You probably need more hands, more velocity, or a brain able to perform all of these tasks in a real parallelization, as we will see further in this document.

What all of this mean for robots? Robots usually are physical entities able to move that have, like the living forms, two critical constraints: size, and energy autonomy and performance. In few words, you cannot usually use the most advanced and powerful computer inside a robot because it is too big for the shape of your robot and you will not probably be able to properly power it with a good time autonomy.

If you add to the last equation also the prize, things start to get complicated. You have, for example, to sacrifice power computing to be more affordable with the proper size and power consumption, that means that your robot will need more extra computing power, and other specials or secondary *CPUs* will be added.

In the last decade, more and more microprocessor and small computers have been developed, especially in the open hardware movement and in the *IoT* industry. Today it is easy to find many of relative cheap microprocessors with enough computer power to control a simple robot or and *IoT* device. In this area, two main boards have been crucial in the last years: The Arduino boards and all the open hardware boards related to this movement, and the Raspberry Pi.

3.1.1.1 Democratizing the hardware: The Arduino movement

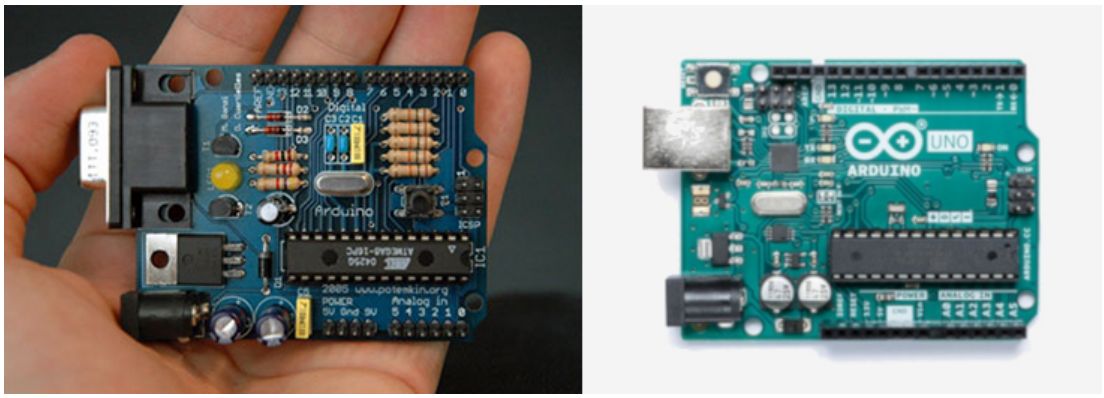


Fig. 3.2: One of the first iterations of the Arduino UNO board (left). The Arduino UNO model today (right), one of the most famous used boards.

In a world where the open movement of software has changed the way like traditional software companies, like *Microsoft*, work, it is a logical step also to have *Open Hardware*. An *Open Hardware* is a hardware design or architecture that it is under an *Open Source* license. These licenses usually allow any person to create, manufacture, modify and even sell this hardware or a derivative of it, always respecting the original authority, and contributing to the open hardware community with the new improvements and designs.

In this way, the Arduino movement has been crucial in the democratization of the *Open Hardware* movement and the creation of the denominated *Maker movement.*

Arduino was born in 2005 (53), when *Massimo Banzi* and *David Cuartielles* created a programmable board for creating interactive art design projects in the *University of Ivrea*, Italy. Almost two decades later, *Open Hardware* boards are widely used in Universities and a lot of companies. These boards are relatively cheap and easy to use, thanks to the use of *Open Source* common hardware libraries, typically written in C/C++, under the standard programming language paradigm of Arduino.

Today *Open Hardware* boards are often used in primary and high-school education, university research, fast prototyping and even products in private and public companies all around the world. An *Open Hardware* board is cheap to use and replace, it usually has a straightforward learning curve, a huge and active community for learning and improving the system, and all, or at least most of the source files, published under an open license to be used and modified if it is needed.

For robotics, the *Open Hardware* movement has been a fantastic change in the way of doing research and create prototypes and commercial robots. Nowadays, a eight years old child can create and program his first line follower robot almost without any help and with a very low cost. This means that today, we are able to create new robots with lower costs and taking less effort in terms of time and learning. Thanks to this democratization, today is easier than ever to use your own robotics solutions and experiment and research to try to solve some of the challenges that the robotics field is actually facing.

As we see in more detail in the next chapters, in this work *Open Hardware* boards, like *Arduino Uno* or *ESP32* (54) have been used in the developed robots as a way to quickly fast prototype the high-level control, the movements and others features, and as a way to quickly develop the communications between the high-level and low-level system of the proposed architecture.

3.1.1.2 A computer for everything: The Raspberry pi

In 2012, the *Raspberry Pi foundation* created the first version of the Raspberry Pi (55). The Raspberry Pi is a single board small computer which was created as a very cheap and capable computer for teaching computer science in developing countries (Fig. 3.3).

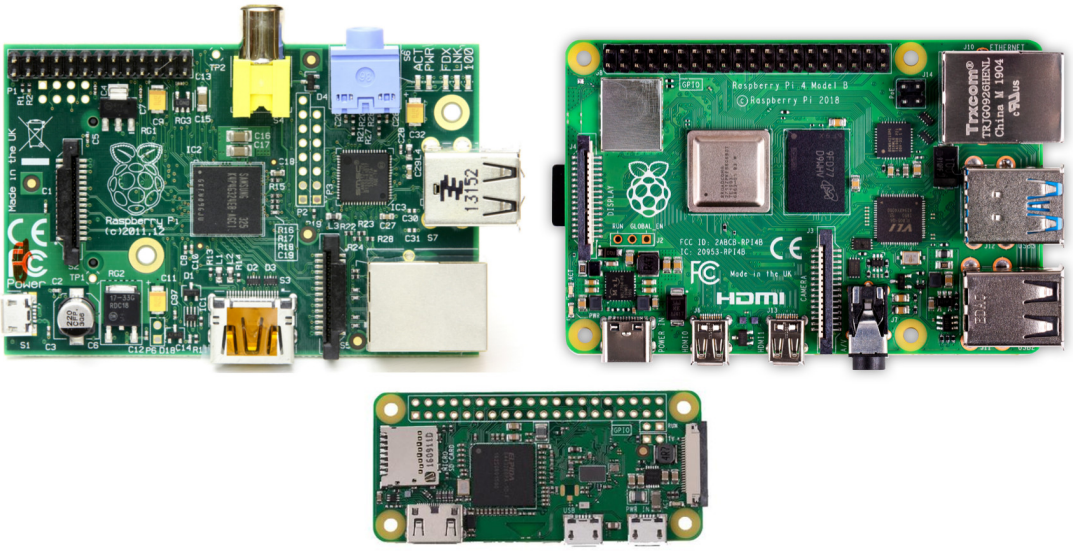


Fig. 3.3: First Raspberry Pi (left). A Raspberry Pi zero, a tiny computer (bottom). A Raspberry Pi 4 model B (right), one of the last and more powerful models.

However, it soon became more than a product for only such a noble purpose. The Raspberry Pi computer quickly became a perfect low prize solution for robotics, fast prototyping, and *DIY* projects for the *Maker movement* and even as the main computer for some Kickstarter products.

Today, as it happened with the Arduino and *Open Hardware* movement, you can use a Raspberry Pi for almost anything. It can be a multimedia center for your TV, a simple radio transmitter station, a small server, a retro videogame emulation station, a *sniffer* for hacking and *IoT* purposes, a control server for your 3D printer and much more. And above all of that, an economical first computer to teach computer science using Linux.

The Raspberry Pi is a small computer with all peripherals that are typically needed, like an *HDMI* output and audio output, a CPU, small GPU, USB ports, Wifi and Bluetooth, and event special connectors for small cameras and others.

However, the most interesting thing about the Raspberry Pi is the general inputs and outputs pins (GPIO) (Fig. 3.4) that comes in every model and that, in conjunction with its low price, small size and *relative powerful* performance, makes the Raspberry Pi an ideal CPU for small robots and research. Furthermore, they are compatible with the robotic operating system *ROS* and powerful enough to run computer vision algorithms and artificial intelligence engines like *Tensorflow*.

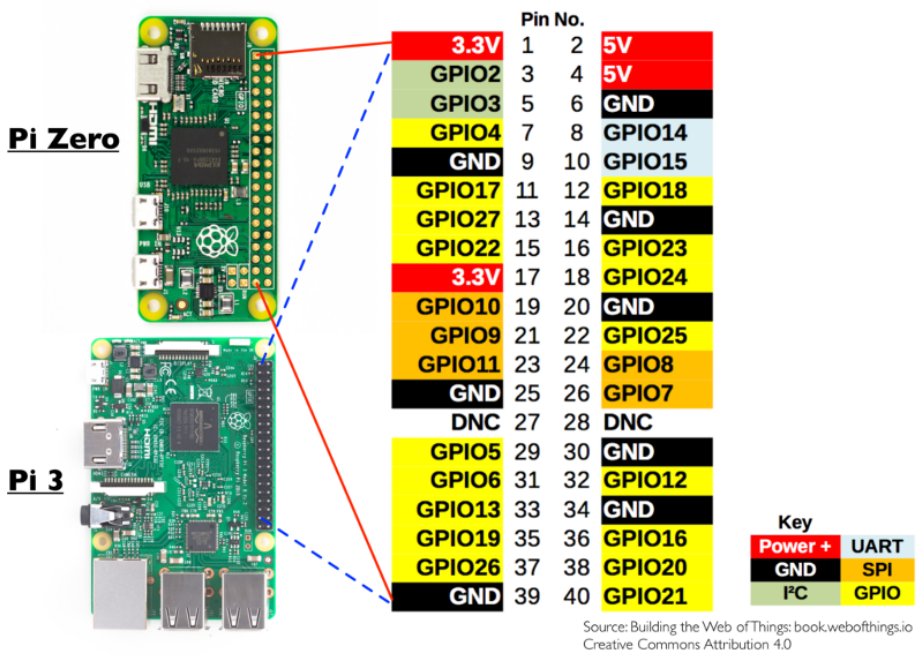


Fig. 3.4: GPIO pins of the different Raspberry Pi. Thanks to these pins and its low price and size, the Raspberry Pi is an interesting choice for robotics applications.

At the moment of writing this document, the *Raspberry Pi foundation* has released more than eleven models of their boards. The last one, the *Raspberry pi 4 model B*, can have until four gigabits of ram using a *4x CPU Cortex-A72 at 1.5 GHz*.

The use of the Raspberry Pi computers has been key in the development of this work. As we will see in the following chapters, thanks to their low price and relatively small size and easy setup, different Raspberry Pi computers have been used as the brain, or high-level system, for the proposed control architecture, being the main CPU of the developed robots.

3.1.2 Controlling the power: The high-level software

Nowadays, the world is full of new programming languages and libraries that makes life better. When we speak about robotics, it is hard to say that we are only using one language, most of the commercial robots can be programmed in different languages, like the *Naoqi* operating system for the *Aldebaran* robots (56), that can be programmed in Python, C++, Javascript or even using a graphical programming language called *Choreographe*. Something similar happens with the robotics operating system called *ROS* (57) where you can create and use packets created with the excellent performance of C/C++, or the simplicity and flexibility of Python. In the end, roboticist has to choose the best solution taking into consideration the balance between *computer performance/programmer effort* and at the same time experiment and embrace new programming languages and ways of doing things.

During the creation of the proposed control architecture of this work, two main programming languages for the high-level control have been used: C/C++/Arduino and Python.

3.1.2.1 It is always there: C/C++

We can say so many things about the new high level programming languages, their flexibility, supposed simplicity, power, and much more. However, all of these things do not matter when your usually limited robot's CPU does not have the right performance that you are looking for. If we talk about computer efficiency, the C language is probably the best choice without entering in the caverns of Assembly. If you need an exceptional performance with high-level programming approaches, like *Object-oriented programming (OOP)* you will have to sacrifice some of the goods things of C and use the probably still most used programming language in the world: C++.

In the development of this work, C/C++ languages have been used inside the framework of Arduino. The Arduino framework it is just a collection of libraries written in C/C++ created for programming microcontrollers and all the sensors and actuators attached to them. Thanks to Arduino, it is straightforward to put the same program to one microcontroller to a different one, with a good performance and an easy learning curve. There are tons of different libraries for Arduino, and it is relatively easy to create your library written in C/C++ and use it inside an Arduino project. Thanks to

these impressive amounts of libraries, and the simplicity of the Arduino framework, it has been very easy to fast prototype the basic behaviour of the robots of this project in a matter of hours.

3.1.2.2 A wizard with so many tricks: Python

In the last decade, very interesting high level languages have emerged. The emerging technologies like the named world wide web 2.0 and the rising of the smartphones and apps development, plus the actual use of web apps for almost everything and the big data and the Artificial Intelligence development have done that Python is one of the most exciting and most used programming languages in the near future.

Python is an interpreted high-level language created for general purposes that emphasize in the code readability and supports multiple programming paradigms, like object-oriented, procedural, and functional programming.

If we think about robotics and fast prototyping, Python is a great language to use. It is very easy to install and import a new library; there is a vast amount of useful libraries usually supported for a large community of people and thanks to that you can connect any piece of code with any piece of code having a functional and well-done program in a matter of minutes.

However, Python is not perfect. When we leave the comfort zone of having a powerful CPU, the difference in performance between a C/C++ code and its equivalent in Python it is always significant. A good example of this is the computer vision field and *AI*. When you do heavy tasks like applying computer vision algorithms or training a neural network, Python is significantly slower than C/C++. That is the reason why so many libraries have reached a great compromise, using Python as a wrapper from the real libraries written in C/C++. With this solution, the programmer gains the fast programming and flexibility of Python, and the machine executes the equivalent C/C++ code without losing too much performance. An excellent example of this kind of implementations is the Python OpenCV wrapper (58).

As we will see further in this document, Python has been the main programming language for the high-level control system of the implemented robots thanks to its high flexibility and easy setup inside Linux-based CPUs like Raspberry Pi.

3.2 Cross technologies and tools

Robotics is a field that covers many fields of study. To create a robot from zero to one, it is necessary to know mechanics, dynamics and kinematics, CAD design, electronics, firmware development, software development, and even things like natural language processing, psicology or, in the case of the bio-inspired robots, a little bit of biology. Of course, this normally means that to create a robot companies and universities need multidisciplinary teams able to work together and create all the pieces of the puzzle. In this section, we are going to talk about all the different technologies and cross-tools that have been used to prototype and create the different experiments and robots developed for this work.

3.2.1 Creating a body: CAD Design and Freecad

Almost every robot created in the world needs a physical body, and structure that, in the end, defines which kind of robot is, what can do, and with which purpose.

During the development of this work, different robots and experiments have been created from scratch or modified using existing designs. To create or modify these CAD designs the *Open Source Freecad* tool has been used (Fig. 3.5).

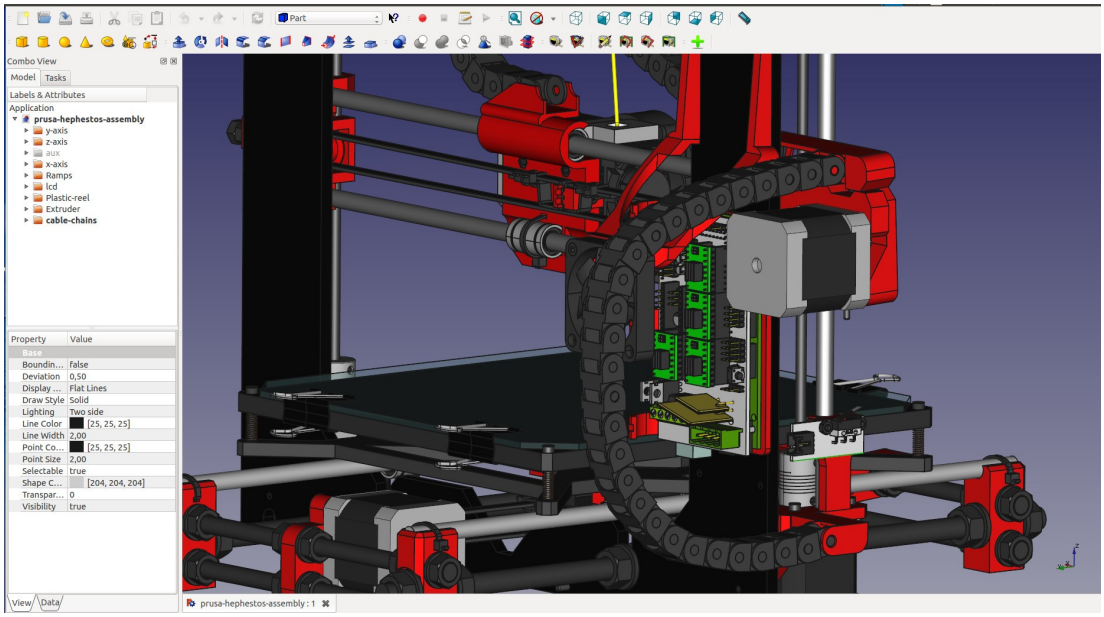


Fig. 3.5: Freecad design of the 3D printer Prusa i3 Hephestos

Freecad (59), is an *Open Source* CAD design tool intended to be used for technical design of mechanical pieces. Its main feature is its *parametric design*, that is, all the parts are created in an incremental way where if we change one parameter, all the design will be updated to fit the new change. This is very useful for prototyping and industrial design and is the same approach that some traditional tools, like *Autodesk Inventor* (60) uses.

One of the most interesting things about *Freecad*, it is the workbenches. As many *Open Source* tools, *Freecad* is a modular program where everyone can create a workbench to expand the possibilities of the tool. There are multiple workbenches for almost everything, like assemblies, generation of drawings, architecture and naval designs, and much more. However the most used workbenches in *Freecad* are the *Part workbench*, to create parts using primitives shapes and do booleans operations, and *Part design*, a powerful workbench to design parts using 2D sketches and extrude them creating complex parts. During this work, different versions of *Freecad*, (0.16, 0.17 and 0.18) have been used. However, all the developed designs are compatible with the last version: 0.18.

3.2.2 From the computer to the real world: 3D printing

3D printers are not new. The first *SLA* printer was created by *Charles Hull* in 1984. However, and thanks to the *Open Source Reprap* movement created by *Adrian Bowyer et al* (61), in the last decade 3D printers have become a relatively affordable and easy to use tools that are present in some many companies, universities, *Fablab* spaces and houses.

There are some many technologies of 3D printing, depending on the way and the materials that are used to create the objects, but the two most used nowadays are the stereolithography (SLA), and the Fused Deposition Modeling (FDM) or Fused Filament Fabrication (FFF). While the quality of the *SLA* printers are far superior from the created with the *FDM* technology, *SLA* printers are more expensive, not only the 3D printer, but also the required materials (normally a special resin) and replacement parts, and require more printing time.

Also, the post-processing workflow for each *SLA* 3D printed object is more dirty and requires more effort than using a *FDM* 3D printer. For all of these reasons, *FDM* 3D printers are more popular nowadays. They are easy to use and mantain, materials,

normally *ABS* or *PLA* plastics are cheaper and the printing and post processing time is tipically smaller than using other tecnologies.

For this work, a *Prusa i3 BQ Hephestos* has been used (Fig. 3.6). The *Hephestos* is a 3D printer based on the popular *Open Source* design *Prusa i3*. It is a very open an easy to use *FDM* printer that, with the proper calibration, maintenance and knowledge, can create objects with an excellent quality and a very low cost. For most of the created robots and experiments, *PLA* plastics have been used.

Fig. 3.6: Prusa i3 Hephestos. A very easy to use and maintain 3D printer.

3.2.3 Creating new circuits: Kicad

Sometimes the commercial boards that we are using are not enough for the design of our robots, and some additional circuits must be added. We can create these circuits using discrete components and breadboards, or maybe if the circuit is complex, we can create a custom PCB circuit.

Kicad (62) is an *Open Source* tool for designing *PCB* boards (Fig. 3.7). It is widely used for small protoyping and commercial purpuses and there are a lot of repositories and manufacturers that offers the *Kicad* symbols and footprints of their components.

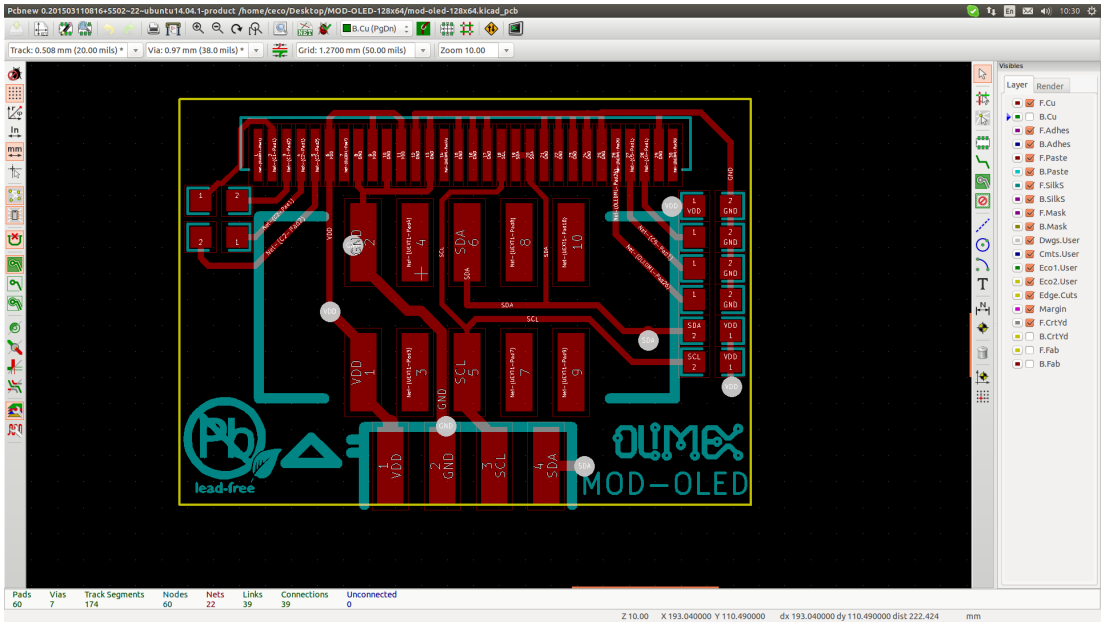


Fig. 3.7: Kicad footprint editor named as PCBNew

For this work, some small *PCBs* has been designed using the *Kicad 5.1.2 Stable Release*.

3.2.4 An IDE for everything: Atom + Plattformio

When we talk about programming, a simple text editor plus the terminal should be enough to do most of the job. However, creating code is not only about creating programs that work. It is also about creating programs that are easy to read, document, fix, scale, and improve by other people or ourselves in a distant future. The *integrated development enviroments* or *IDE* save a lot of time and effort.

When we talk about *IDEs* there is a clear division about which one is better between the developers based in their personal preferences and daily work. Nowadays, *Visual Studio Code* (63), *Sublime text* (64) and *Atom* (65) are the most popular.

For this work, the *IDE Atom* has been used as the main programming environment. *Atom* is a general purpose *IDE* based in web technologies that is totally customizable. It is effortless to install new packages to expand its possibilities, even creating new packages for new needs, being perfect for fast prototyping when different languages and systems are systematically used (Fig. 3.8).

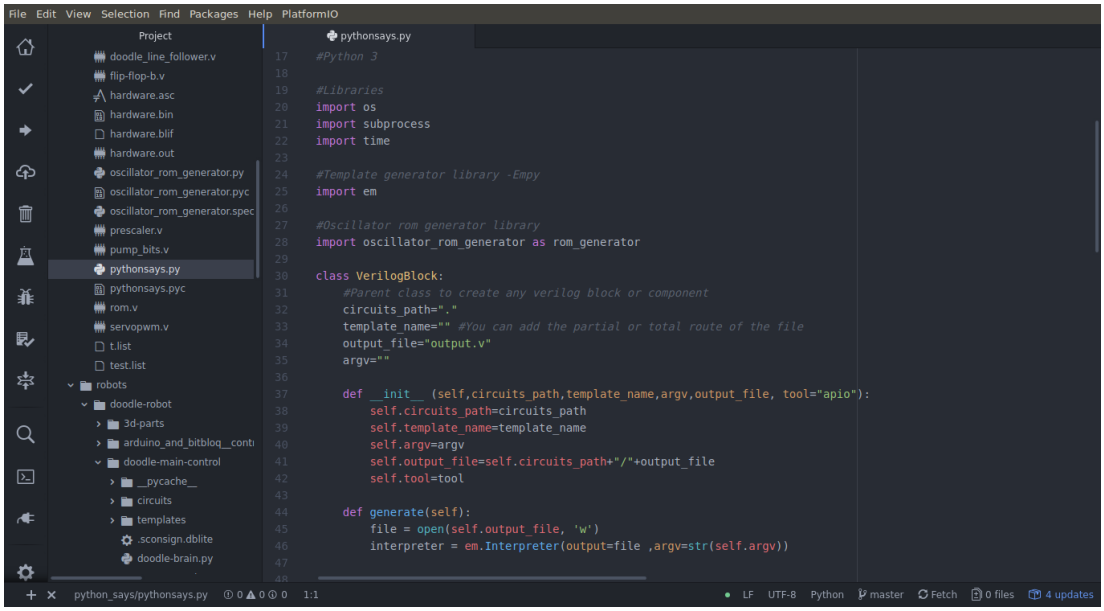


Fig. 3.8: Atom Editor. A general IDE based on web technologies and perfect for doing fast prototyping.

However, there are more reasons for using *Atom*. If we talk about programming microprocessors, the *Plattformio Cross-platform IDE* (66), compatible with *Atom* and *Visual Studio*, can simplify the programming workflow of a board in an impressive way. With *Plattformio* we can create new projects for our microprocessors in a matter of minutes, without worrying too much about all the necessary libraries and drivers that each board needs, *Plattformio* do this work for us. We can also import the last version of a library, keep it updated, and download and see their examples. Having all the debug tools, like the compiler, uploader tool for each board, and serial monitor, that are normally needed for programming a microprocessor.

As we will see in the next section, *Atom* has also been used for developing some *Verilog* code with *Open Source FPGAs* using the *Apio-Ide* extension.

3.3 Technologies used in the low-level system

The low-level system is responsible for the more concise, detailed and repetitive orders, like the whole sequence of movements of the motors of a leg to move forward, the basic sensing response of simple sensors or even things like the security and reflex movements.

In so many cases, this kind of low-level tasks needs to be very fast and work some of them in parallel at the same time. As we said in the technologies used in the high-level, *CPUs* are capable of doing multiple tasks in parallel using multiple tricks to keep the *CPU* always busy, but at the end and due to their architecture, *CPUs* are not capable of doing multiple tasks at the same time without losing performance when we increment the number of tasks working at the same time.

What can we do to have a real parallelization of tasks with the best speed that we can reach? One solution that is typical in robotics nowadays is using multiple secondary microprocessors and others *CPUs* that are specialized in doing only some of the tasks, like the motor behaviour, the sensing of the head of the robot, and much more.

Another approach, the presented in this work, is to use our own circuits, our own hardware, designed to do only what we want to do as fast as possible. The good think about the hardware is that all of these circuits can be totally independent of the others and can work in parallel, with an impressive speed that usually is much faster than their software equivalent program.

However, creating our own circuits is difficult; it requires a lot of discrete components and the construction of the circuits using breadboards or designing and sending to fabricate our own PCBs. The cost is high, the effort is huge compared with the software development, and every time that we want to change some tasks or just fix an error in the behaviour of the robot, we have to create new hardware again.

That is the principal reason why the first approach, using different *CPUs* to solve the task load problem, is more used nowadays. However, if we take the best things of the second approach, if we create specialized hardware that is capable of doing the job faster and in real parallelization, without the need for physically building and manufacturing the hardware every time, we will have an interesting solution that will be able to work with better performance and even more flexibility than the first one.

Is it possible? It is, thanks to the technology called *FPGA*.

3.3.1 The beast that beats all the beasts: The FPGA

The *FPGA*, or *Field-Programmable Gate Array*, is an extraordinary piece of hardware. An *FGPA* is a chip that contains thousands, even millions, of logical gates, bistables, and other basic digital components. The interesting thing about it is that the wires or connections between each component are totally configurable (Fig. 3.9).

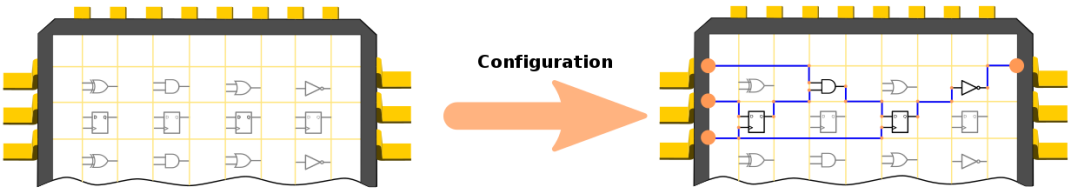


Fig. 3.9: A simplified explanation about how works an FPGA. By Juan Gonzalez Gomez.

Thanks to this special architecture, you can connect some logic gates and flip-flops to create a circuit timer, that can be a different circuit, like an *UART* driver, just rewiring and adding other digital components inside the *FPGA* in a matter of minutes, and without the necessity of building or manufacturing the circuit.

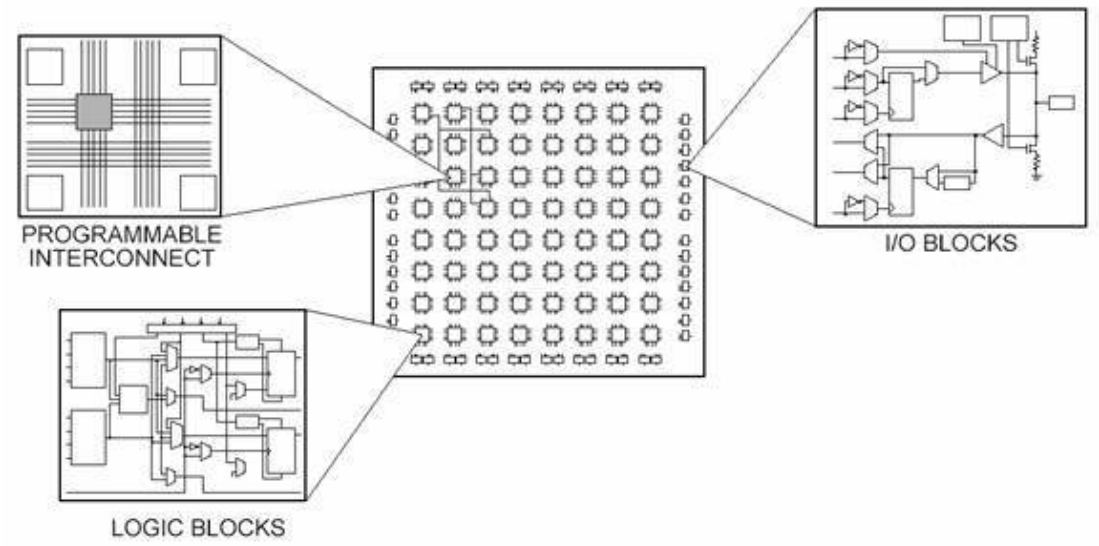


Fig. 3.10: Basic parts of an FPGA. Thanks to the programmable interconnected wires we can connect the different logic blocks and I/O blocks to create different circuits.

The internal architecture of an *FPGA* (Fig. 3.10) can change from model to model and manufacturer, but in general, most of the *FPGA* have the following parts:

- PIO = Programmable I/O: The available programmable inputs and outputs of the *FPGA*
- PLBs = Programmable Logic Blocks: The logic blocks, a special architecture usually formed by four input *LUT* or *Lookup tables*, a *Full adder (FA)* and a *D-type flip-flop*. With this architecture, a logic block is able to create different reconfigurable logic gates (Fig. 3.11).
- BRAMs = Block RAM Memory: Blocks to store large quantities of data, like lookup tables and more.

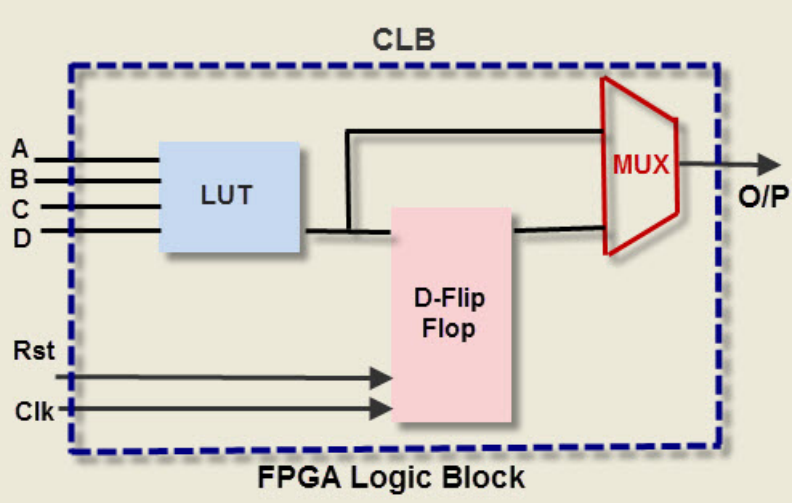


Fig. 3.11: The Logic block, the basic part of creating any circuit inside an FPGA. Image created by elprocus.com

Thanks to these different parts and the configurable interconnected wires we can create almost any digital circuit inside an *FPGA* with the only limitation of the space, or the number of available *PIO*, *PLBs* and *BRAMS* inside the chip.

The *FPGAs* have so many advantages. An *FPGA* combines the best parts of the software and hardware worlds. It is essential to understand that, when you create circuits using an *FPGA*, you are not simulating a circuit, you are creating it inside the chip, it is real hardware, with real components that follow the real digital electronic

laws. This means that even a badly designed circuit will probably be much faster than its equivalent software program. As we previously said in this chapter, the architecture of *CPUs* is not really designed to do real parallelization. We have to jump from one task to another quickly and use multiple cores for achieving something similar. **When we talk about hardware and *FPGAs*, the problem of parallelizing tasks is trivial**. The circuits work in parallel and at the same time, without sharing resources, and we can synchronize them using clock signals and others. Of course, designing hardware is usually more complicated and requires more effort than software. However, it is possible to accelerate the hardware process design thanks to the *HDL* languages.

3.3.1.1 Describing Hardware - The HDL languages

Hardware is hard. This is a common phrase used nowadays to explain in a few words the difference between the hardware and software development. When we talk about designing circuits, the first and best approach is drawing the circuits with pencil and paper, creating the circuit diagram to truly understand what components need the circuits and how they are connected. At the beginning of the history of mechatronics, after the blueprint usually came the physical creation of the circuit, using discrete components.

However, with the development of the microchip and the integrated circuits, the manufacturing of circuits increased its complexity with the use of special machines like *CNCs* and all the chemical process that requires the creation of a *PCB*. The use of computers to create these new kinds of circuits was mandatory, and for this reason, programs to draw the circuit diagram on the computer in a graphical way were created. These graphical programs are useful for describing circuits and are still used in hardware development. However, with the rise of the *FPGA* technology and at the same time the creation of high-level languages in the world of software, a new way of creating circuits were created: *HDL* languages- A *Hardware Description Language*, or *HDL* is a computer language *to describe circuits* in a high level and more abstract way that using circuits diagrams. It is important to note that **they are not programming languages**, just like when we describe a web page using *HTML*, with *HDL* languages we are not programming a circuit, we are describing how it is and works. This difference with a software programming language it is so crucial to avoid many errors in the circuits, there are some things that can be programmed in software that do not have

sense and are impossible to create in real circuits, and for these reasons a change of mindset when we switch between developing software to develop hardware is critical.

3.3.1.2 VHDL

VHDL is a hardware description language used to describe the systems and behaviour of electronics circuits. It was created in 1983 by the *U.S Department of Defense* and it is well based on the programming language *ADA*. *VHDL* is designed to manage the parallelism inherent in hardware designs, and it is strongly typed in order to represent common operations of hardware. It can be used to describe, simulate, and synthesize circuits in *ASICs* chips, *CPLDs* and *FPGAs*.

Although *VHDL* is commonly used in the development of *FPGAs*, it has not been used for this work in favor of the *HDL Verilog* language.

3.3.1.3 Verilog

Verilog is a *HDL* language designed to model electronic systems and can manage the parallelism present in the hardware designs. It is normally used to design and verify digital circuits, but it can be used also with analog, mixed circuits, and even to program cells creating *Synthetic biological circuits* (67).

Verilog was created practically at the same time that *VHDL*, between 1983 and 1984, by *Prabhu Goel, Phil Moorby* and *Chi-Lai Huang*. *Verilog* was created with a syntax similar to the programming language *C*. A design written in *Verilog* consist in a hierarchy of modules that communicates between them using inputs, outputs and bidirectional ports.

```
1  module counter(
2    input wire clk,
3    input wire rst,
4    output reg [7:0] value
5  );
6  always @(posedge clk)
7    begin
8      if(rst==1)
9        value<=0;
10     else
11       value<=value+1;
12   end
13 endmodule
```

Listing 3.1: Verilog example of a counter

For this project, *Verilog* has been the main *HDL* program to describe and simulate the different circuits present in the *low-level* control system. Its syntax is clear and very similar to *C/C++*, and the modular design of *Verilog* simplifies the problem of creating, reutilize, and maintain complex circuits formed by more basic components.

The *FPGA* industry is a robust industry of more than thirty years old. The main use for *FPGAs* nowadays is for hardware design and development of microprocessors and other kinds of complex *ASICs*. *FPGAs* are usually used in the industrial world by very specialized electronic engineers. There are two main manufacturers: *Xilinx* and *Altera*, plus a bunch of few manufacturers that also creates their own *FPGAs*, like *Lattice* or *Microsemi*. Due to the huge *market share* than *Xilinx* and *Altera* have, the *FPGA* industry can be considered nowadays an oligopoly, where the hardware is normally very expensive, and licenses are always privative and designed for medium-big companies and not for personal or sporadic projects.

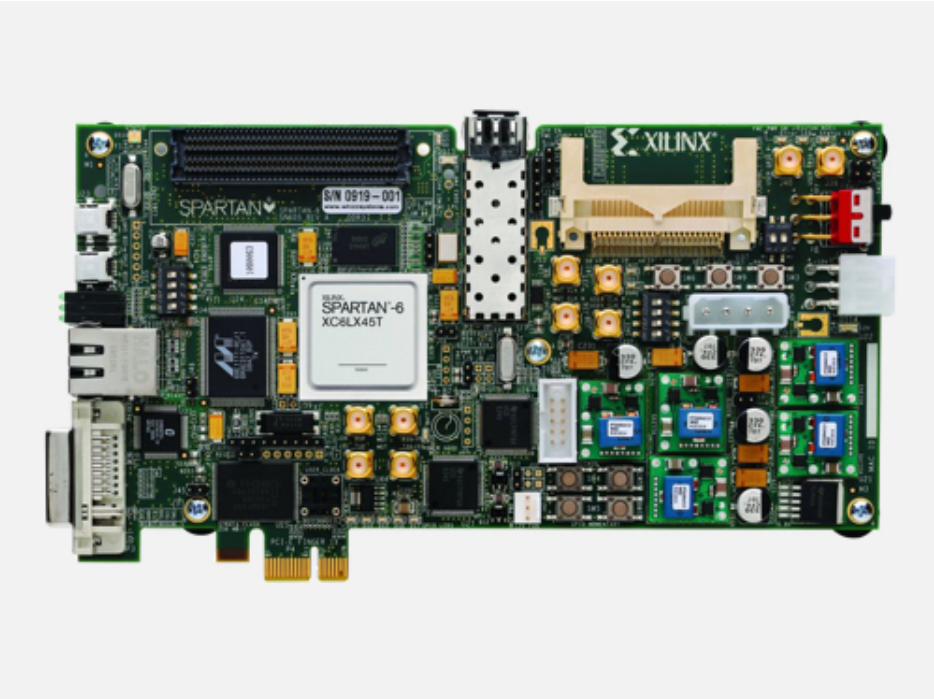


Fig. 3.12: A Xilinx Spartan 6 development board. One of these official kits usually are around 600 euros. More advanced boards can cost thousands of euros.

In the last years, this situation has become to change thanks to the appearance of the *Open Source FPGAs*.

3.3.2 The Open Source FPGAs

Open Source FPGAs are something really new. At the time of writing this work, an *Open Source FPGA* is defined as an *FPGA* that can be programmed, verified, synthesized, and uploaded the bitstream to the board using only *Open Source* tools.

3.3.2.1 The Project Icestorm

The term *Open Source FPGA* appeared three years ago with the release of the work done by *Claire Wolf* (68). Claire did an impressive work of reverse engineering using a cheap development board called *Icestick* (69) that contains an *ICE40 FPGA* done by the company *Lattice* (Fig. 3.13).

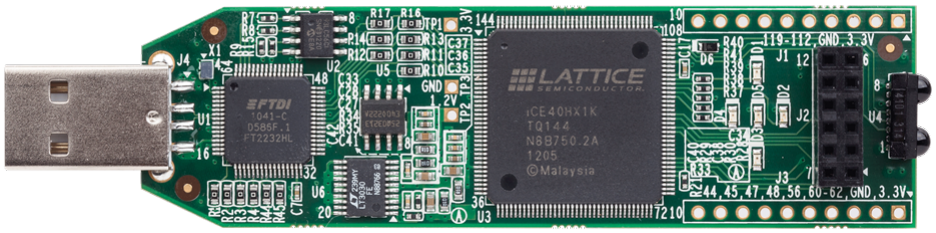


Fig. 3.13: The Icestick, a small and useful development board that was used by Claire Wolf to do reverse engineering and develop the first Open Source FPGA.

Thanks to the tools developed by *Claire Wolf* under the project called *Project Icestorm* (70), in 2015 it was possible, for the first time, to do the whole typical workflow for creating a circuit inside an *FPGA* using only *Open Source tools* and without the need of the official tools of the manufacturer.

As we can see in Fig. 3.14, there are four tools inside the project *Icestorm*. *Yosys* do the synthesis of the Verilog design, transforming the *HDL* circuit to a netlist *.blif* file, where the different components and their connections are described. Once we have the netlist file, the tool *arachne-pnr* do the place and route, preparing these components and connections to be inside the respective *FPGA* that we are using and generating a bitstream in *ASCII*. The tool *Icespack* will convert this *ASCII* bitstream to a proper binary file that will be finally uploaded to the *FPGA* using the tool *Iceprog*.

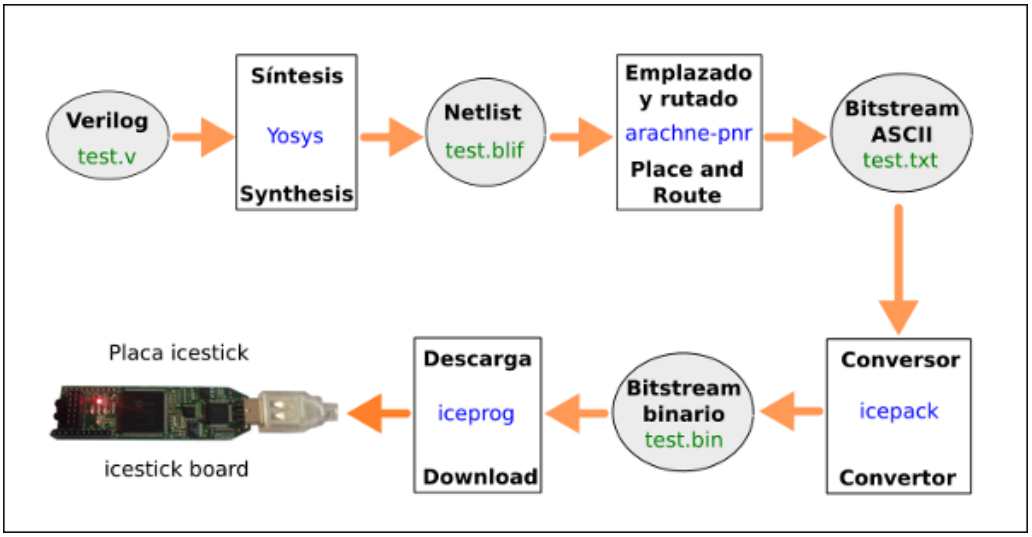


Fig. 3.14: The Icestorm project workflow. By Juan Gonzalez Gomez.

All the tools inside the *Project Icestorm* are easy to install using their official github repositories. The typical workflow described in Fig. 3.14 can be executed using only four commands:

```
$ yosys -p "synth_ice40 -blif test.blif" test.v
$ arachne-pnr -d 1k -p test.pcf test.blif -o test.txt
$ icepack test.txt test.bin
$ iceprog test.bin
```

Listing 3.2: Commands used in the project Icestorm

3.3.2.2 The FPGA Wars Project

After the publication of the *Project Icestorm*, many engineers and people from the *Maker* movement, start to play and experiment with the new tools and *Ice40 FPGAs*. In Spain, an *Open Source* community called *FPGA Wars* born with the idea of expanding and improving the contents and tools of the *Open Source FPGAs* (71).

Fig. 3.15: FPGA Wars. A Spanish-speaking community that creates, expands, and improves the contents and tools of the Open Source FPGAs.

The community was based on the previous idea of the 3D printing *Clone Wars* community (72), and it has been boosted thanks to the work of people like *Juan Gonzalez Gomez, Jesus Arroyo, Eladio Delgado* and so many others. This work has been developed thanks to the help, contents, tools, and above all, support, of the *FPGA Wars* community.

3.3.2.3 An FPGA for everyone: The Icezum Alhambra

With the publication of the *Project Icestorm* tools, all the *Maker* movement around the world started to design new boards that use the compatible *ICE40 FPGA* family. Some of these boards were designed as a Raspberry Pi shield, like the *icoBoard* (73), others have been designed for educational purposes using multiple hardware modules, like the *iCEBreaker* (74), and others like the *TinyFPGA* (75) and *Fomu* (76) were designed to be as small as possible (Fig. 3.16).

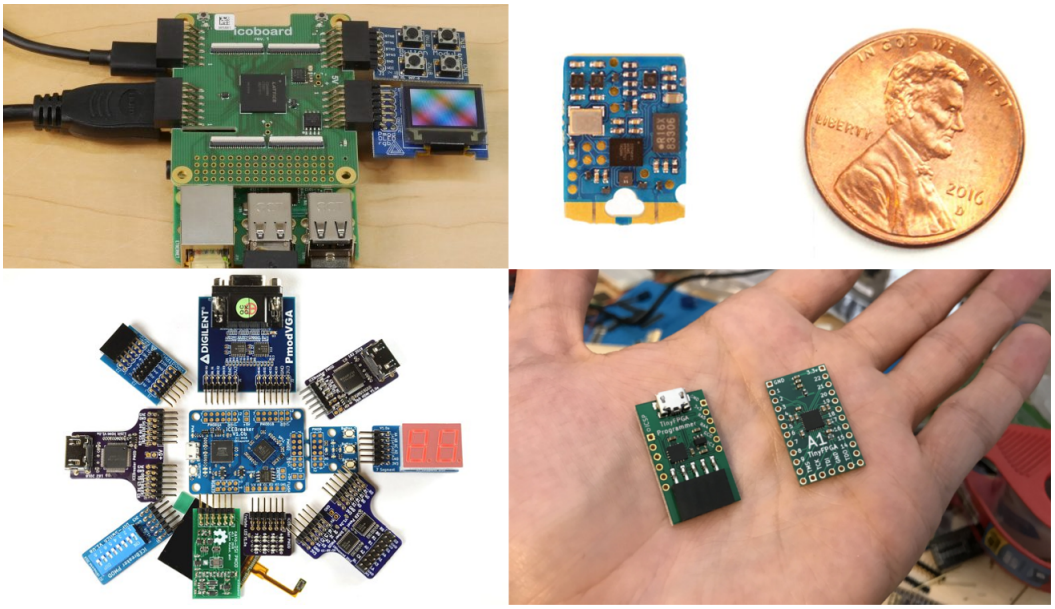


Fig. 3.16: Different Open Source FPGAs. Icoboard (upper left), Fomu (upper right), iCEBreaker (bottom left) and TinyFpga (bottom right).

In Spain, the *FPGA Wars* community designed a board called *Icezum Alhambra* (77).

Fig. 3.17: The Icezum Alhambra, model II. An Open Source FPGA with the layout of an Arduino Uno specially designed for robotics projects.

The *Icezum Alhambra* (Fig. 3.17) is a board inspired in the *Arduino UNO* board layout with the aim of being compatible with the different peripherals and shields of this board. It was designed at Pinos del Valle, Granada, by *Eladio Delgado*, and it has interesting features for robotics thanks to its *Arduino Uno* design. The three array pinout of *VCC*, *GND* and signal, is useful to directly connect simple sensors and motors without the necessity of additional circuitry. It has two integrated buttons that can be used for changing between configurations and modes of the robot and a useful small array of eight *SMD* LEDs to physically show some data like the value of an 8-bits register.

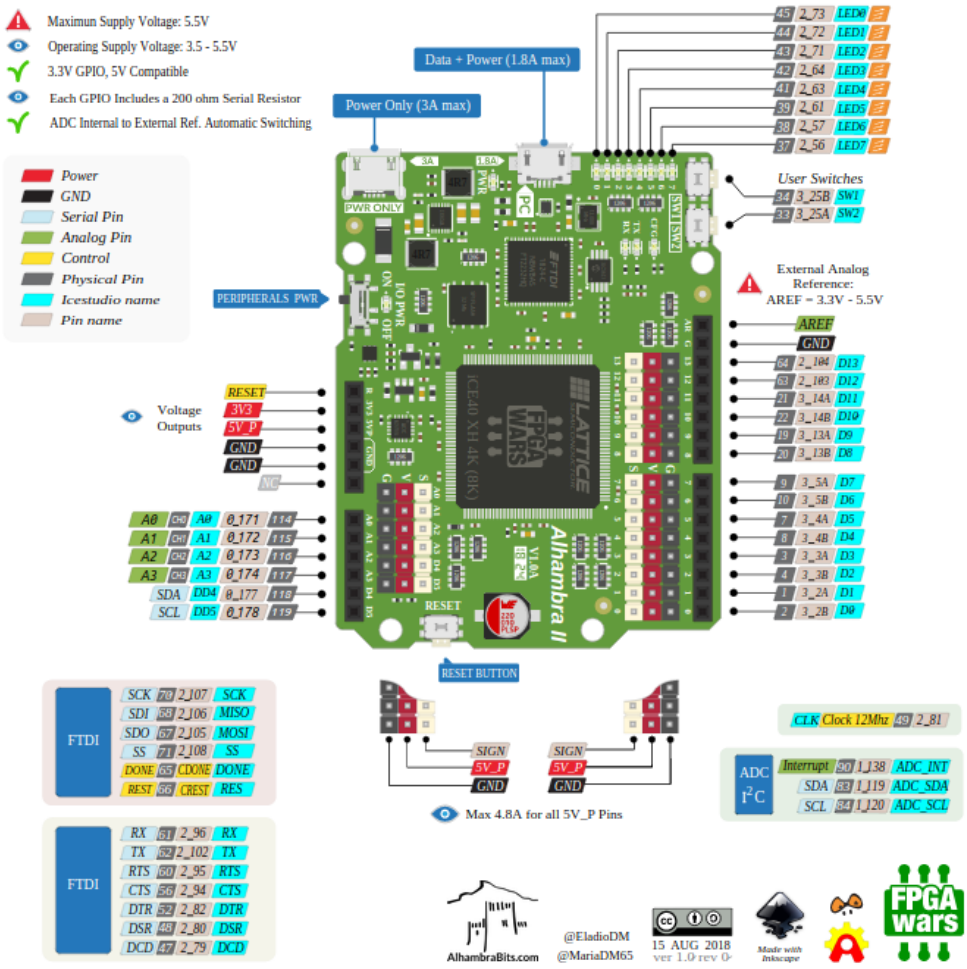


Fig. 3.18: Icezum Alhambra II pinout. The board has interesting features for robotics development, like the three array pinout, the LEDs array, the peripherals power button, and an excellent power management.

The *Icezum Alhambra* has had two iterations, the first iteration has an *ICE40 FPGA* of 1K, while the second version has a bigger *ICE40* of a size of 8K. As it can be seen in Fig. 3.18, the *Icezum Alhambra II* has a power button to switch off the power of the peripherals, very useful to turn off the motors and sensors of a robot without powering down the whole board. Another interesting feature is the double micro-usb connector, one for powering and data, and an additional one for only powering, having a total of 4.8 Amps for powering the pins. The board also has an *ADC* chip to read analog values that communicates with the board using the *I2C* protocol.

The *Icezum Alhambra* I and II have been the main boards used during the development of this project. They are the core of the low-level system and, thanks to the *Open Source* tools developed by the community, designing circuits for this *FPGA* board is as easy as programming an *Arduino* board or a *Raspberry Pi*. In the following chapters, we will discuss how this board has been used to develop different robots and projects to create the proposed bioinspired control architecture.

3.3.2.4 Boosting the Icestorm Tools: APIO

Apio it is a tool developed by *Jesus Arroyo et al* that takes the work done by the *Icestorm Project* to create a multi-platform toolbox that uses static pre-built packages, project configuration tools and easy command interface to verify, synthesize, simulate and upload verilog designs in *Open Source FPGAs*. If the tools developed by the *Project Icestorm*, like *Yosys or Arachne* were easy to use, using *Apio* is even easier to do all the process of installation and setup of the dependencies, boards used and much more. A typical workflow of verification, synthesys and upload to the *FPGA* using *Apio* it only requires a few *bash* commands.

```
$ apio verify
$ apio build
$ apio upload
```

Listing 3.3: Typical Apio workflow

Using these commands, it is possible to create new verilog circuits and upload them in a matter of seconds using a terminal and any text editor or *IDE*. If you are using *Plattformio* and *Atom*, it is also possible to use *Apio-IDE* (78) for integrating the *Apio* tools inside the *Atom IDE* (Fig. 3.19).

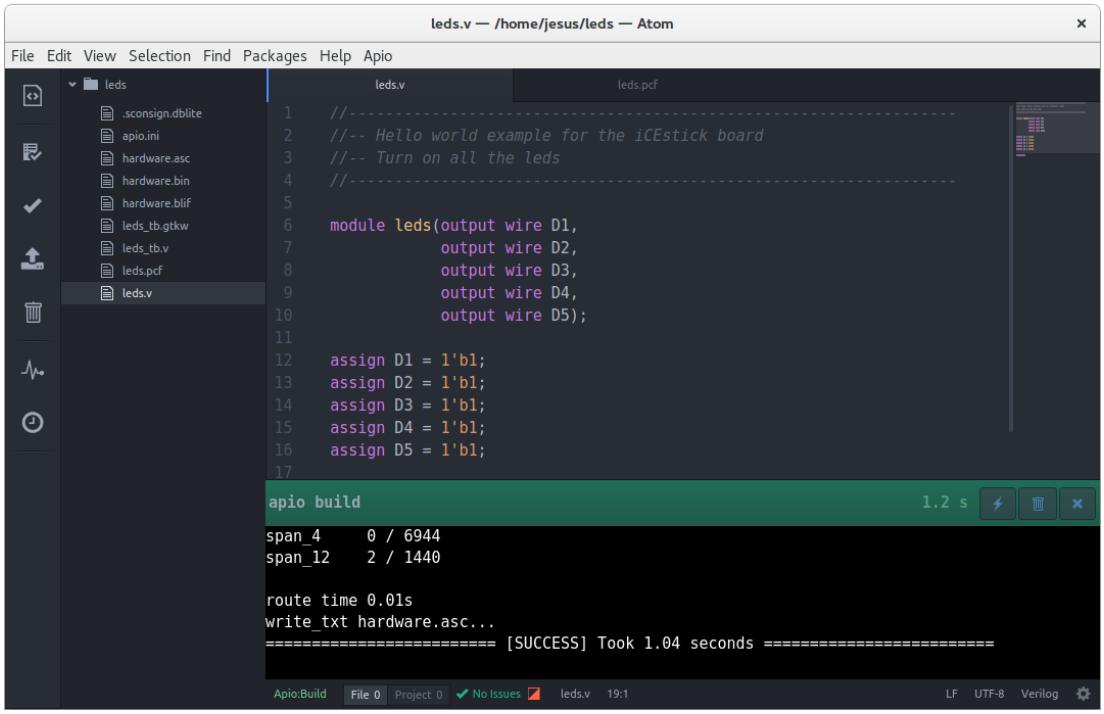


Fig. 3.19: Apio-IDE. An Atom package that integrates the Apio tools inside the Atom IDE.

For the development of this work, *Apio* has been used as the main way to verify, synthetize and upload verilog designs into the *FPGA* of the low level system.

3.3.2.5 Icestudio: A visual editor for Open Source FPGAs

Icestudio (79) is a very interesting program based on the idea of bringing the learning of digital circuits for everyone. It uses graphical blocks that can represent different logical blocks, like logic gates, inputs and outputs, and more complex circuits modules like counters, flip-flops, timers, UART modules, and much more.

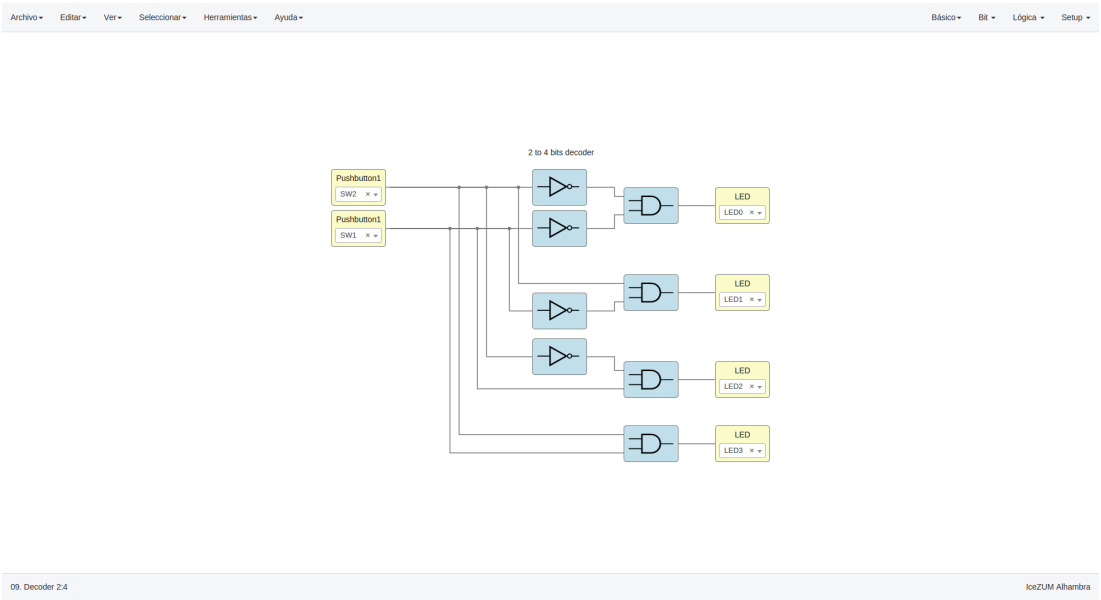


Fig. 3.20: Icestudio. A graphical program for designing digital circuits with Open Source FPGAs without the necessity of any HDL knowledge.

However, the most fascinating thing about *Icestudio* is the mix between graphical blocks and the *Verilog* language. All the graphical blocks are just graphical representations of a circuit that is translated to a *Verilog* circuit before being verified, synthesized and uploaded to the *FPGA* thanks to the tool *Apio*. It is possible to combine graphical blocks with *Verilog* code blocks and create our own blocks, or *Verilog* modules, with their inputs, parameters, and outputs (Fig. 3.21).

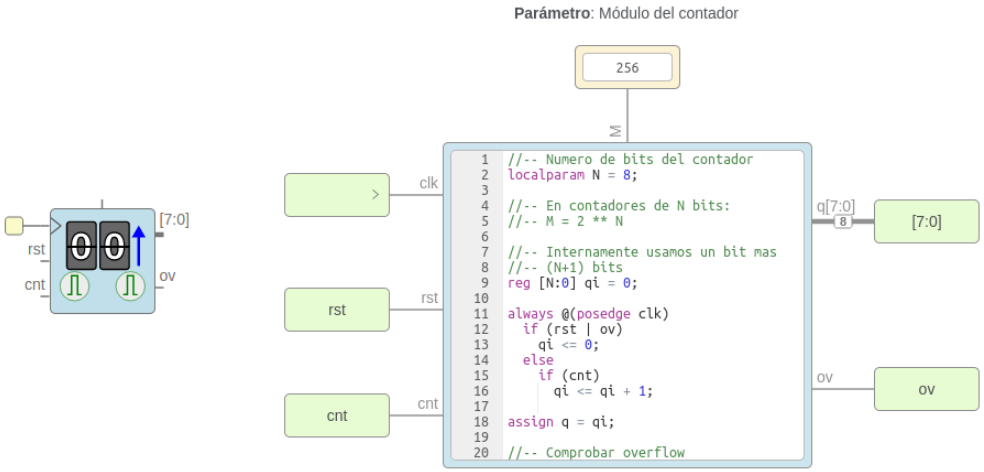


Fig. 3.21: An Icestudio 8-bits counter graphical block and its internal definition wrote in Verilog when we do a double click on the block.

For the development of this work, *Icestudio* has been used for fast prototyping and designing the first circuits of a robot. Thanks to its combination of graphical blocks and *Verilog* code, it is easier to create, modify and test new circuits than just only using *Verilog* code. Thanks to the generation of code that becomes a graphical block that can be inside other blocks, with *Icestudio*, we can decide the level of abstraction we want to work at. Due to this ability, *Icestudio* is starting to be used as a first approach to teach digital circuits design in high schools, fablabs, and even primary schools. In the next chapters, we will see in detail how *Icestudio* has been used to develop the firsts circuits of the different robots and experiments developed in this work.

3.3.2.6 Why Open Source FPGAs?

As we will see in the following chapters, the *Open Source FPGAs* boards and tools have been used as the main low-level platform to implement the system in the proposed bio-inspired control architecture. But why *Open Source FPGAs*? Why not using the traditional and classical boards and tools provided by companies like *Xilinx* or *Altera*?

The *FPGA* industry is more than thirty years old. During so many years, the amazing piece of hardware that is an *FPGA* has been used for only a couple of applications, like professional hardware design. Nowadays, *FPGAs* are having a new raising thanks

to the new golden era of *Artificial Intelligence* and the era of the *Big Data*. However, and despite the knowledge and good products made during the years thanks to these big companies, there are some reasons to use *Open Source FPGAs* in commercial products and research:

- **Autonomy:** The circuit designer can develop their circuits and chips without depending on the desires of the companies. Instead of needing the specific hardware and proprietary tools decided by the manufacturer, the designer can choose which operating system and which tools wants to use to develop their solution.
- **Is not a black box:** it is possible to use the *Open Source FPGAs* tools without worry about how they work. However, if we want we can access to the sources of these tools to see how they work and improve them imitating the successful *Open Source* model of the software industry, where communities and companies are working together to create stable and affordable tools for everyone.
- **Creating the tools that we need:** One of the most exciting things is that we can create new tools for applications not expected by the manufacturer. In scientific research this is so important, it means that for the first time, we can create *FPGAs* tools for specific and not mainstream fields of research, like neuromorphic circuits, more advanced robotics integrations with *FPGAs* and even artistic applications capable of using these technologies in ways that researchers and engineers we are not able to imagine yet.
- **Affordable tools for everyone:** *FPGAs* are in general expensive, the licenses of the privative hardware are also limited and not cheap. For researchers, universities, small companies, and users and hobbyists, the *Open Source FPGAs* can represent an affordable alternative to develop professional and valid solutions in many applications.
- **Educational applications:** Thanks to the *Open Source FPGAs* boards and tools and contents developed by the community like *Icestudio*, the *FPGAs* are starting to be used in non-formal educational environments, like fablabs, to teach digital electronic design for non-electronical experts, and in formal-educational environments like high-schools and even schools. The *Open Source FPGAs*, as

part of the *Hardware movement* are affordable and easy to adapt and use in different economical and social contexts, being perfect to be used as educational tools for everyone.

- **Robots that do not need humans to create their own circuits:** Thanks to the *Open Source FPGAs* we can create for the first time robots that are able to design, verify, synthesize and upload its own circuits without human intervention. This interesting feature, with the privative tools, was hard or even impossible to realize due to the restrictions imposed by the manufacturer, that thinks in the *FPGAs* like an industrial tool with the main purpose of being used by hardware engineers to prototype new microchips and circuits.

As we have seen in this chapter, the tools used for the development of this work are numerous and varied. The robotics fields involve a lot of scientific disciplines and fields of knowledge. Creating the different robots and experiments used in this project had implied the use of different tools to create the hardware, shape, and software of each robot. Today, more than ever, robotics experts have a wide variety of tools and knowledge to be able to create their own robots literally from scratch.

In the following chapters we will explain in deep the proposed bio-inspired control architecture, and how the tools explained in this chapter have been used to create experiments and robots to develop and validate the different aspects of this control architecture.

4

The Bio-Inspired Control System

In this chapter, we are going to explain the robotics bio-inspired control system that is the base of this work. We will talk about its bio-inspiration, explaining how some of the animal nervous systems found it in most of the vertebrates, like the human one, and some insects have been the base to create a robotics control system that has some of the features, advantages, and disadvantages that are present in nature. We will explain the different parts of a complex animal nervous system and how these parts have been implemented in a robotic control system using the technologies mentioned in chapter 3 of this work.

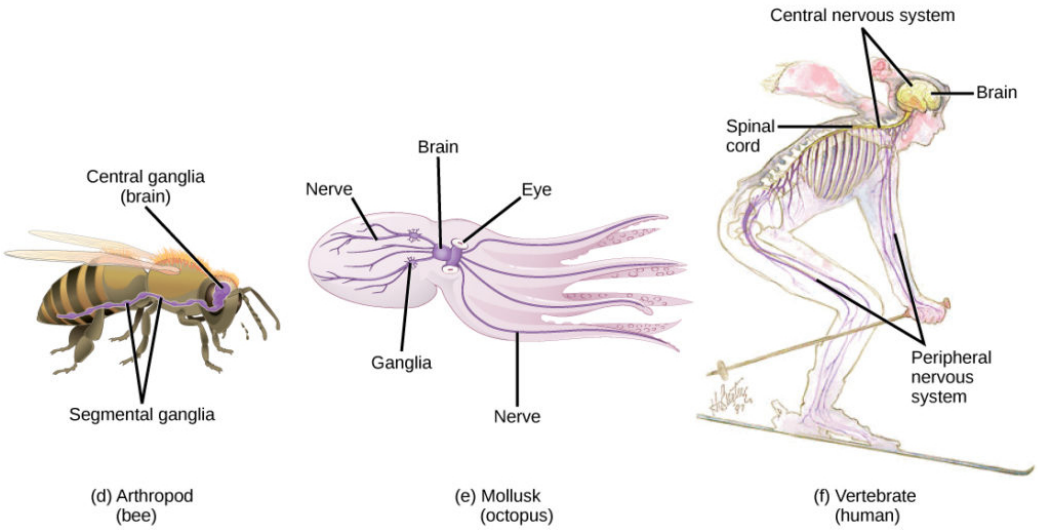


Fig. 4.1: Different animal nervous systems that are present in nature. By Michael Vecchione, Clyde F.E. Roper, and Michael J. Sweeney.

4.1 The Human nervous system

Nature is full of impressive different living forms, but every form of life, even the most basic ones, needs a way to communicate their different cells, organs, and systems to coordinate their different parts to stay alive. During thousands of years, these living forms have evolved from basic groups of cells to complex insects, fish, and vertebrates (Fig. 4.1).

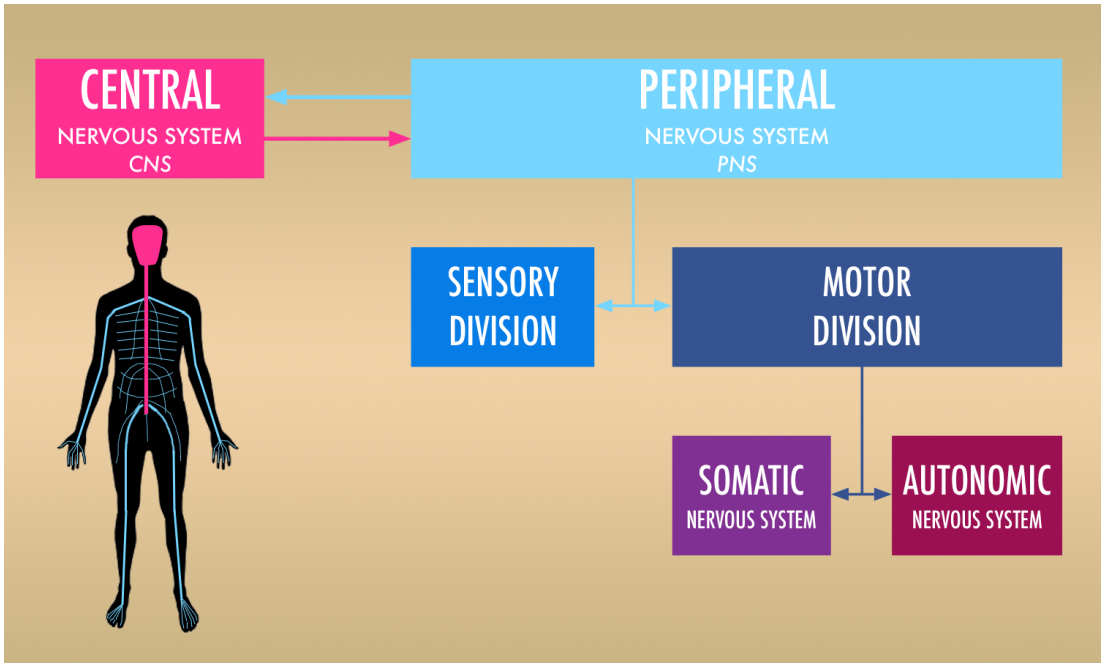


Fig. 4.2: Human nervous system. The *CNS* is responsible of the high level orders, like *moving the arm* or *walking*. The *PNS* is responsible for the basic human senses and the voluntary and involuntary movements.

One of the most impressive systems of communication is the nervous system found in complex animals, where the human nervous system is a great example of how the nervous system have evolved in nature to reach a high degree of performance.

As we can see in fig. 4.2, the human nervous system is formed by two main parts: the *Central Nervous System* or *CNS*, and the *Peripheral Nervous system* or *PNS*.

4.1.1 The Central Nervous System - CNS

The *Central Nervous System* or *CNS* is mainly formed by the brain and the spinal cord. **Is the high level system, responsible of the conscious responses and decisions, and high level orders like *"walk"*, *"take this object"* or *"say hello"***.

All the nervous system are formed by *nerves*, which are big groups of neurons, the base of the nervous system. Neurons are specialized cells with the ability to send signals rapidly and precisely to other cells using electrochemical waves that are transmitted to one or more neurons creating connections that are called *synapses* (Fig. 4.3).

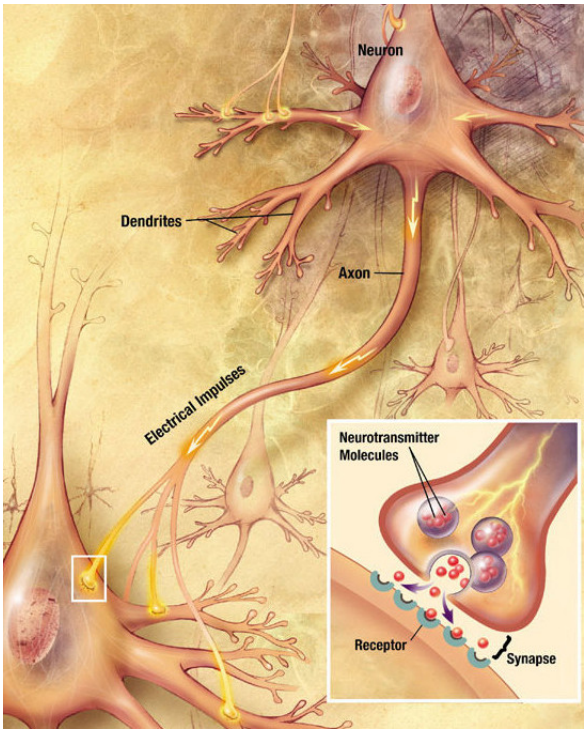


Fig. 4.3: The connection synapses between two neurons. The axon transmits the electrochemical signals that are captured by the different dendrites of other neurons.

When multiple neurons are connected, a neural pathway or neural circuit is created. This concentration of neural connections are bigger in the *CNS*, especially in the brain, where there are large networks formed by millions of neurons that are continuously creating connections that controls our senses, organs, muscles, decisions, memories, feelings, consciousness, and ultimately everything we are.

Despite the big effort and hard work that the science community is putting in the discovery of how the brain works, we know little about it. Although the brain scan technology has helped us a lot to learn more about the brain and its diseases (Fig. 4.4), it is still difficult to observe the different parts of a living brain, looking for significant signals that are not more stronger than the electrical noise signals of the own sensors, and represents the activity of thousands of neurons at the same time. Nowadays, we are always looking at the big picture without being able to observe the little and essential details. It is as if we want to identify a boat on the horizon, trying to know which kind of boat, how many people have, and where it is going in the sea horizon with only our own eyes.

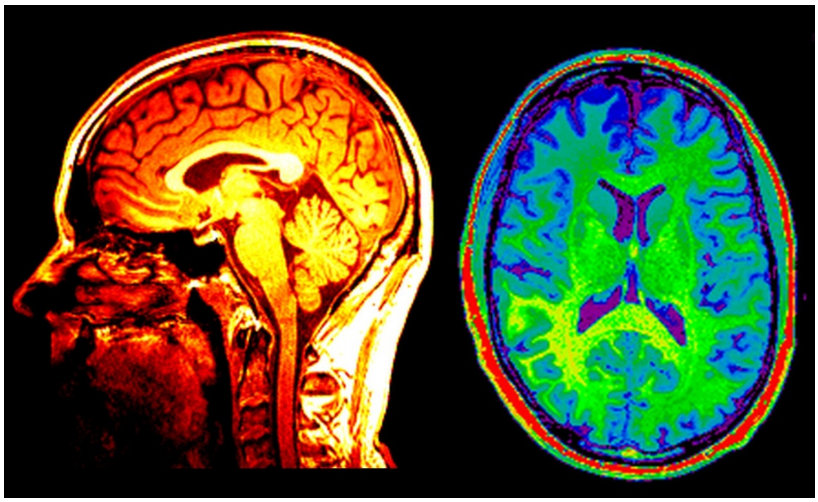


Fig. 4.4: A scan of an human brain or TAC.

Although the field of *Artificial Intelligence* is experimenting a new golden era, we are still using the same mathematical principles and tricks from twenty years ago inspired in the biological interaction between neurons in the animal brains. Artificial Neural networks are a very simplified and abstract model from a natural neural network that can create some impressive expert agents that can learn how to identify objects, animals, create different voices, locomotions patterns and even, create somehow art (80). However, the reality is that we are still far away to deeply understand how works an animal brain. If we want to create robots with a significant level of intelligence, we will need a better understanding of the whole *CNS* system, and how really works a brain.

4.1.2 The Peripheral Nervous System - PNS

The *Peripheral Nervous System* or *PNS* is the rest of the nervous system that is out of the brain and the spinal cord. It reaches to the furthest ends of the body, and it is mainly formed by nerves and ganglia. **The *PNS* is the responsible of the low-level orders, like the voluntary and not voluntary muscles movements, the vitals organs actions and vital reflexes, the communication of the senses, and much more.**

If we retake a look at fig. 4.2 and fig. 4.5, we can see that the *PNS* is divided into two main parts: the sensory division and the motor division.

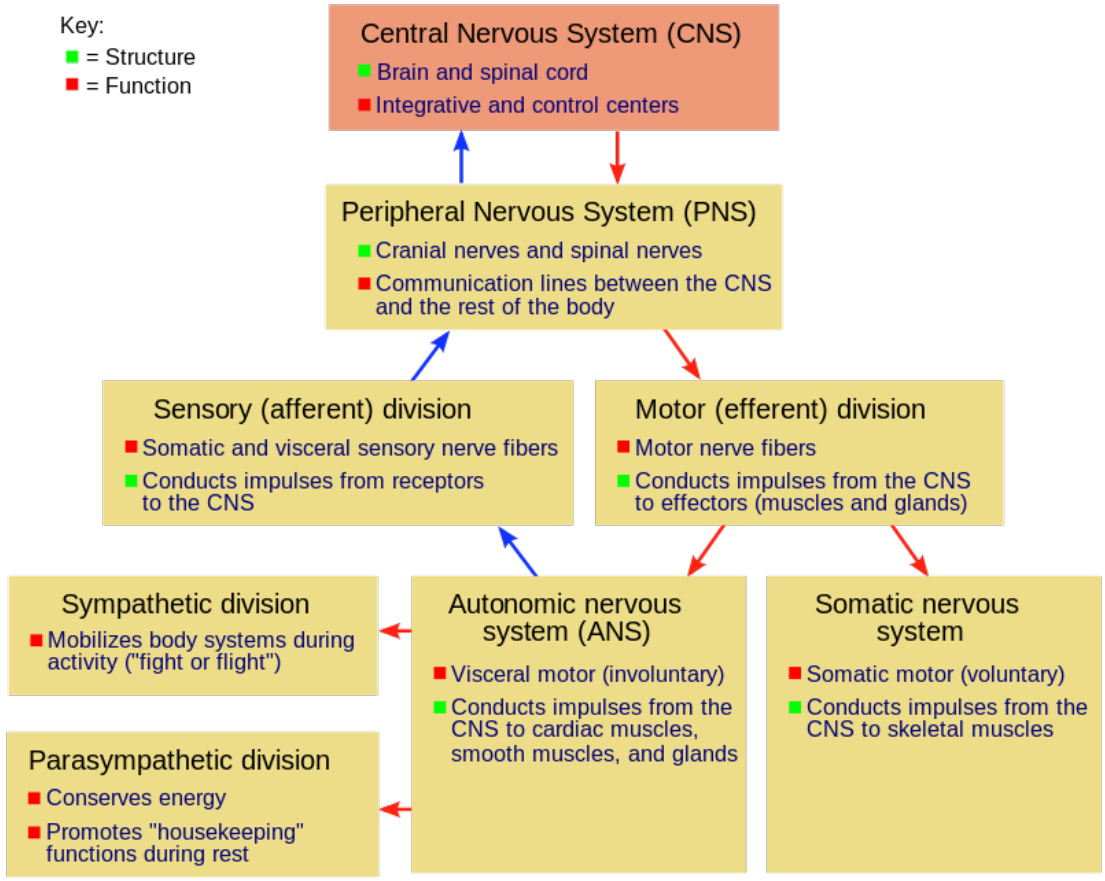


Fig. 4.5: Extended nervous system diagram.

The sensory division controls all the data and communication taken by the internal and external human senses. It basically connects and carries all types of sensory information to the *CNS*, like our big sensing organ: touch, smell, pain, body position

or *proprioception* and vital visceral information about the state of internal organs, like blood pressure and much more.

The motor division is responsible for the control of all the muscles of the body. However, it does not only control muscles, but it also controls glands allowing the body to regulate blood pressure, breathing rate, sweat production, an increase in a hormone, and much more. The motor division is subdivided into two main parts: the somatic nervous system and the autonomic nervous system.

The somatic nervous system controls all the voluntary movements. We are conscious of theses movements in so many levels, and we can voluntary move a specific muscle in a specific way. Moreover, at the same time, it is also responsible for non-voluntary movements like the reflexes that involve the response of a muscle, for example, the patellar reflex.

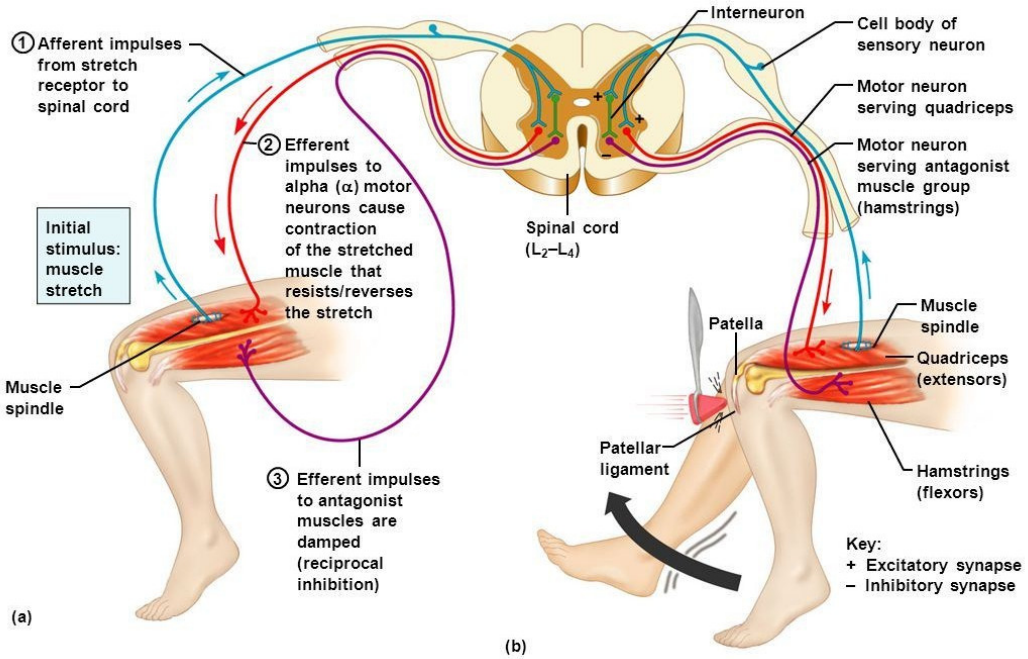


Fig. 4.6: Somatic Nervous system. An example of what happens when we voluntary move the muscle of the leg (case (a)) and how is involuntarily moved by the patellar reflex (case (b)).

On the other hand, the *autonomic nervous system ANS* is responsible for all the unconscious functions that control the body and their internal organs. Some examples are breathing, cardiac regulation, stomach movements, and much more. It is also responsible for the reflexes that correct the proper functioning of the system, like coughing, swallowing, or vomiting.

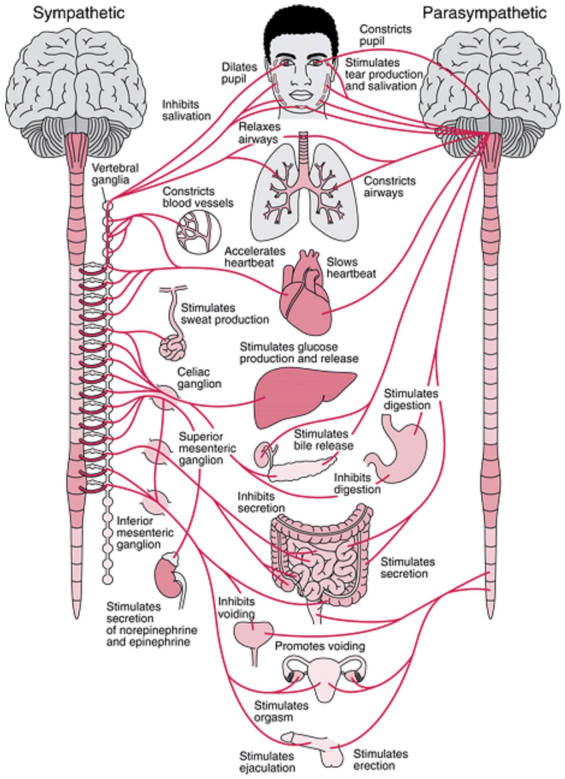


Fig. 4.7: The Autonomic nervous system or ANS is divided into the sympathetic and parasympathetic divisions. Most of the organs are controlled by the two divisions that, in some ways, "fight" for the action or the relaxation of each related organ activity.

The *ANS* is divided in two key parts, the *sympathetic* and the *parasynpathetic* system. As we can see in fig. 4.7, the sympathetic system is responsible for the actions that prepare the body for the action (fight or flight), for example: increase the heartbeat, while the parasympathetic system prepares the body to relax and conserve energy (rest and digest), for example: slow the heartbeats while we are resting. Most of the organs and other systems of the body are controlled at the same time by these two systems, resulting in a kind of constant fight where depending on the situation, one of the systems tries to predominate over the other one.

Thanks to this amazing nervous system present in a simplified or more sophisticated way in most of the living forms, nature has created very different and successful animals with some impressive skills that would be impossible without these efficient and useful nervous system architecture. With the división of high level and low level task between the *CNS* and *PNS*, the brain does not have to worry about vitals questions like controlling all the muscles for breathing or heart-beating to keep us alive, regulating these organs thought high level orders that the *PNS* will follow and convert into the proper low-level actions and movements. Thanks to these task division, the brain can focus on other important tasks like converting all the data received by the sensors in useful information, take decisions based in this information and, in some extreme cases like the human one, create the fascinating thing that we called consciousness thought the abstract thinking.

Fig. 4.8: A person getting abstract data from a smartphone and converting it in information that makes him laugh while he is also listening to music and walking towards his next goal. For us, humans, this is normal but is a little miracle that happens all the time thanks to our nervous system architecture.

Trying to recreate some parts of this successful nature control architecture in robots, can lead us in some exciting experiments where we change the traditional and industrial precise closed-loop control with a bio-inspired control where the way of controlling our robots can drastically change. In the next section, we will explore how we have created a hybrid robotics control architecture that it is inspired in the ones found in nature.

4.2 The Bio-inspired Robotics Control Architecture

In this section, we will talk about the proposed bio-inspired control architecture of this work that is inspired by the complex nervous control system found in insects, complex animals, and humans. We will point out the main features and designing principles that follows this architecture and which technologies we can use to implement it.

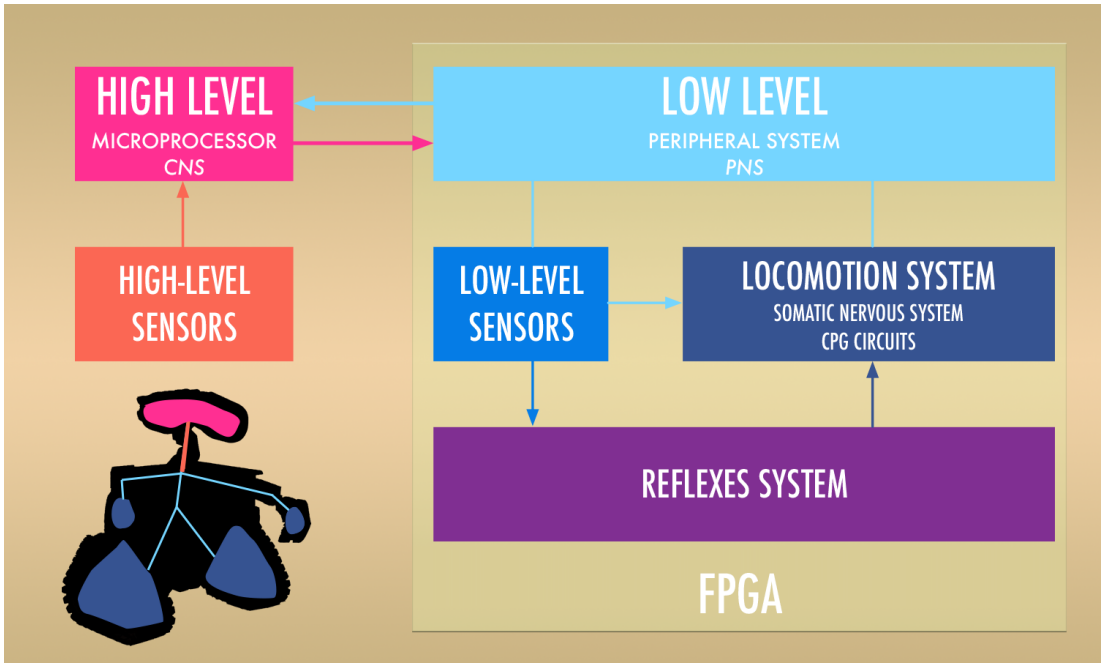


Fig. 4.9: Proposed robotics hybrid control. The main idea is to use a traditional microprocessor as the *brain* or *CNS* and an Open Source FPGA as the low-level system or *PNS* to liberate the high-level control of repetitive and automated tasks.

As it can be seen in fig. 4.9, the proposed architecture is a direct inspiration of the animal nervous system, including the human one (Fig. 4.2). Before starting to explain each part of the architecture, it is important to define what do we define as a *high-level* and *low-level* orders:

- **High-level orders:** We define a high-level order as an abstract order that resumes in one idea a task that can be complex in its execution, or can group many actions in a simple phrase. Humans are very good processing this kind of complex and usually very abstract high-level orders and thoughts that, thanks to the miracle of the language, can be translated into more low-level detailed orders,

or grouped into more complex and high-level orders without losing the ability of compress the vital information of the order in a few words.

Machines, included robots, are not very good understanding this kind of high-level orders where the goal is clear, but the way to reach it is not really defined. Usually, is the human who makes the simplification and details the step by step orders to the machine. However, if we want to create robots that are able to interact with our world, that is richer than just physical objects and is full of abstract ideas fully interconnected between each other thought a complex language, we need machines with the capacity of creating, understanding, communicating to other entities, and translating to low-level orders the abstract thoughts that domains the human world.

- **Low-level orders:** We define the low-level orders as a detailed and specific order to execute an action that, put it with other low-level orders in the right way, allow us to reach a more complex goal. Low-level orders can be translated into more specific and detailed orders that at the same time, can be translated into more specific low-level orders in an almost infinite loop. Humans are complex living forms that are made from millions of cells that follow relatively simple rules coded in their *DNA*. Due to this complexity, most of the low-level orders that happen inside the body are mostly automatic and unaware. It is only when we make something new for the first time, or we try to make and action following very specific orders when we discover the complexity of the execution of detailed low-level orders in a conscious way. In the end, the way to learn a new skill has the objective of repeat the right actions again and again until we reach the desired result that, with more practice and time, will become a part of us that will execute most of the low-level orders in a semi-automatic way.

  On the other hand, machines are excellent following low-level orders. They can repeat the same order with a high level of precision and repeatability, but they are not able to fully understand what is going on and why the low-level orders are executed in some way to reach a big goal. They need a human or another more complex machine to give them the low-level commands in the desired order to reach a high-level order, usually, though an algorithm described in a program or other mechanism.

Fig. 4.10: An example of a high-level order that can be splitted in more low-level orders that, like a Russian doll, can be splitted in more detailed and low-level orders and so on...

A good example of a typical high-level order and how this order needs to be translated into low-level orders will be a phrase like *"Go to the kitchen and give me a beer, would you mind?"*. The human or machine that receive this order has to understand many things in just seconds. First of all, it has to decide if it is going to follow the order or reject it, maybe the order is impossible to reach, maybe requires a lot of time, maybe it is already busy with another order more important or, maybe, just do not want to do it. It has to communicate thought the language or other ways the intention of following or not the order. In the affirmative case, now it needs to translate this abstract order to more detailed low-level orders like "If I sitting on the sofa of the living room, first of all, I have to get up and go to the kitchen", then "I have to go to the fridge, because the human loves the beer frozen, and it is the normal place where I can find a beer", "It has to be a beer with alcohol, not without it, because it is saturday night and this person do not like zero-alcohol beers", "I should also grab a glass for the beer, and I should also open it with a bottle opener", "After that, I should return to the living room with all the objects that I grab and give it to the person". All of these phrases have to be translated into more and more specific and more detailed low-level orders to move the required motors to reach one of the goals using the right sensors for each need and grabbing the data that will be converted into useful information to reach, in

one of the multiple solutions, a goal that was compressed in the relative simple order of "Go to the kitchen and give me a beer, would you mind?". (Fig. 4.10).

The field of study of how a machine can inference meanings and really understand what the human has said in a phrase that does not have all the required information is called *Natural Language Processing* and it is a fascinating field of study where linguistics will help us to create more advanced artificial intelligence engines. This inference using the *Natural Language processing* is essential to decide what is a high-level order and what is a low-level order. Without it, we will need a human to decide, thought algorithms written in code, what is what and how to execute each one of these abstract tasks.

The architecture described in fig. 4.9 proposes the use of a microprocessor as the high-level control or *CNS*. The microprocessor is responsible for all the high-level orders, like *path-planning* or other general and common task that in a robot normally have a high computation cost, like the execution of complex neural network for object recognition and other computer vision algorithms. Only the sensors which are so complex that require a high-level computation, like the computer vision ones, should be directly controlled and connected to the high-level control.

However, the most interesting part is the low-level control. Although the high-level system works in the traditional robotics way using a microprocessor to execute the orders, the low-level control works like a simplified *Peripheral Nervous System* or *PNS*. The architecture has a locomotion system that works as the natural somatic nervous system. This locomotion system is able to generate *Central Patter Generator* signals or *CPG* (see section 2.2) to imitate natural animal movements in a robot. The low-level sensors part is the equivalent to the natural sensory division, where we can find too many different sensors that are simple enough to be directly controlled by the low-level system. Some examples could be boolean sensors, distance sensors like ultrasounds, *IMU* sensors, temperature and humidity sensors, and much more. The proposed architecture also has a reflex control part that takes the data of the low-level sensors and control the actuators to react as fast as possible to an urgent event without the intervention of the high-level control.

As we will explain in detail in the following sections, the low-level control is implemented inside an *FPGA* described as digital circuits that control most of the locomotion and sensor robot behaviors, **improving the execution speed dramatically and**

freeing up the high-level system to be only focused on complex high-level tasks.

4.2.1 The High-level control

The high-level control is the equivalent of a brain and spinal cord (*CNS*) inside the robot. It is responsible for all the high-level decisions that the robot have to take and execute. Some examples are path planning, complex computer vision algorithms, machine learning, task decisions, and much more.

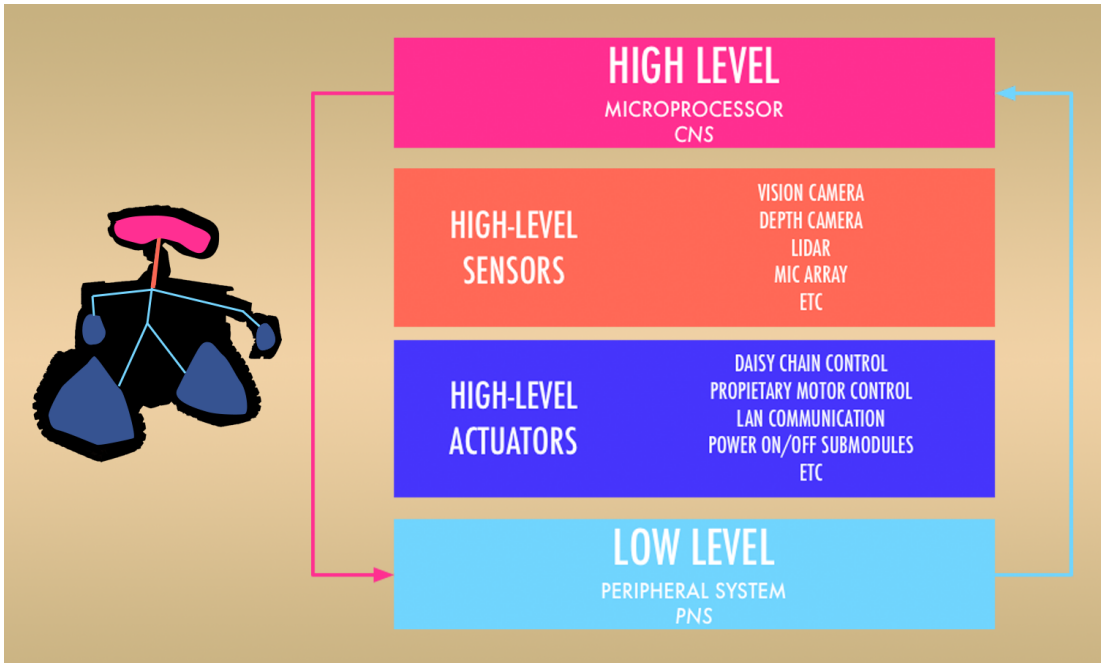


Fig. 4.11: The high-level control works with a traditional microprocessor or computer. It controls the high-level tasks, and it is also responsible for the high-level sensors and actuators. It communicates high-level and abstract orders to the low-level and receives information from it.

The high-level control is also responsible for high-level sensorization and actuators, like sensors or actuators too complex to be managed by the low-level system, because of the communication protocol, the kind of data and its interpretation or the complexity on its control. The control of the computer vision module of the robot or the cloud points of a *Lidar*, could be a great example of a complex sensor that, it could be controlled by the low-level system thought digital circuits inside the *FPGA*, but the

time to do it, difficulty and effort are too much higher than in the microprocessor of the high-level system.

The high-level system is very similar to the other traditional robotics control and can be implemented, as always, in any microprocessor or computer as desired. In the following sections, we will see some examples of how the high-level system has been implemented.

One of the main advantages of the architecture proposed in this work is that we can put most of the tasks in the low-level part. This advantage means that we can put most of the task to work in parallel and at the same time in the low-level system, working with the typically superior speed that the hardware digital circuits have in comparison with software. Thanks to this low-level circuits, we can free up resources for the high-level system to do more complex tasks.

With this architecture, the high-level system will be focused on critical tasks for the correct execution of the robot, doing and receiving high-level orders like "go-forward", "stop" or "there is too much light in the room", without worrying about the low-level details.

There is an important question that needs to be formulated: Which parts and systems of the robots should be in the high-level and which parts in the low-level? What should be done by software (high-level) and what should be done using hardware (low-level)? What is better for each system? Having the functionality without too much effort, doing a program for that, or describing the hardware of this functionality in the low level, improving the speed execution dramatically and allowing it to be executed in parallel with other tasks?

As we can see in the example of fig. 4.12 most of the repetitive task, like periodic locomotion, should go in the low-level system, in addition to any sensor simple enough to be controlled in a relatively easy way using digital circuits. However, although we are defining some recommended rules to follow, it is the roboticist who has to decide what has to go in the high or low-level control. This decissión has to take into account the complexity of the implementation in hardware compared to software, the time and effort required for each level and at the same time the advantages of having the task in the low-level (real circuits, much faster speed, task in real parallelization) or in the high level (simplicity, typically less developing time, more drivers and already done libraries). In general, is a good choice to firstly implement a feature in the high-level

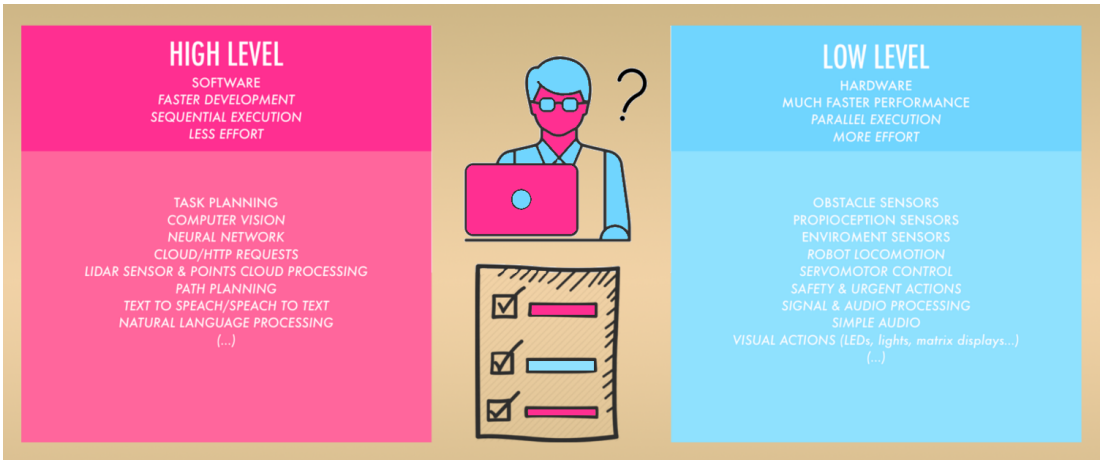


Fig. 4.12: With the proposed architecture, the roboticist has to decide which level control is responsible of each task, thinking always about the advantanges and disadvantages of implementing the task on each control level.

system, using a more traditional approach thought programming, and once the feature is implemented without errors and working properly in the robot, do the equivalent implementation in hardware to free up computing time in the high-level system at the same time that the speed of execution is increased and able to work in parallel with other circuits without losing performance.

4.2.2 The Low-level control

The high-level is a very important part in the correct operation of the architecture, however, the most interesting part is the low-level control. The low-level control works as a peripheral nervous system (*PNS*) and it is responsible of all the low-level, concise and detailed orders of the majority of the sensors and actuators implemented in the robot. As the natural system that tries to imitate, the low-level control must have the capacity of execute most of the task that it does in real parallization and as far as possible. To achieve these crutial features, the system is implemented in form of digital circuits inside an *Open Source FPGA*.

As we can see in fig. 4.9 and fig. 4.13 , the low-level system is the key point of the proposed architecture. It is responsible of very important modules of the robot, like the specific locomotion orders, control of the motors and low-level sensorization, including reflex actions.

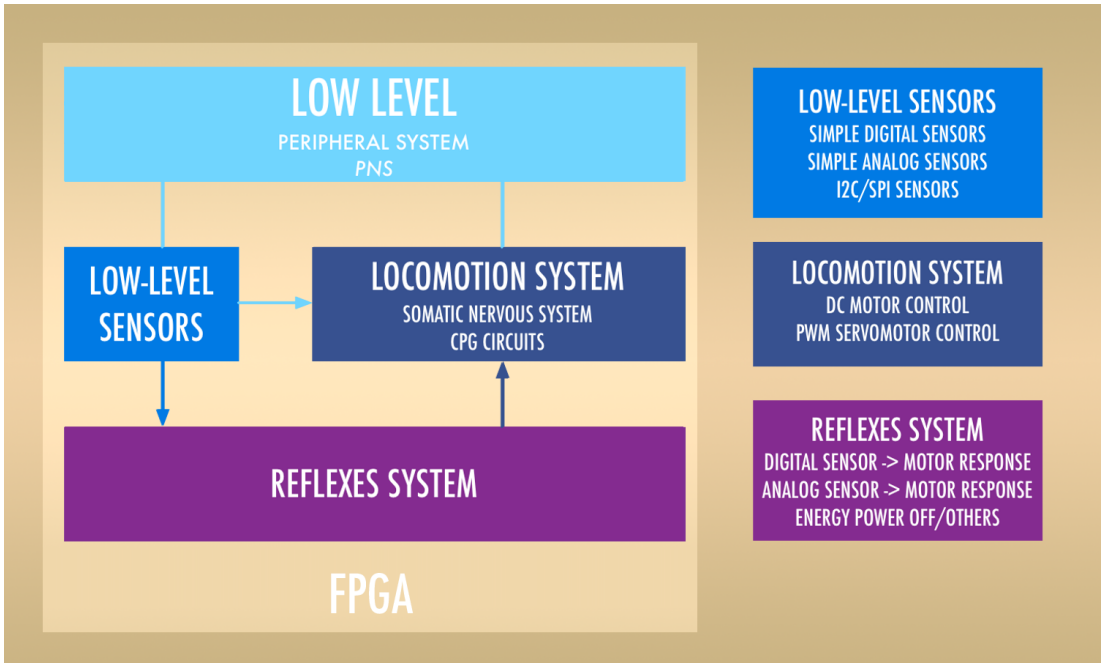


Fig. 4.13: The low-level system controls all the low-level sensorization and locomotion system. The low-level system is able to do reflexes actions, directly connecting sensors with actuators thought digital circuits for automatic, urgent and safety actions.

As we said, the low-level it is implemented in the form of hardware digital circuits which speed it is superior than their equivalent in software and can work in parallel without the problem of saturating the system performance. These difference between the way of working in the high-level system, in which each order is executed sequentially, and the low-level system, where each order it is executed in parallel, it is key to understand the flexibility of the proposed control architecture, and how each high-level task and low-level task are executed (Fig. 4.14).

However, building discrete circuits or manufacturing custom *PCB* boards requires a lot of time, effort, and money, and has the inconvenience that each change, fix, improvement or a new feature, requires new circuits that have to be made by hand or manufactured. The solution is a technology that combines the best features of the hardware world, like excellent performance and real parallelization, with the flexibility and development speed of the software world: the *FPGAs.* The *FPGA* technology allows us to create real physical circuits inside a chip where the circuits can work in parallel and with an insignificant loss in performance compared to a manufactured

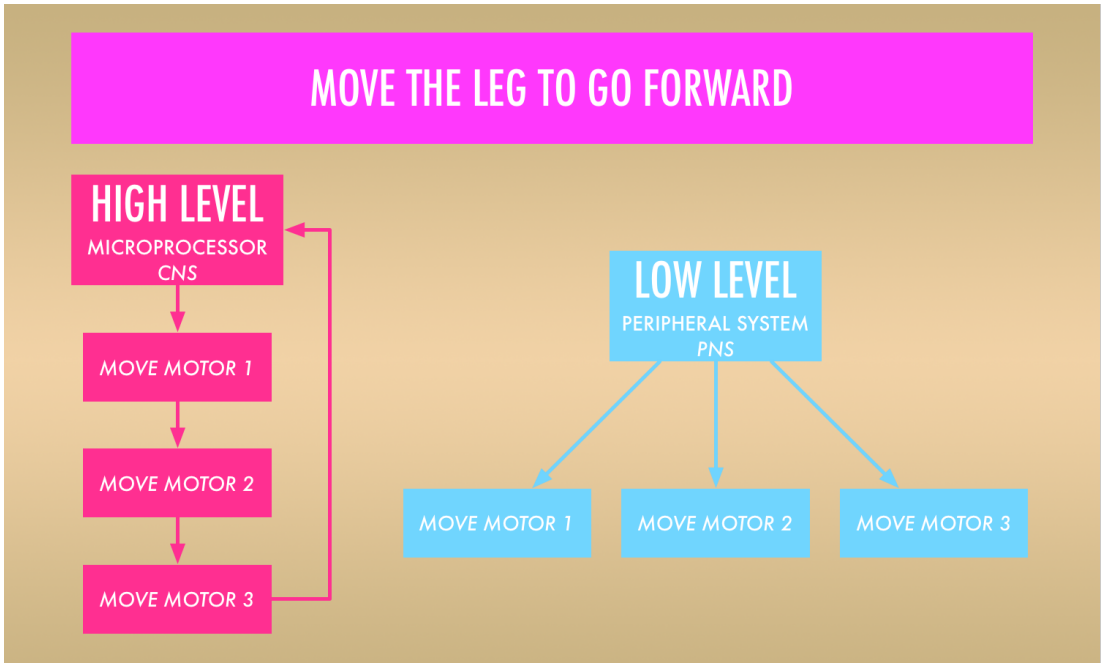


Fig. 4.14: For the task "Move the leg to go fordward" each level control do a different approach. While the microprocessor of the high-level has to execute the movements commands sequentially, the FPGA of the low level system can execute all the movements in parallel, without waitings.

circuit. One of the most important features of an *FPGA* is the ability to add, update, or delete circuits as needed. Traditionally these features were only useful for the hardware designer, who always has to use the manufacturer's tool to design, synthesize and upload a new circuit to the *FPGA*. As we said (section 3.3.2), thanks to the *Open Source FPGAs* tools, we can create robots that are able to automatically generate new circuits using its high-level control, synthesize them and upload them to its low-level control or *FPGA* in a matter of seconds and without human intervention. Using the proposed architecture and the *Open Source FPGAs* tools, we have created robots that can automatically create and modify parts of their own nervous system, changing as needed and in almost real-time the digital circuits, the hardware, that defines their low-level control or *PSN*.

4.3 Tips and rules to implement the architecture

In this section, we are going to comment briefly some of the tips and rules that are useful to implement the proposed bio-inspired robot architecture.

- **New systems can be first implemented in the high-level:** Due to the differences between the high-level system and its software development way of work, and the low-level system and its totally different hardware mindset, it is normally easier to test and develop a *work in progress* new system in the more traditional high-level control. For example, we can test and programming the movement of a leg until we exactly know which kind of locomotions do we need. Once the system is perfectly defined and works well, we can implement it in the low-level system, freeing up the high-level system for other tasks, and obtaining a system that will normally run faster and in parallel with other tasks.

- **If something is periodic and repetitive. It should go in the low-level:** If a task is repetitive or has a periodic beaviour, it should go in the low-level system, that is the specialist in this kind of low-level and semi-concious tasks and movements. The high-level system should control only the more abstract, opened, and complex tasks.

- **Basic sensors should go in the low-level control, complex ones, in the high-level:** Simple sensors are very easy to implement in hardware and can work in parallel and with an impressive velocity and performance. However, it can be difficult to implement some task with complex sensorization like computer vision cameras, in the low-level system. Although implementing this kind of complex sensors in the low-level system is very interesting, it normally requires a lot of more effort than putting them in a computer, where normally we already have the drivers and libraries needed for controlling it.

- **The Roboticist dilemma. The designer has the last word about in which part and how a system has to be implemented:** The architecture is very flexible, and allows the robotics designer the freedom to put a task in the high or low level as desired. Althought there are some important tips and advices, at the end the designer has to choose how to implement one system taking into account the developing time, performance and effort of each task.

5

Robotics implementations and other experiments

In this chapter we are going to talk about the different experiments and robots that have been designed, built, and created to test the bio-inspired architecture proposed in this work to see what it is possible, what is not, and what are the advantages and disadvantages of creating robots following this architecture control.

Creating a robot from zero to one is never easy, but at least, nowadays, it is possible with the right knowledge, some patience, access to affordable digital fabrication tools and cheap electronic components, and some effort. All the technologies that have been used for all the robots and experiments mentioned in this chapter are described in detail in chapter 3.

5.1 You always receive more than you give. An Open Source project

From the beginning of this master thesis work, the idea was to create an *Open Source* project. Due to the project is supported in the tools and previous work that other people has developed in the numerous *Open Source* communities, it seemed fair to create a robotics project to contribute and expand the tools and knowledge of the *Open Source* community.

For this reason, the project has been public available from the beginning at the *Github* repository: ***"https://github.com/jcarolinares/fpga-biorobots"*** (81).

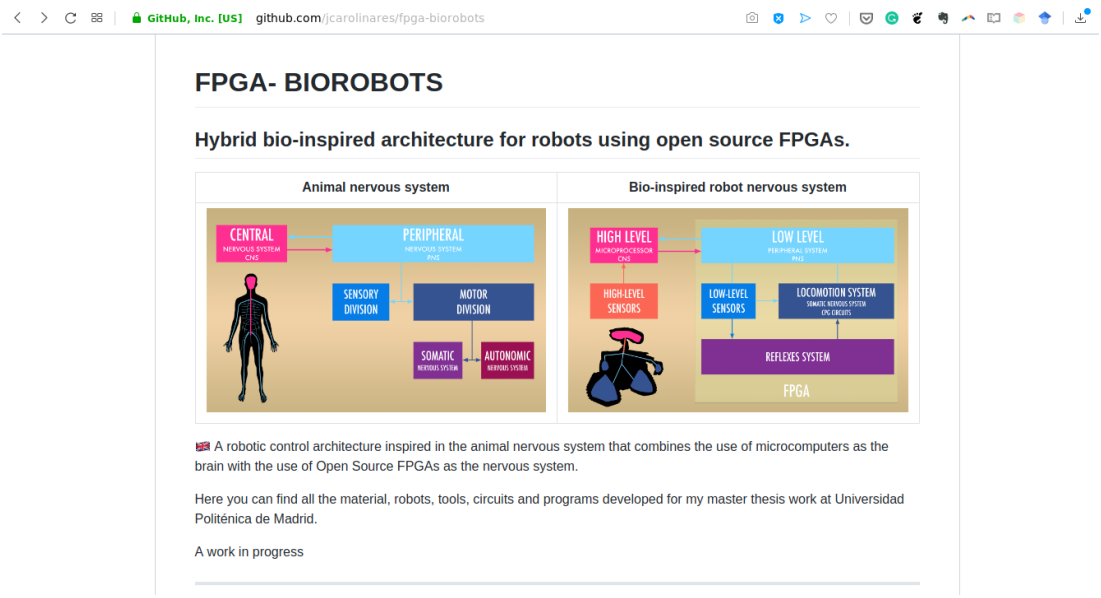


Fig. 5.1: The Github repository where this master thesis work has been developed.

However, the decision of making this master thesis work *Open Source* is not just a personal choice. Nowadays, the technology development is faster than ever. If a company, or a research team, wants to be able to use the new technologies at the right moment and in the right way, create your own solutions and spend the time, effort, and support that this requires is not typically the best option anymore. The best way to do it is by working all together under *Open Source* licenses: companies, universities, and individual people. The *Open Source* projects are not an alternative to the *"serious business"* anymore. They are an integral part of the majority of the technological big and small companies and universities that work altogether to improve these tools for one simple reason: in general, it is cheaper and better for everybody.

This project has been possible thanks to the *FPGA Wars Community* (71). The *FPGA* boards used in this project have been created, developed, and supported thanks to the community. The community has also developed some of the tools used in this work, and sometimes, the own community has developed new features because this or other projects need it. Most of the bugs found it during the development of this work have been solved thanks to the mailing list and the amazing active community help

and support, been this work also a way to create value to the community and bring support to other people and *Open Source* projects.

Although a master thesis work is by definition an individual work, the benefits that give share it with others is surprisingly vast. Thanks to this *Open Source* approach, this project has reached more than we initially believed that was possible, not being, hopefully, a project that will be forgotten in a drawer, but a project that will serve as a basis for other future projects and different applications.

5.2 Architecture experiments and tests

In this section, we are going to talk about the different small experiments and robots that have been done in order to play with the *Open Source FPGAs* boards and tools, and their communication with microprocessors and computers. We will talk especially about *Doodle*, a little educational robot created to figure out if the robotics architecture was possible to implement and which kind of tools are necessary to have a working architecture. We will also talk about other little robots that use the architecture, like *Randofo*, little experiments related with reflexes movements, and even trying to implement the architecture in little humanoids robots like *Zowi*. We will create modules to use *Open Source FPGAs* in robots that use the *Robotics Operating System* (ROS) and other little interesting experiments.

5.2.1 Doodle - An educational robot created to test the bio-inspired Architecture

Doodle (82) is one of the main robots created for this project. *Doodle* is a relatively simple and small robot designed with the purpose of creating the minimal and most simple robot with legs. The idea is implementing and testing the whole bio-inspired architecture proposed in this work avoiding the mechanical, electronics, and design problems that more complex robots have. With a robot like *Doodle*, we can be only focused in implementing and developing all the tools and technologies needed to create the proposed control architecture without worrying about other technical challenges.

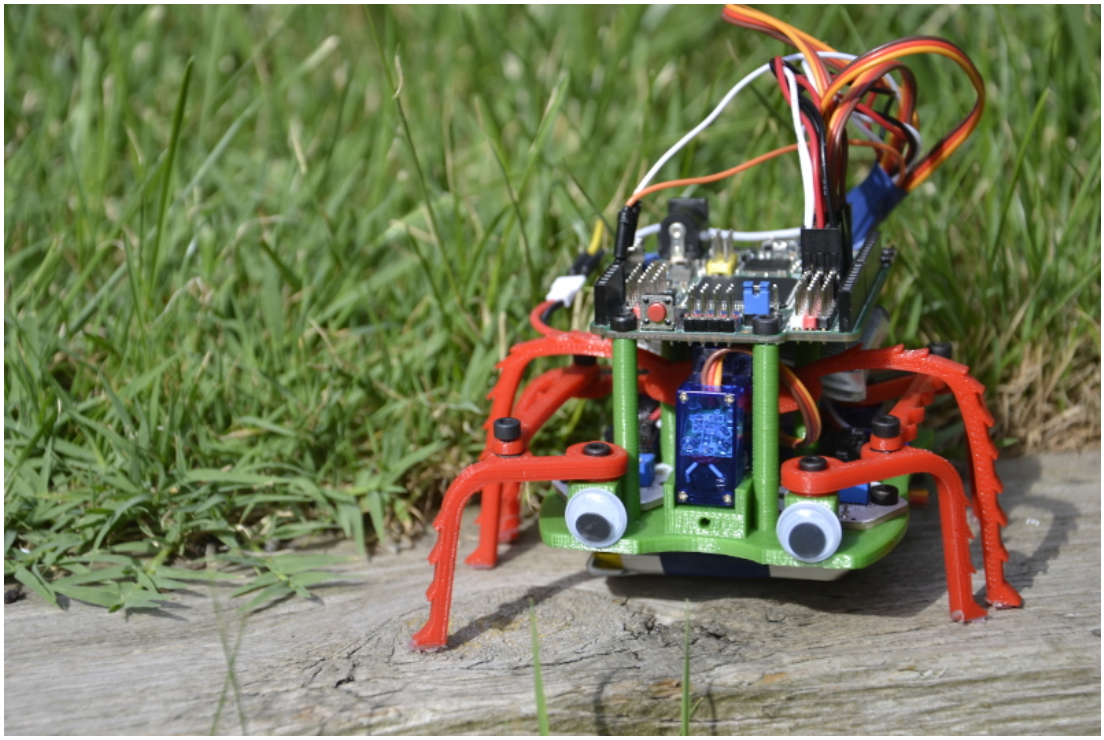


Fig. 5.2: Doodle robot. A simple bio-inspired hexapod able to generate its own CPG circuits in almost real time.

Besides its main goal, *Doodle* has been designed as an educational robot with the potential of been used in non-formal and formal education, with the purpose of teaching basic leg locomotion and programming across biology and other subjects. For this reason *Doodle* is cheap and uses common components. It is easy and fast to build, and can be programmed using different tools and levels of difficulty depending of the age.

5.2.1.1 Design of the bio-inspired architecture

Doodle is an small hexapod robot that moves its six legs with just three degrees of freedom (*DOF*) using three simple mini servomotors. With only three *DOF*, the robot is able to go forward, backward, make turns and curved trajectories.

The robot's mechanism is simple: one motor controls the left legs lateral movement, other motor the right legs and the last motor controls the central legs, lifting each side alternatively to elevate or putting on the floor the left or right side during the locomotion.

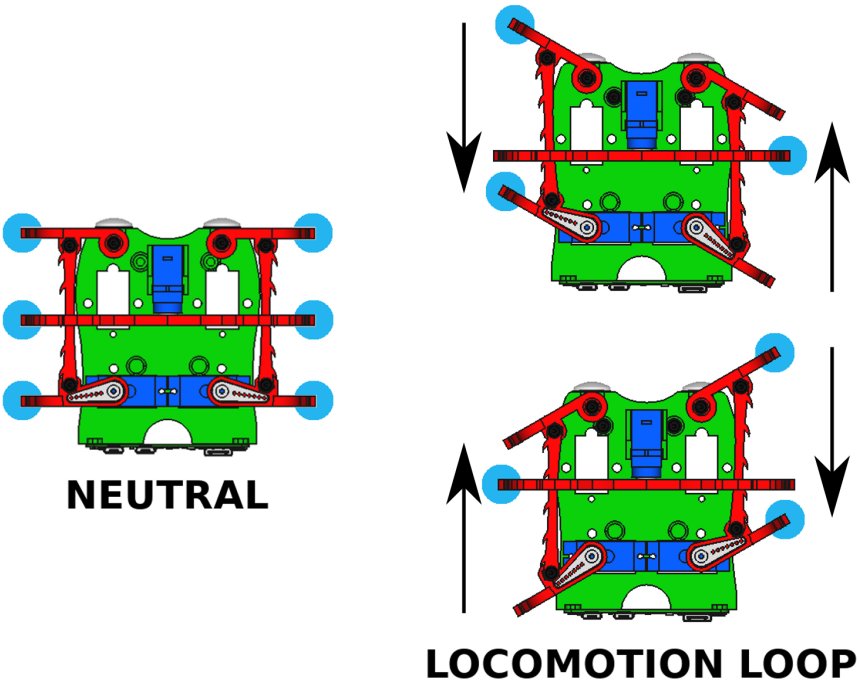


Fig. 5.3: Doodle's locomotion. The blue dots represent the legs in contact with the floor. The movements are the basic movements formed by triangles that are typical in hexapod animals.

This basic locomotion to go forward (Fig. 5.3) is the most common in hexapods. It works like the oars in a canoe. While one part is rowing from front to back inside the water, the other side has raised the oars to put them from back to front, ready for the next movement.

As we can see in fig. 5.4, *Doodle* has a relative simple control schema. However, this control has all the fundamental elements of the proposed bio-inspired architecture of this work.

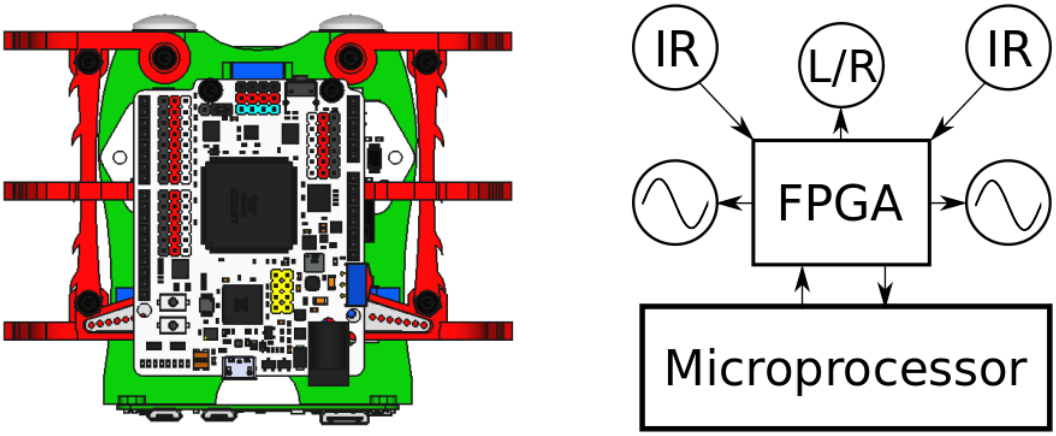


Fig. 5.4: Doodle bio-inspired architecture. A simple structure that has all the essential components of the proposed architecture of this work.

The robot has two main parts: the high-level control or microprocessor, and the low-level control or *FPGA*. The microprocessor is a *Raspberry Pi Zero W*, a small computer perfect for prototyping small robots. The low-level system is implemented inside the *FPGA Icezum Alhambra* board (section 3.3.2.3). Both systems are connected thought two micro-USB port. The board, specifically designed for robot prototyping, follows the design of an *Arduino UNO* board, being compatible with most of the sensors, actuators, and *Arduino* shields that are available on the market. While the high-level system works on high-level tasks, like as we will see forward, creating new circuits and locomotions for the robot, the low-level system is responsible for all the low-level commands, like the control of the motors and sensors. *Doodle* is a very simple robot, with only three servomotors. The central one, marked as *"L/R"* in fig. 5.4, only has two positions: raising left legs or raising right legs. However, the motors that control the movement of the legs (marked with a wave signal) have a more interesting approach: the use of *Central Pattern Generators* or *CPG* (section 2.2).

The *CPGs* used in *Doodle* are very simple oscillators signals, normally triangular or sinusoidal signals (Fig. 5.5). Although the movements of the robot are very simple, and the legs only move back and forward, there is a huge difference between moving

the robot with discrete orders ("move left legs forward to this position and move left legs backward to this other position") and the use of oscillator signals. With the use of *CPGs*, the robot takes better advantage of the contact with the floor, resulting in longer distance traveled with smoother and more natural movements (83).

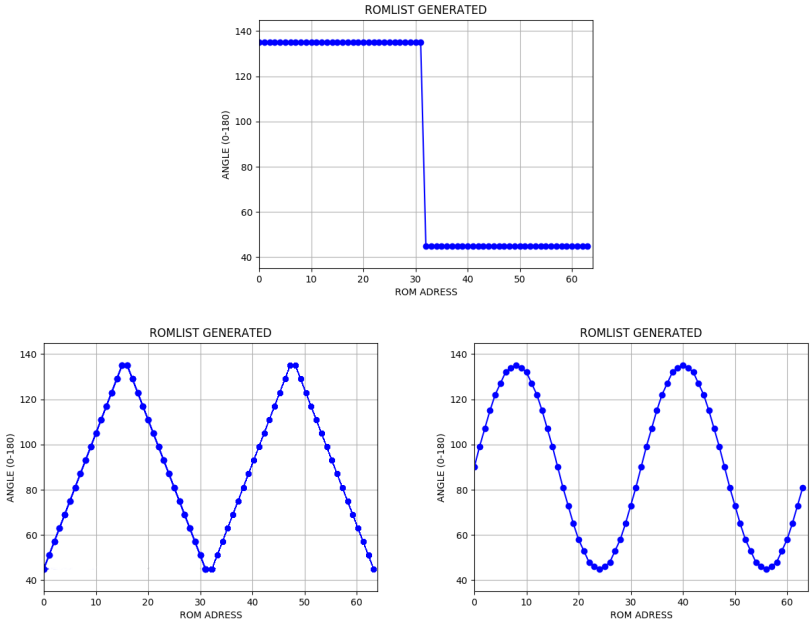


Fig. 5.5: Different signals used for the locomotion of Doodle's legs. The most efficient is the sinusoidal one, followed by the triangular one.

Attached to the *FPGA* of the low-level system, they are also two digital infrared sensors (*IR*) to detect black or white surfaces. As we will see in the following sections, these infrared sensors are cross-connected to the left and right motors, inhibiting the movement signals of the legs. Typical applications are for example follow a black line, an application not very common in robots with legs, or preventing the fall from a table. However, the most interesting point is that, due to the implementation of these simple digital sensors in the low-level system, all the behaviours are inside the *FPGA* in the shape of digital logic circuits. This means that the sensors are going to actuate with an impressive speed, at the same time and with total control over the servo motors, even above the orders of the high-level system. Resulting, as we will see in more detail later, an interesting approach to develop reflexes movements for safety situations.

5.2.1.2 Design of the electronics

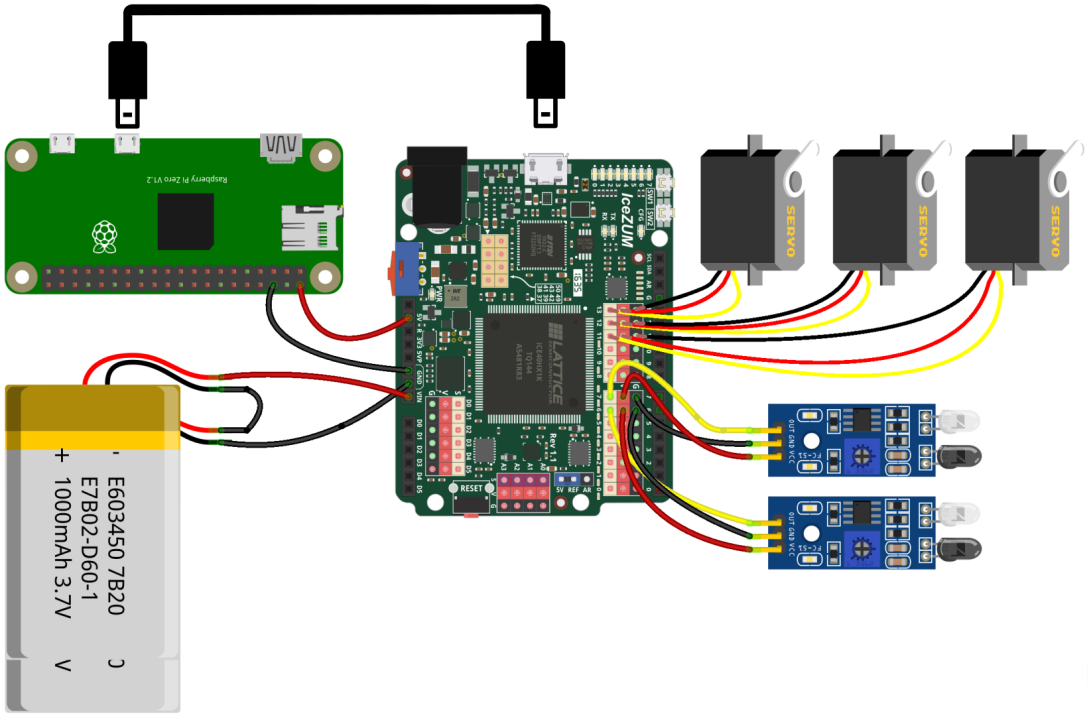


Fig. 5.6: Doodle hardware and connections. Created with Fritzing with the Icezum Alhambra resources created by Jorge Lobo.

Doodle has an easy electronic setup. As we can see in fig. 5.6 is composed of the following elements:

- **Raspberry Pi zero W:** A popular chip micro-computer that acts like the high-level system or *CNS*. The computer normally runs *Linux* and it has an integrated Wifi and Bluetooth connections, a handy *GPIO* pins and a micro-USB used to communicate with the *FPGA* of the low-level system.

  Although the speed of this micro-computer is not too high, it has a very small form factor, being perfect for a little robot like *Doodle*.

  The board can be powered thought the *micro-usb* cable that connects the *Raspberry Pi zero W* with the *Icezum Alhambra* or connecting the 5V pin output of the *Icezum Alhambra* and ground to the *Raspberry* GPIO power pins.

- **Icezum Alhambra (model I or II):** The *Icezum Alhambra* (77) is an *Open Source* board compatible with the *Open Source FPGAs* tools. The shape of the board is the same as the *Arduino UNO*, and the core of the board is a *Lattice ICE40 FPGA* (84). Depending on the version of the board, the *FPGA* has a size of 1K (around 1000 logic cells) in the first version or 8K (around 8000 logic cells) in the second version. These sizes are not big for the standards of the *FPGA* world, but are more than enough for the robotics applications developed in this project and even to implement a *RISC-V* microprocessor on it (85).

- **3x SG90/SG92R servo motors:** One of the few requisites of *Doodle* is that it has to be as cheap as possible. If later we want to build more complex robots, with more *DOF* and motors, it is important to find out what the motors can and cannot do and with which prize. Some of the most used servomotors in the hobbyist and *Maker* community are the *SG90* model. They are cheap, small, typically have a torque of 2.5kg/cm and works in a 180º degrees closed loop that can be easily controlled via *PWM* signals. For *Doodle* we use three of these motors having a good torque and movement speed. We also experiment with the *SG92R* version, which has even more torque but more current drawing.

  The servo motors three-pin jumper connector can be directly connected to the *Icezum Alhambra* three-pin connector rack. Saving a lot of cables and additional circuits to power and control each motor.

- **2x Digital infrared sensors:** In this case, we have used two simple sensors made by following a line, composed by an infrared emitter and receiver, plus a small control circuit to convert the signal from analog to digital which threshold is controlled using a potentiometer.

- **Short micro-USB to micro-USB cable:** The *Rasperry Pi Zero W* and the *Icezum Alhambra* have a micro-usb port for data and power. There are multiple ways to connect both boards, thought simple GPIO pins digital signals, using some GPIO pins as an UART port or the chosen one, using the micro-usb ports plus the integrated *FTDI* chip to make an UART communication between both boards.

In the end, the connection is easy and straightforward, requiring only a short micro-usb cable to micro-usb cable to connect the high-level system with the low-level system.

- **2x 1S Lipo batteries:** Power management is always a big headache. Batteries are heavy and require space. More space, more capacity and power are needed it. As we will see in the next section, for a small robot like *Doodle*, where all the available space is used by the legs, the selection, and placement of the batteries is a non-trivial challenge.

  Doodle uses two *Lipo* batteries, each cell of 1S, 3.7V and 240 mAh. Putting together in series, the resulting power supply is 7.2 volts with an autonomy of 480mAh. *Doodle* was designed using the first version of the *Icezum Alhambra* and the robot can be easily powered thought the barrel jack of this version or the *GPIO* pins.

  Alternative external power supplies, like AA/AAA batteries or powerbanks, can be used to power *Doodle*, but more additional space will be required.

The updated list with links to buy all the components can be found at the *Github readme* of *Doodle* (82). Without taking into account the price of the two main boards (around 60 € for both) the total for the rest of the components without replacements are around 15-30 €, having a prize of approximately 90 € in the case of the full pack, and a prize of 25 € per robot in the case that we already have the boards for educational uses in other projects.

The detailed budget of the robot, including replacements and additional elements, can be found at the *Appendix A*. These prices have been applied for the workshop of *"Robotics and FPGAs, building Doodle"* done in the Oshwdem Faire 2018 (86).

5.2.1.3 Design of the body

Creating a robot from scratch is always a challenge. *Doodle* was born inspired in other similar robots with the same locomotion principles, but which design, build difficulty and replicability were not enough for the characteristics that were sought in the robot (87). When a robot is created with the goal of being public and be used by others, the design has to be as clean as possible, especially in the case of educational robots.

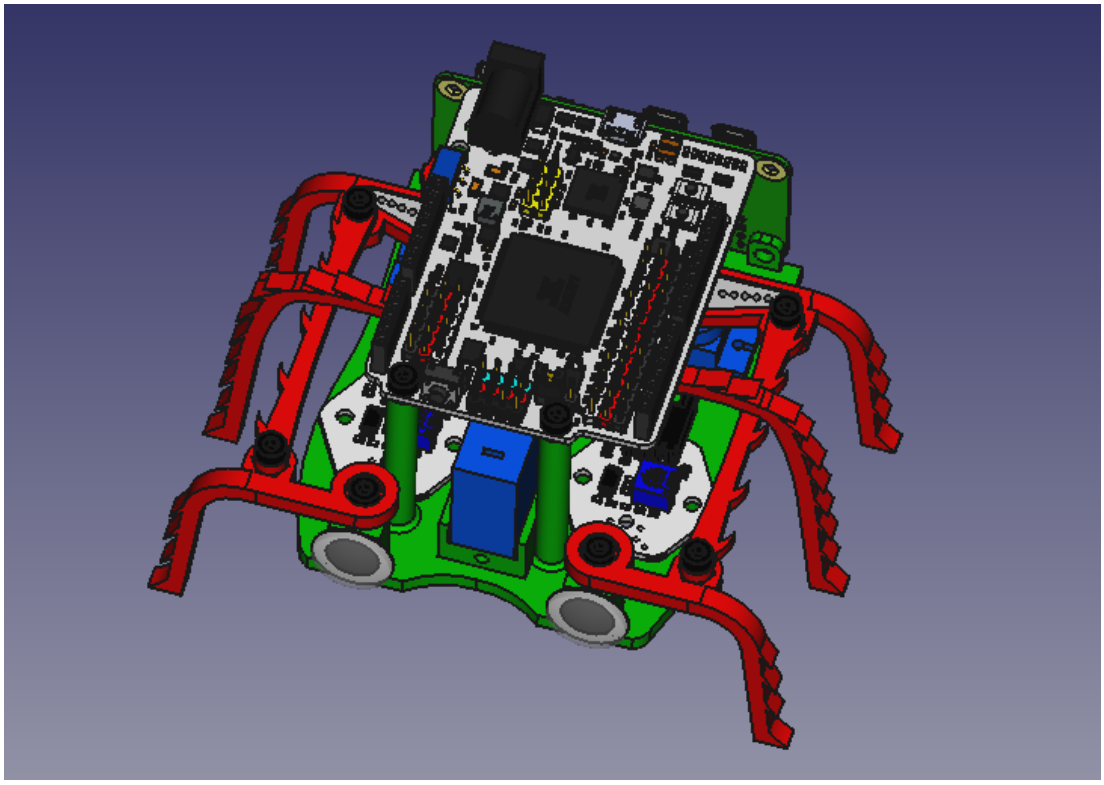


Fig. 5.7: Doodle design made in Freecad. A robot like Doodle has to be easy to modify, 3D print, and build.

The design has to be simple enough or at least well documented. The robot should use common components, like common motors, electronics, and standard screws, and should be easy to build and replicate in another part of the world with a different 3D printer. All of these aspects have been taken into account to create the robot *Doodle* design. *Doodle* has been created using the traditional *sketch and extrusion* paradigm design thought incremental iterations using the tool *Freecad* (section 3.2.1).

When we start a new robot design, one of the key aspects is to know which not 3D-printed components has the robot. In the end, the body of a robot is, somehow, just a holder to keep all the elements, like the electronics, motors, and other parts in the desired place. The shape of a robot mark its features and interactions with the world, and the distribution of its different components are key to create this shape.

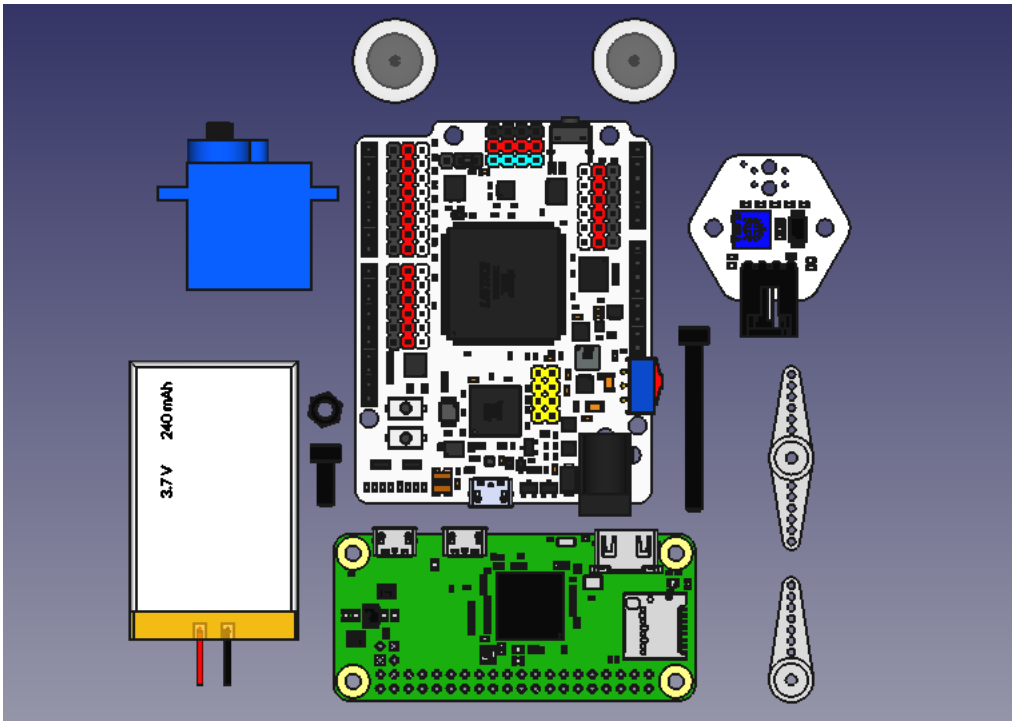


Fig. 5.8: Doodle auxiliar CAD files like servo motors, the two boards, screws, IR sensors, batteries, and googly eyes.

For this reason, one of the fundamental aspects of a good CAD design is to add to the program and use from the beginning all the auxiliary components CAD files, like motors, screws, and boards used in the robot. These components will simplify the difficulty of the design and save time and errors. If one of these auxiliar CAD designs cannot be found, sometimes it is worth it to spend some time creating the component.

Starting with the base of the robot is in the case of *Doodle*, a good idea. In robots with legs, the distribution of the different components is critical for the correct locomotion of the robot. Designing our parts around the center of coordinates is important to create symmetric parts faster and avoiding errors.

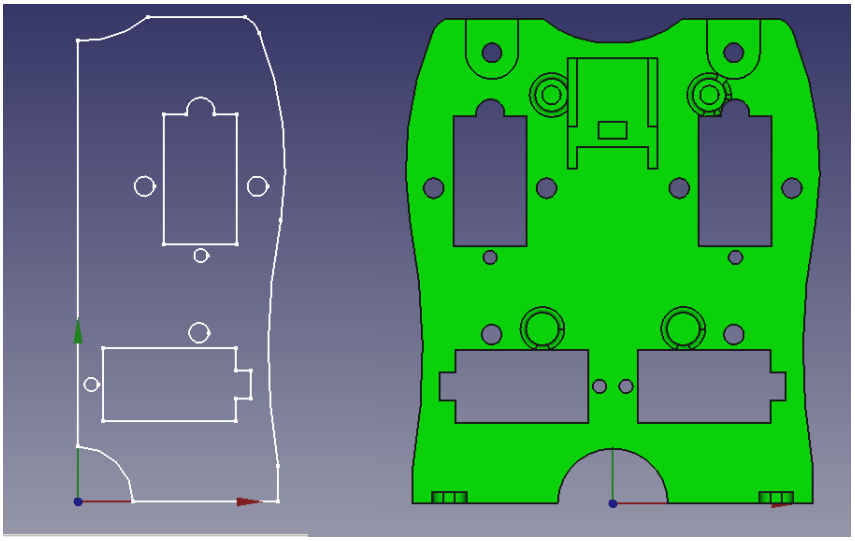


Fig. 5.9: Doodle base designing process.

The basic base shape of *Doodle* has been created as a symmetric part where only half of the design is created, and the other part has been mirrored (Fig. 5.9). The asymmetric components are added after the mirroring.

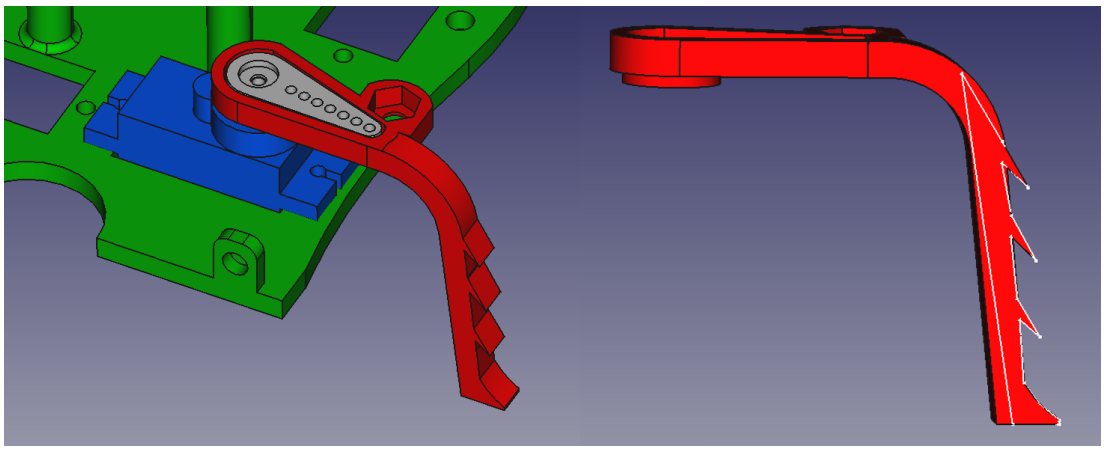


Fig. 5.10: Back leg design attached to the servo. It also has "insect hairs" for aesthetics.

The back legs have been designed taking into account the servo motor horn to attach the motor axis to the legs. The leg can be easily 3D printed with little supports. An additional visual feature has been added to the legs to reassembly the hairs of an insect (Fig. 5.10). The back legs are critical to the correct working of the robot and have been the part with most iterations of the whole design.

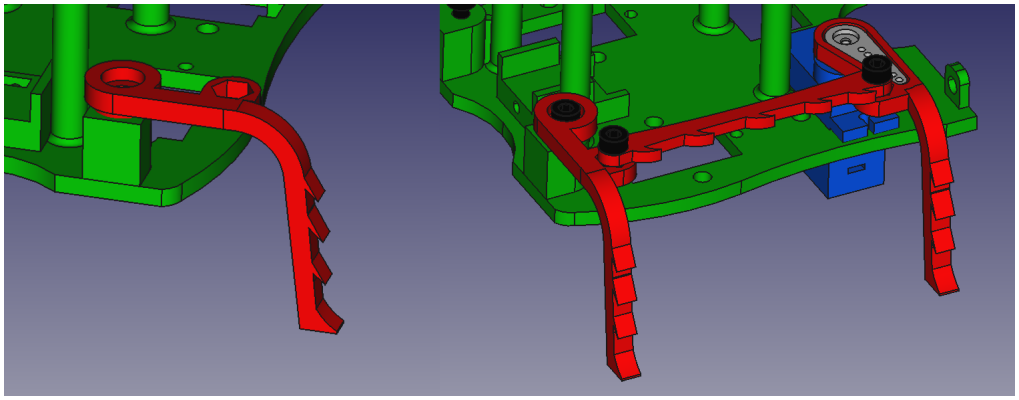


Fig. 5.11: Doodle front leg design. The front legs are connected to the movement of the back legs using an additional chain leg (right).

One of the most interesting features of *Doodle* is that it is a hexapod that controls all its legs with only three servo motors. This is possible thanks to the design of the front legs, that are free and rotate around a large screw attached to the base. To connect the movement of the back legs to the front legs, an additional *chain leg* connects the two legs (Fig. 5.11). One of the most interesting features of this design is that the *chain leg* connect the two legs using embedded nuts and screws in the top part of the legs. Thanks to this design, the whole *chain leg* can easily escape from its normal position in the case of a dangerous mechanical movement of the legs, protecting the whole mechanism and avoiding breaking the legs.

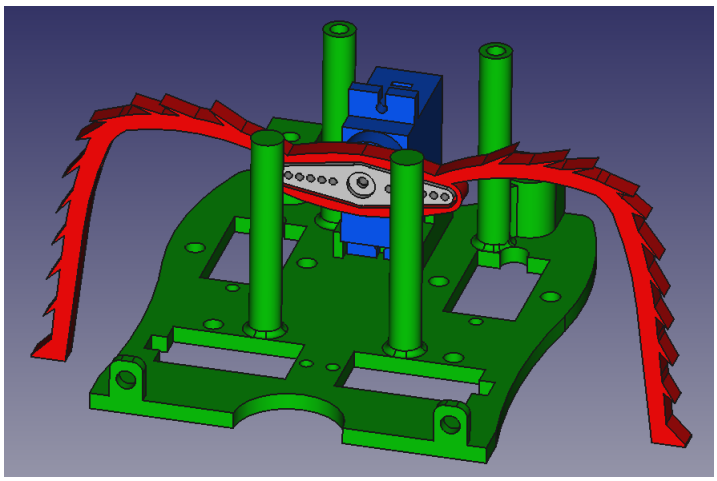


Fig. 5.12: Doodle central leg design. The central leg is key to properly raise and lower the legs of Doodle.

The central leg has been designed like the base, designing only half of the leg and mirroring the other one. For this case, the servomotor axis is working in an orthogonal plane regarding the robot base, and the servo is attached by pressure to the base using a M2 large screw and nut that keeps it in its position (Fig. 5.12).

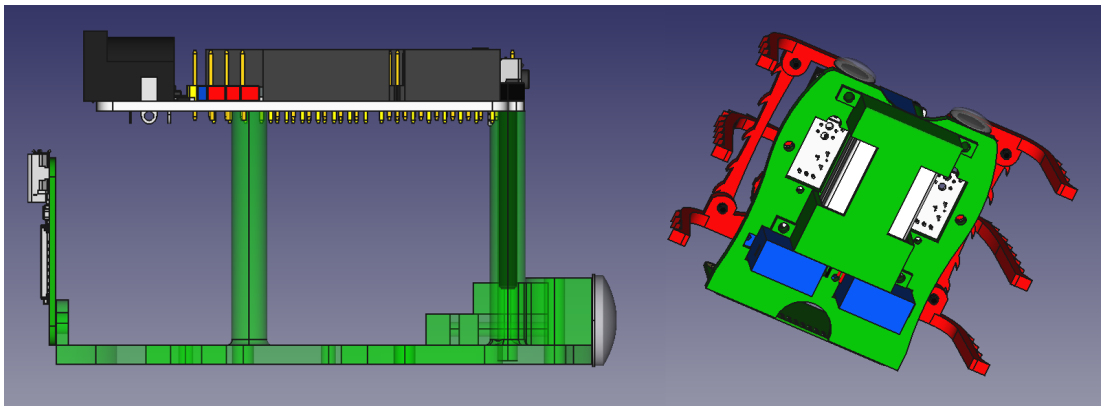


Fig. 5.13: Doodle side view and bottom. Most of the space between the FPGA and the base is for the movement of the legs.

To put the brain (*Raspberry pi*) and the nervous system (*Icezum Alhambra FPGA*) in *Doodle*, different methods have been used. The *Raspberry pi* is screwed it using two simple vertical holes in the back of the base. The *Icezum Alhambra* is attached on the top of the robot using the columns that emerge from the base. Finally, for holding the batteries and have a better weight distribution, the batteries are put below the base using a special part for them (Fig. 5.13).

Thanks to the low-cost 3D printing technology (section 3.2.2), a robot can be designed and, in a matter of hours, being printed with a ridiculous cost of just cents. This simplifies the iterations process paradigm boosting the design. In addition, *Open Source* projects have the advantage that other people are building and testing the robot, discovering different design issues faster and even fixing and improving the design.

Some of the issues that *Doodle* has faced and have been fixed, are the weakness of the initial holder columns, been easily breaking at the base, or the different tolerances presented in the most common cheap *SG90* servo motors horns, and the original ones, needing some additions and fixes in the leg designs.

Besides these problems, *Doodle* is a simple design that can be easily 3D printed in half of a day, and build it in just 30 minutes.

5.2.1.4 Creating the low-level circuits

The low-level system of *Doodle* is probably one of the most exciting parts of the robot. All the locomotion and sensorization of the robot are implemented in this part, so the robot is able to move without a microprocessor. Although numerous circuits have been tested using *Doodle*, in this section we are only going to talk about the main one, the line follower done using *CPGs* signals and basic sensorization.

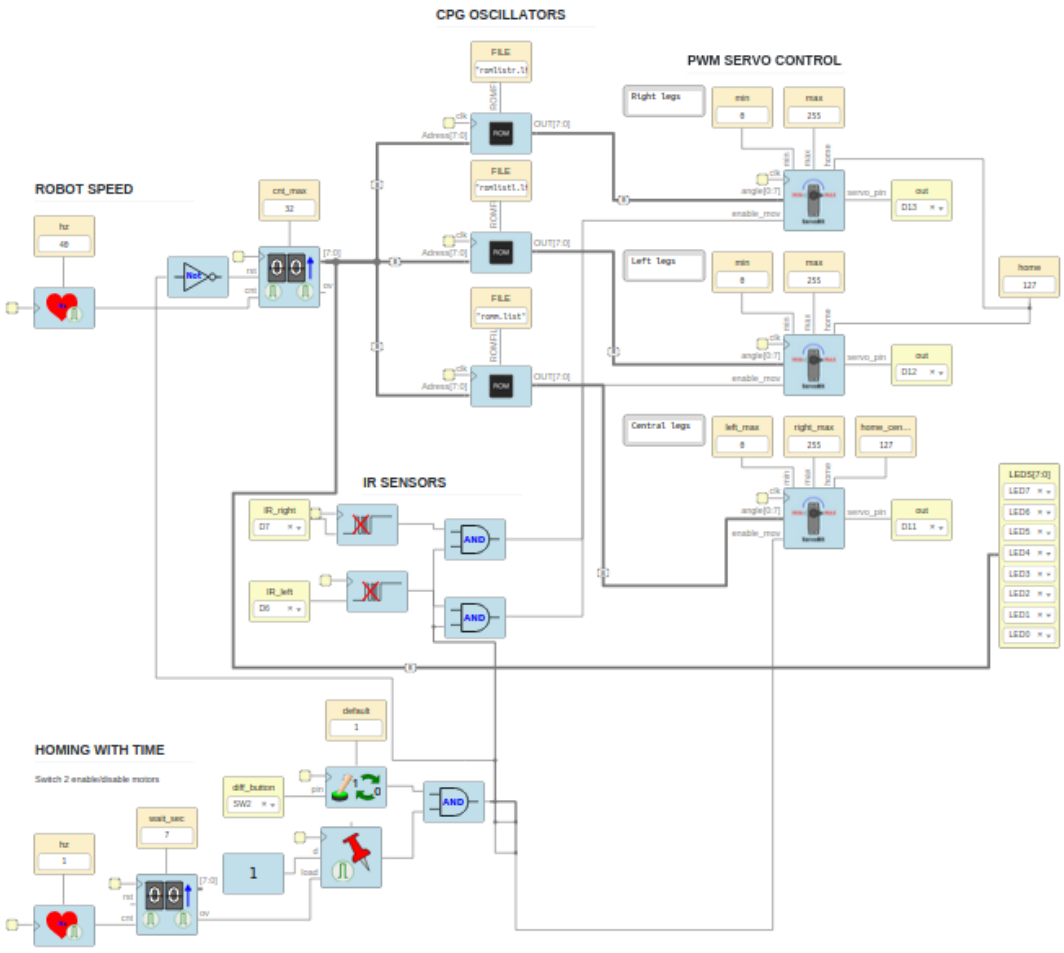


Fig. 5.14: Doodle low-level control. A low-level developed in Icestudio that uses CPGs signals and basic sensorization to follow a line.

The first circuits to play with *Doodle* and its *FPGA* were done in *Icestudio* (section 3.3.2.5). *Icestudio* is a perfect playground for creating new circuits, thanks to the combination between graphical blocks and custom *Verilog* code.

The circuit (Fig. 5.14), can be divided in different sections. The *robot speed* section is the part that controls the execution velocity of the robot. To do that, the section has a pulse signal generator (heart) that can modify its velocity in *Herzs*. This pulse generator is connected to a 8-bits simple counter that can change its maximum count thought the *cnt_max* parameter. The *cnt* input controls the increase of the counter and *rst* puts the counter to zero (Fig. 5.15).

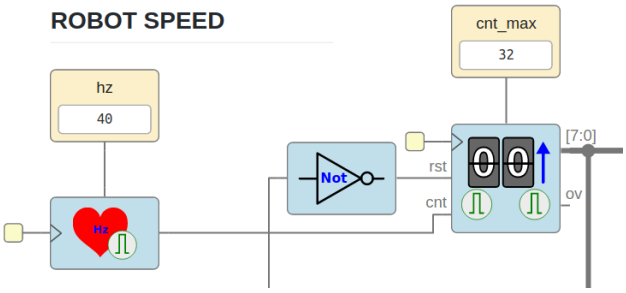


Fig. 5.15: Doodle robot speed circuit

The *homing with time* section controls the initial behaviour of the robot. In the beginning, the robot puts all the legs in the neutral or home position and the counter of the *robot speed* section at zero during a time defined by the *wait_sec* parameter.

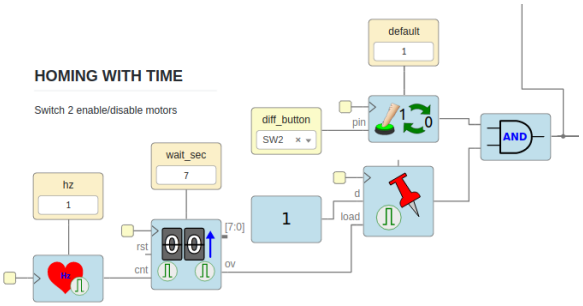


Fig. 5.16: Doodle robot homing with time circuit

This behaviour is controlled thanks to a pulse generator of *1 Hz* and a counter connected to a simple flip-flop with an initial value of 0 and a stable value of 1 after the first clock signal. In parallel, the push button *SW2* of the *FPGA* is connected to a block that switches between 1 and 0 at every push. With this circuit, we can put the robot to the home position and reset the speed counter pushing the *SW2* button.

The final enable signal is the result of the two circuits connected thought an *AND* logic gate (Fig. 5.16).

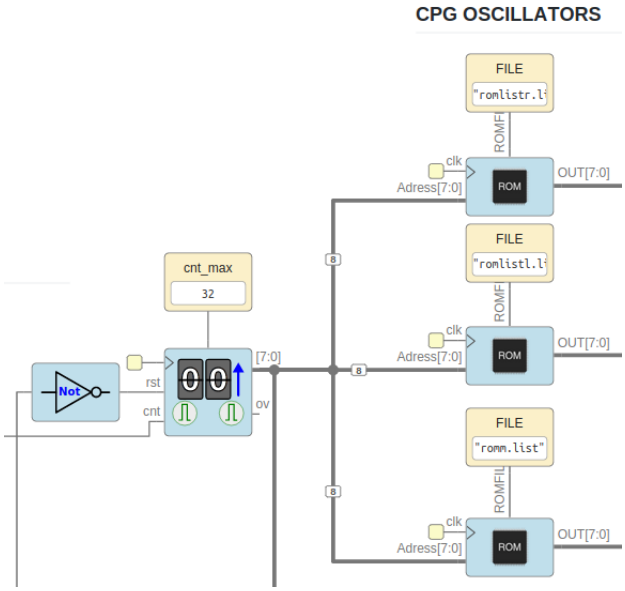


Fig. 5.17: Doodle robot CPG roms circuit

The core of the circuit is the *CPG oscillators* section (Fig. 5.17). There are different options to create *CPG* signals in hardware. The first one is to create circuits that calculate the different values of the signal. This has been done in other examples with counters to generate triangular signals, but for sinusoidal signals and more complicated signals, it involves math operations of multiplying and dividing that usually requires so much effort and space in an *FPGA*. In the *FPGAs* circuits usually these signal are generated using *look-up* tables that store the pre-calculated values of the signal.

For this case we have created three look-up tables that store the three *CPG* signals that control each servo motor of the robot. We have use *ROM* modules of a size of 32x8 bits that load the values of the *ROMs* using the auxiliar files *romlistr.list*, *romlistl.list* and *romm.list*.

These files are plain text where each line represents a slot of the ROM memory coding a byte in *hexadecimal* notation. The generation of these files by hand is tricky and requires so much effort for a human. As we will see in the *PWM Servo Control* section, the angle values of the servo motors move between 0 and 180 degrees. These

values have to be mapped to a byte range between 0 and 255 that have to be written in *hexadecimal* notation.

Doing this by hand to generate a triangular or sinusoidal signal is confusing and requires more than four operations per value. To facilitate these tasks and be able to play with *CPGs* in hardware in an easy way a *Python* script called *oscillator_rom_generator.py* has been created (88).

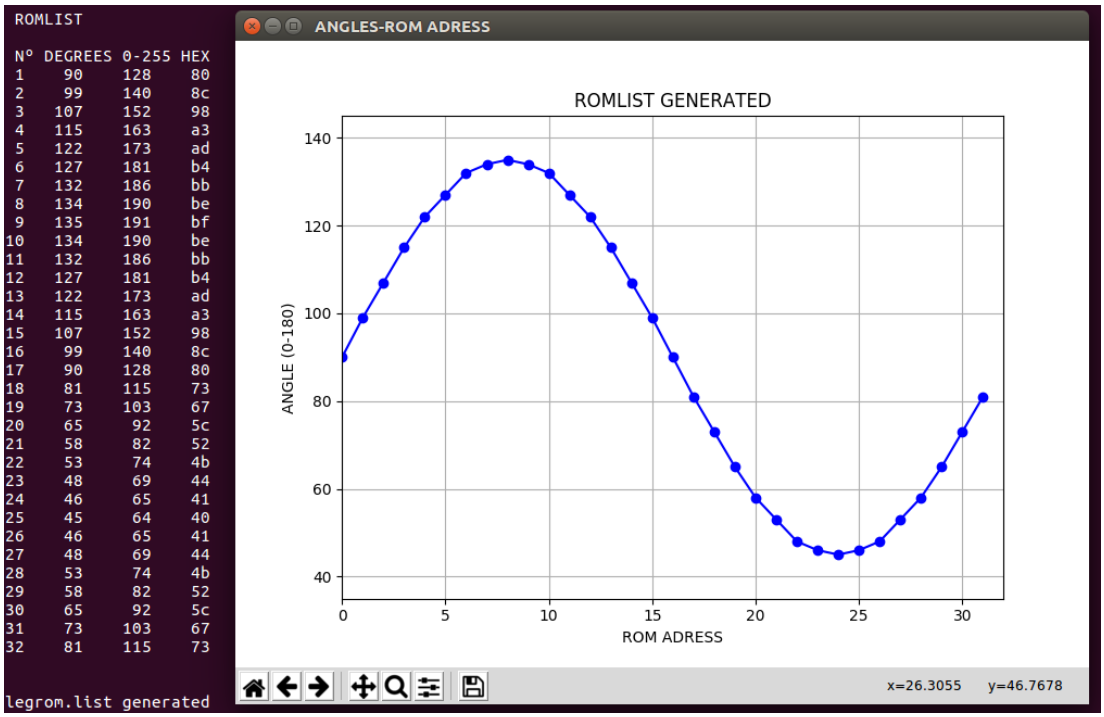


Fig. 5.18: Results of the ROM generator script

The script can be called thought command line or as a *Python module*. The only parameters that are necessary to indicate are the form of the signal (sinusoidal, squared or triangular), the maximum values, the size of the *ROM* and the name of the *rom list* that will be generated.

```
1 $ python3 oscillator_rom_generator.py -sin 0 180 32 legrom.list
```

Listing 5.1: CPG ROM generator script

The result is a *ROM* file with a signal like fig. 5.18. Thanks to this python program, it is very easy and quick to play with different *CPG* signals for the locomotion of the robot. In the case of *Doodle*, the two servo motors of the legs are normally controlled by

a triangular or sinusoidal signal and the central one by and squared signal that elevates each side of the robot alternatively. The *ROM* blocks are controlled by the *robot Speed* counter and the output of them are connected to the *PWM* servomotors blocks.

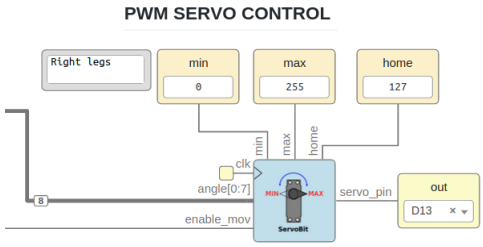


Fig. 5.19: Servo Motor PWM Control Block

The servo motor block is a standard *PWM* servo control that has been modified to fit the needings of the robots of this project. The block receives the angle to move from the respective *ROM* memory in a byte bus and converts the signal to the desired *PWM* for the servo.

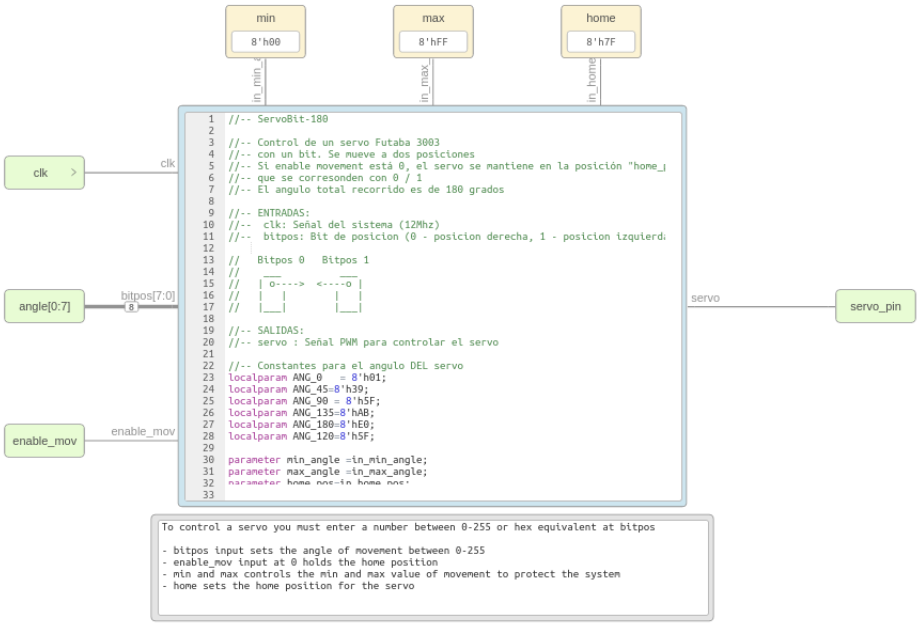


Fig. 5.20: Verilog code of the custom PWM motor block

As we can see in fig. 5.19 and fig. 5.20, more inputs have been added to properly control the motors of the robot. The *enable_mov* input enables or disables the motor,

putting the robot to the home position, controlled by the parameter *home*, or allowing the free movement. Another two additional parameters have been added to increase the movement safety of the robots with the *min* and *max* parameters that can establish a minimum and maximum angle for the motor. If the servo receives and angle out of this range, it will only reach one of the limits. This is very useful in the case of a robot like *Doodle* where angles smaller than 30º or bigger than 150º will make the legs collide with their own body having the risk of breaking the leg or the servo motor.

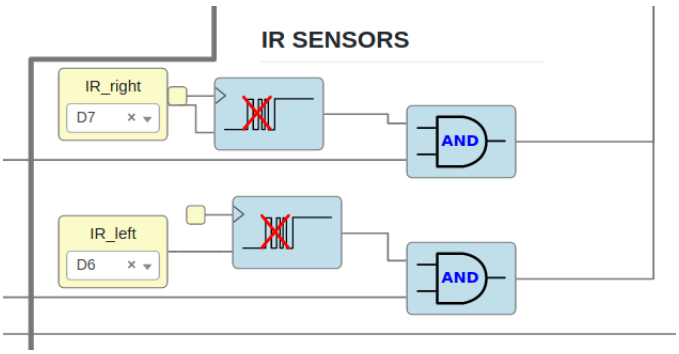


Fig. 5.21: IR sensors of Doodle to follow a black line or avoid a fall from a table

The low-level sensors of *Doodle* are two digital infrared modules that can be calibrated thought a potentiometer to detect a black line or avoid the fall from a table. They are connected to a debouncer circuit to improve the stability of the sensor and connected using an *AND* gate to the *SW2* switch button that will typically have the value of 1 (Fig. 5.21). The output of these *AND* gates are connected to the left and right PWM servo motor blocks enable pins. In the case that one of the sensors detect the black line, the legs of this side will stop moving, turning the robot until the sensor will not detect the line anymore. In the end, these low-level sensors work with *Doodle* as *Braitenberg vehicles* (89) where the output of the sensors are directly connected thought hardware to the movements of the robot, having a great speed response and working independently from the high-level.

Icestudio is very interesting to prototype and test different circuits and modules in an easy and faster way. However, for the whole architecture proposed in this work, one of the main features is the capability of the *high-level* of creating and modifying new circuits of the *low-level* system and upload them to the *FPGA* in real-time. For this reason, it is necessary to describe all the modules and circuits in traditional *Verilog* files. The conversion from the *Icestudio* circuits to the *Verilog* files is relatively easy to do thanks to the fact that every block described in *Icestudio* is at the end a *Verilog* module (Fig. 5.22).

```
module doodle_line_follower(
  input wire clk,
  input wire swh1,
  input wire swh2,
  input wire r_ir,
  input wire l_ir,

  output wire r_leg,
  output wire l_leg,
  output wire c_leg,
  output wire[7:0] led
);

//Internal wires
wire out_pump_speed;
wire [7:0] out_counter;
wire [7:0] out_rom_r;
wire [7:0] out_rom_l;
wire [7:0] out_rom_c;
wire r_out_debouncer;
wire l_out_debouncer;
wire out_flip_flop;
wire out_init_timer;

//Velocity of the robot
pump_bits #(.M(200_000))
  pump_speed(
    .clk(clk),
    .clk_out(out_pump_speed)
  );

//Counter that decides with address of the memory read
counter #()
  counter(
    .clk(out_pump_speed),
    .rst(swh1),
    .value(out_counter)
  );
```

Fig. 5.22: Part of Doodle line follower circuits described in Verilog code.

As we will see in the next section, this *Verilog* code will be the basic template for the *high-level* control for generating new circuits based on it.

5.2.1.5 Creating the high-level brain

The *high-level* brain of *Doodle* is implemented in a *Raspberry Pi Zero W*. This micro-computer is very small and fits on the back of the robot, been connected to the *low-level* control thought a micro-usb to micro-usb cable. Thanks to this connection, both levels can communicate with each other using the *UART* protocol.

However, sending *high-level* orders to the *low-level* system and receive data from it is not the only purpose that this communication has. Thanks to the *Open Source FPGA* tools, the *high-level* is able to create, verify, synthesize, and upload new circuits to the *low-level FPGA* in almost real-time.

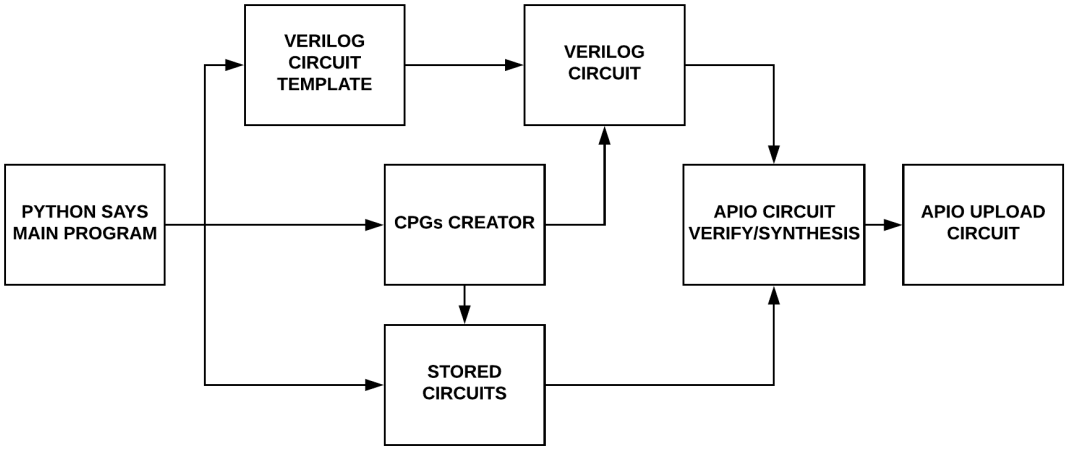


Fig. 5.23: Python says. A Python module executed in the high-level to create, verify, synthesize, and upload new circuits to the low-level nervous system.

This means that bio-inspired robots like *Doodle* have the capacity of reconfiguring their own nervous system without human intervention. This interesting feature has been developed in a *Python module* called *Python says* (90).

As we can see in fig. 5.23, *Python says* is a *Python module* that is called by the main execution program of *Doodle* to administrate the *low-level*. The *high-level* can decide to create a new circuit from a template or adding existing modules, or take an already built circuit from its storage, or most interesting, directly from a repository on the cloud. A typical case could be the bad behaviour of a leg. The *high-level* system can decide to change the whole locomotion of the nervous system in order to move better with a damaged leg, or just remove the leg from the nervous system to avoid more

damage and false inputs. Once the system has decided to create a new circuit from itself or take an existing one, *Python says* will verify the *Verilog* code, synthesize the circuits for the *Icezum Alhambra FPGA* or another equivalent board and, if everything is right, upload the new circuit to the nervous system *FPGA*.

```
1 //Velocity of the robot
2 pump_bits #(.M(@M))
3   pump_speed(
4     .clk(clk),
5     .clk_out(out_pump_speed)
6   );
```

Listing 5.2: Empy verilog template example

To create new circuits, *Python says* uses a system of *Verilog* circuits templates that can be customize using the *Python Library Empy* (91). *Empy* is a powerful library to work with templates in *Python* using expansion symbols (@), as we can see in the example code of this page, *Python says* can control the parameter *M* that controls the speed of the robot, putting in any case the high-level system the value that it wants for each moment. The most interesting thing about *Empy* is that it is possible to embed *Python* code directly in the template file. This is very useful as we will see forward in section 5.3, to embed *for* statements and lists that control the number of *ROMs* and *PWM servo motors* modules that the *high-level system* wants to create, being able to add, remove or customize essential parts of the nervous system and changing a big part of the robotic behaviour completely.

The rest of the *high-level* system works as a traditional robotics control that uses a computer. We can execute different code in different languages for computer vision and *AI*, running modules of task decision, slam and much more, and even integrate the whole architecture with the *Robotic Operating System ROS* (section 5.2.5).

With the *high-level* system working as the brain and making high-level orders thought software, and the *low-level* system controlling in parallel the locomotion, basic behaviour and sensoring of the robot thought hardware, *Doodle* is a simple robot with the full bio-inspired architecture working properly. Once the viability of the architecture has been demonstrated, it is time to test the architecture in other experiments and finally integrate it into a more complex robot.

5.2.2 Randofo - Implementing the basic architecture in other robots

During the development of this work, one of the main goals has been experimenting with the architecture and the *Open Source FPGAs* in as many robots as possible. One of these robots is *Randofo* (92), a bizarre and funny robot with a curious locomotion.

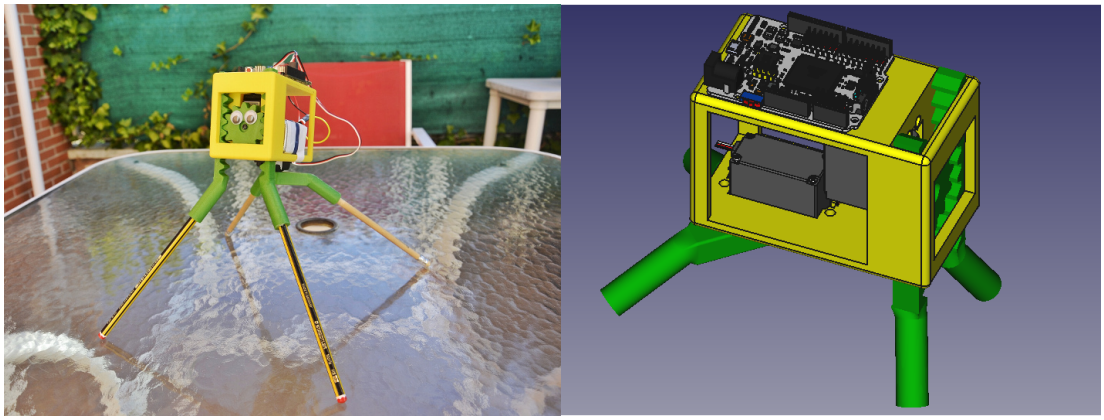


Fig. 5.24: Randofo. A very creative simple robot that uses pencils as legs. The central part was adapted to fit the new motors and the Arduino UNO/Icezum Alhambra boards.

After experimenting with the first locomotion iterations of *Doodle*, it seemed interesting to test in other robots how easy was implementing the locomotion circuits and how many changes were needed to do it.

In this sense, *Randofo* is an interesting robot to test it due to its simplicity and peculiar leg locomotion. *Randofo* is a robot created by *Randy Sarafan* (93) that uses four pencils as legs (Fig. 5.24). Its mechanism is simple but witty, the front mechanism uses a simple gear to move the front legs up and down alternatively, and the back legs are moved by a horizontal servo motor to push the robot with each leg alternatively. The locomotion is simple but tricky to control and resemble the crawling movement of a baby.

Unfortunately, *Randofo* was made for a different electronic setup, and it was necessary to make some changes in the CAD design to adapt it for the use of *Futaba* servomotors and *Arduino UNO/Icezum Alhambra* shape boards (Fig. 5.24).

Once the design of the robot was adapted, the *Arduino* code was tested. The robot movement was just right, and requires a lot of calibration and playing with some offsets parameter to have a satisfactory forward movement without so much slip.

On the other hand, putting the *FPGA* and adapting the locomotion circuits of *Doodle* was extremely easy (Fig. 5.25). Thanks to the wave generator program, it was easy to test and create the right *CPGs* movement signals for *Randofo*.

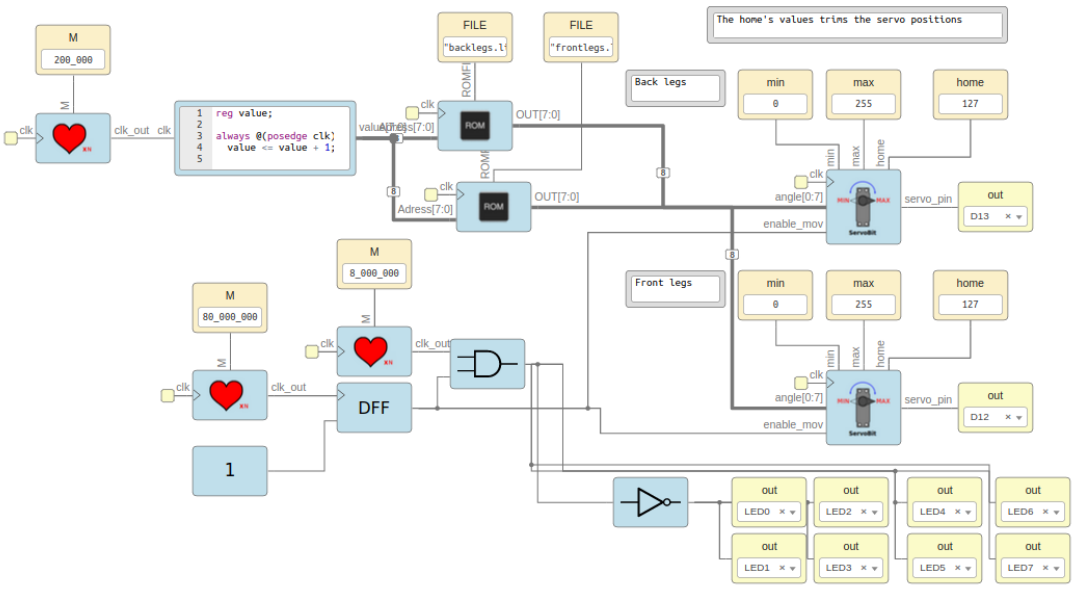


Fig. 5.25: Randofo circuit. A quick adaptation from the basic Doodle control circuit based in the use of ROMs that contains CPGs signals to control the servo motors.

The result was surprisingly good (94). Thanks to the *CPGs*, the robot movements were smoother, and the gait distance longer. In general, all the locomotion is far superior compared with the *Arduino* version, being the basic low-level locomotions circuits of *Doodle* an excellent reference to create hardware locomotion in other robots.

5.2.3 Reflexes movements and experiments

One of the most exciting aspects of the bio-inspired architecture that has been explored in this work, it is the possibility of creating reflexes movements. When we talk about reflexes movements in robots, one of the traditional approaches in the industry are the *deadman* systems to detect if something went wrong, like an electroshock or a human crossing the safety barrier of an industrial arm robot. These systems usually work directly cutting the power of the robot, a dangerous approach in robots with too much weight, or blocking all the motors though software interruptions.

With the proposed architecture we can go in a different direction, including reflex movements, not only for safety, that are directly implemented in the hardware of the *low-level system*. This approach has some exciting advantages; the first one is the execution speed. In case of danger, the reflex action will be executed as fast as possible, no matter in which task or moment of the execution we are, a circuit is a circuit, if a danger signal is activated, the circuit will immediately respond with the correspondent action. More importantly, this response, as it happens in humans and other animals, will be automatically and will not depend on the *high-level* system software. It does not matter what the *high-level* system is ordering; if the *low-level* control reflex system is detecting a danger, it is going to do whatever needs.

For testing these kinds of implementations, the sensors of *Doodle* has been implemented in the *low-level* system, acting directly to the locomotion motors. Other experiments, like the building and programming of a robotics eye using an *FPGA*, has been developed (95).

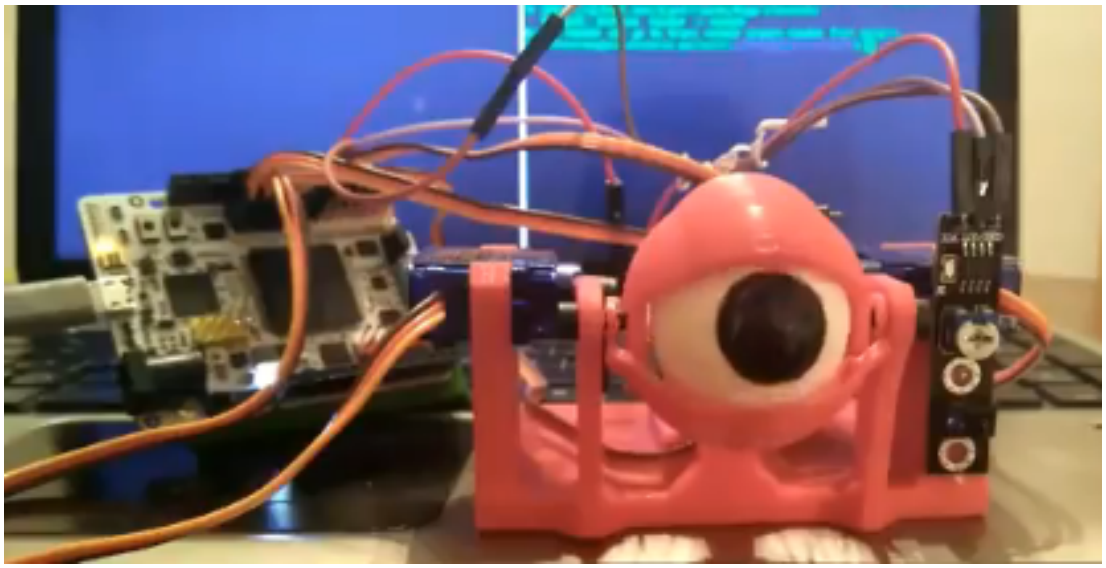


Fig. 5.26: Robotics eye controlled with an Icezum Alhambra. The system can move the eye in all the directions and can also open and close the eyelids.

As we can see in fig. 5.26, it is a robotics eye that uses four servo motors and a wire system to move the eye in all the directions and close and open the eyelids of the eye. In the side of the mechatronic eye, it has been added a digital infrared sensor to detect when someone is very near or is touching the eye. The eye moves in a pseudo-random

way in all the directions. When a near object is detected, the system closes the eyelids immediately, protecting the inner eye.

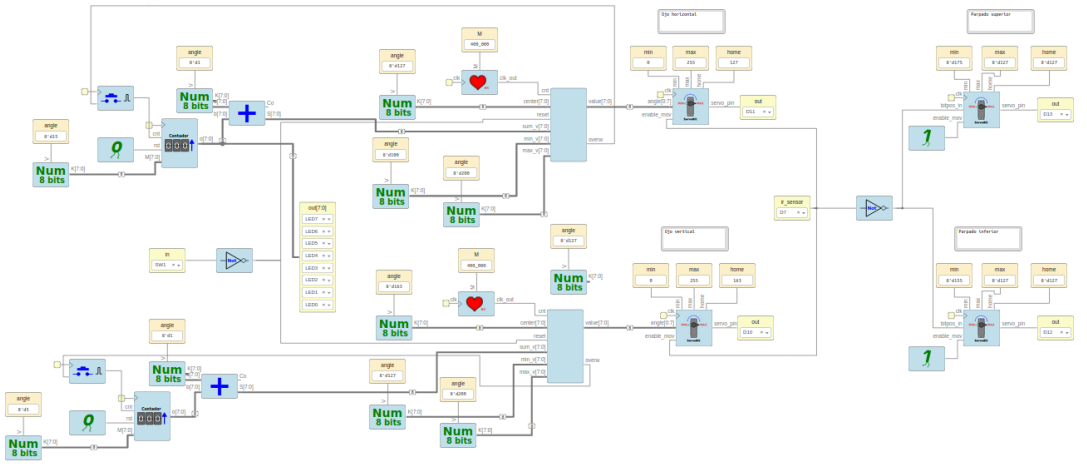


Fig. 5.27: The Icestudio circuit for moving the mechatronic eye. Two counters with differents phases move the X and Y motors of the eye, while the IR sensor controls the reflex movement of the eyelid.

With the proposed architecture (Fig. 5.27), it would be possible to create robots with a whole *nociception* system, having like in real animals, different sensors to protect the robot from different dangers, like skin sensors for punctures or cuts, sensor for burns or for one of the worst enemies of a robot: water. All of these sensors can work in parallel in the *low-level* system, no matter how many of them we have, the computational resources of the *high-level* system of the robot will not be affected.

If we add this kind of hardware implementations in future robots, we will be able to create robots with protective instincts that will help the protection of the robot and at the same time will add a new layer of natural reacting, like close the eyes when an intense light is turning on, or going back when someone collides with it unexpectedly. These actions will help to improve the social reactions towards the robots in social situations where humans and robots have to live or work together.

5.2.4 ZowiHumanoid - Experiments with more advanced locomotions

From the beginning, *Zowi* (section 2.2.2) was created using oscillators as the main method of control. Thanks to these oscillators Zowi's locomotion is greatly simplified, being able to control a high level of *DOF* in an extremely effective way. This control is a fascinating advantage to program humanoid robots, as *Javier Isabel* explains and shows on his robots (45). Although its original version and the commercial one, only have a body and two legs with a total of four *DOF*, the *open-source community* and robotics students associations like *Asrob* (96) have created new versions with head and arms with a total of nine *DOF*, even more. These versions are called *Zowihumanoids* and are used as recreational combat robots (Fig. 5.28).

Fig. 5.28: Different Zowihumanoids created (left), the Zowi Maker Robot created for this project (right).

With the purpose of implementing the bio-inspired architecture on it, the *Zowi Maker Faire* Zowihumanoid has been created (97).

With the *Zowi Maker Bot* the Zowihumanoid design has been adapted to use the *Icezum Alhambra FPGA*, being able to control all the arms, legs and head though oscillators created in hardware. However, due to Zowi's locomotions are complex and require more time and effort, only a simple forward movement has been implemented, being a future goal the implementation of more complex locomotions and the addition of the high-level control to the robot.

5.2.5 APIO ROS - Integrating Open Source FPGAs with ROS

The *Robotic Operating System* (*ROS*) is a popular framework designed with the goal of being an standard for programming robots, creating an abstraction between the hardware and the programming of the robot with the objective of facilitate the reuse of the code in other platforms or changing the hardware of a robot just making minimal changes on the software.

ROS born in 2007 in the University of Standford, since then, lot of versions linked with different versions of *Ubuntu* has been released. Nowadays it is a common tool in robotics research, and it is becoming, despite its problems, a serious candidate of being one of the standards in the future robotics industry.

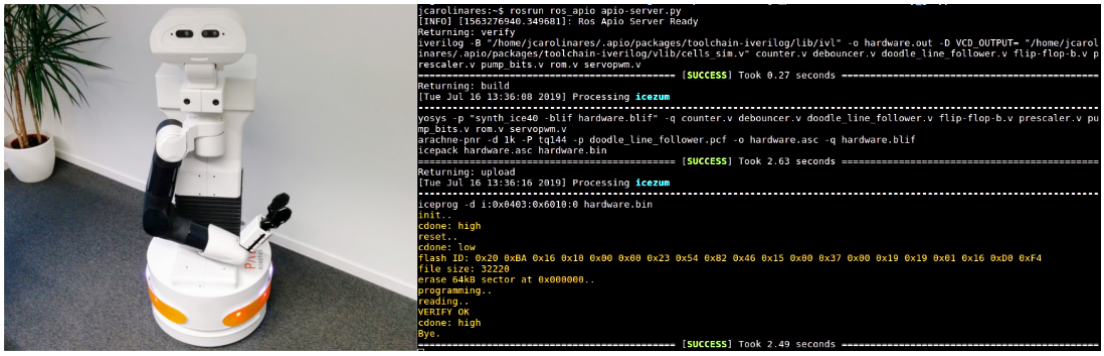


Fig. 5.29: Apio ROS service module (right) implemented in a Tiago robot (left).

For this reason, a *ROS* module service has been created (98). The module receives *ROS calls* for a *ROS* client node and using *APIO*, verify, sinthetize and uploads new circuits to the *Open Source FPGA*.

The module has been tested in a *Tiago* robot with an *Icezum Alhambra* connected to it. The results are great and open new possibilities of implementing in the future the proposed bio-inspired architecture adding an *FPGA* in robots controlled with *ROS* like, for example, *bio-inspired* robots for social uses like pets animal for autistic children and others where the natural movements that offer this kind of architecture can be handy.

5.3 Crabdle - A bio-inspired robot with a Brain and a Nervous system

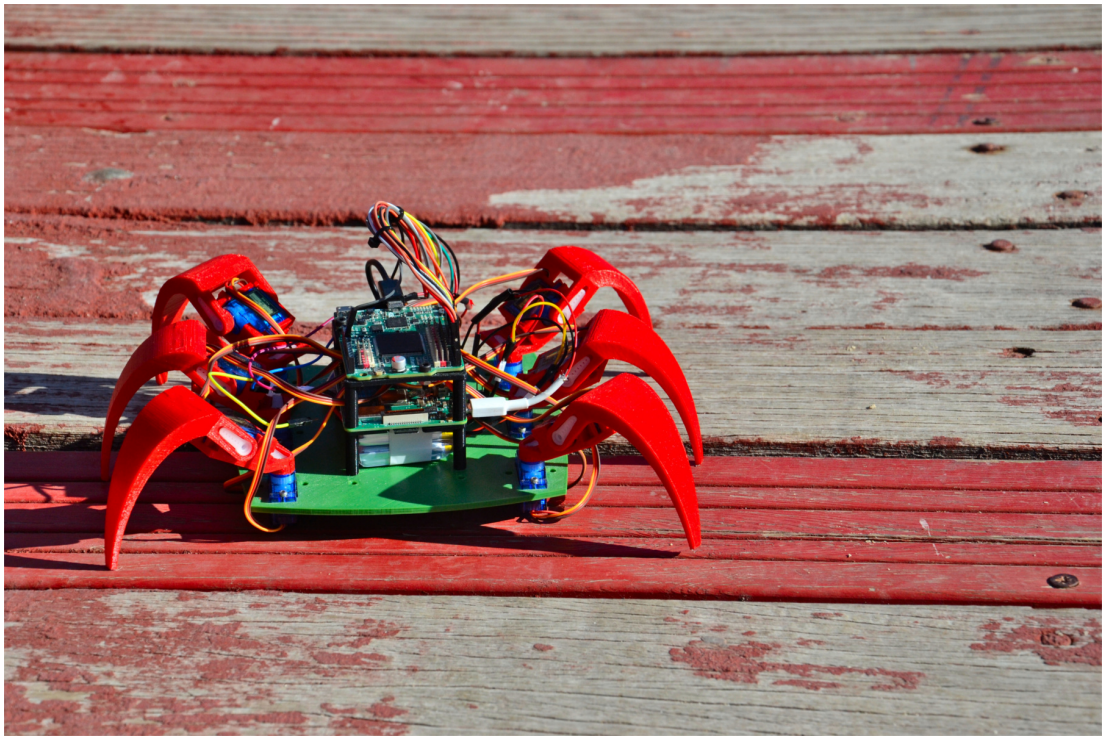


Fig. 5.30: Crabdle. A hexapod with 12 DOF created with a brain and a nervous system to implement the bio-inspired architecture of this work.

Crabdle (99) is a bio-inspired *Open Source* robot with a *Raspberry Pi* as a brain or *CNS* and an *Open Source FPGA* as the nervous system or *PNS*. The main purpose of creating *Crabdle* was to apply all the experience learned in the creation and implementation of the control architecture in *Doodle* and other robots and experiments and apply the proposed architecture control as the core of the robot.

While *Doodle* was a relatively simple robot created to play with the bio-inspired architecture avoiding the typical problems related to more sophisticated robots, *Crabdle* is a more complicated robot in all its levels. The number of degrees of freedom is twelve instead of three. The electronics are the same base as *Doodle* but with more components and trying to create an affordable robot without compromising its functionality. The electronics, especially power management, are more complicated. The design is more

ambitious and plays with the limitations of the digital fabrication tools available for this work, without losing the possibility of creating a big robot easy to create, modify, and 3D printed. Although the whole design of the robot is more complicated, it has been sought to be an easy-to-assemble robot for everyone, avoiding the use of a lot of screws, required special tools, and complicated building steps.

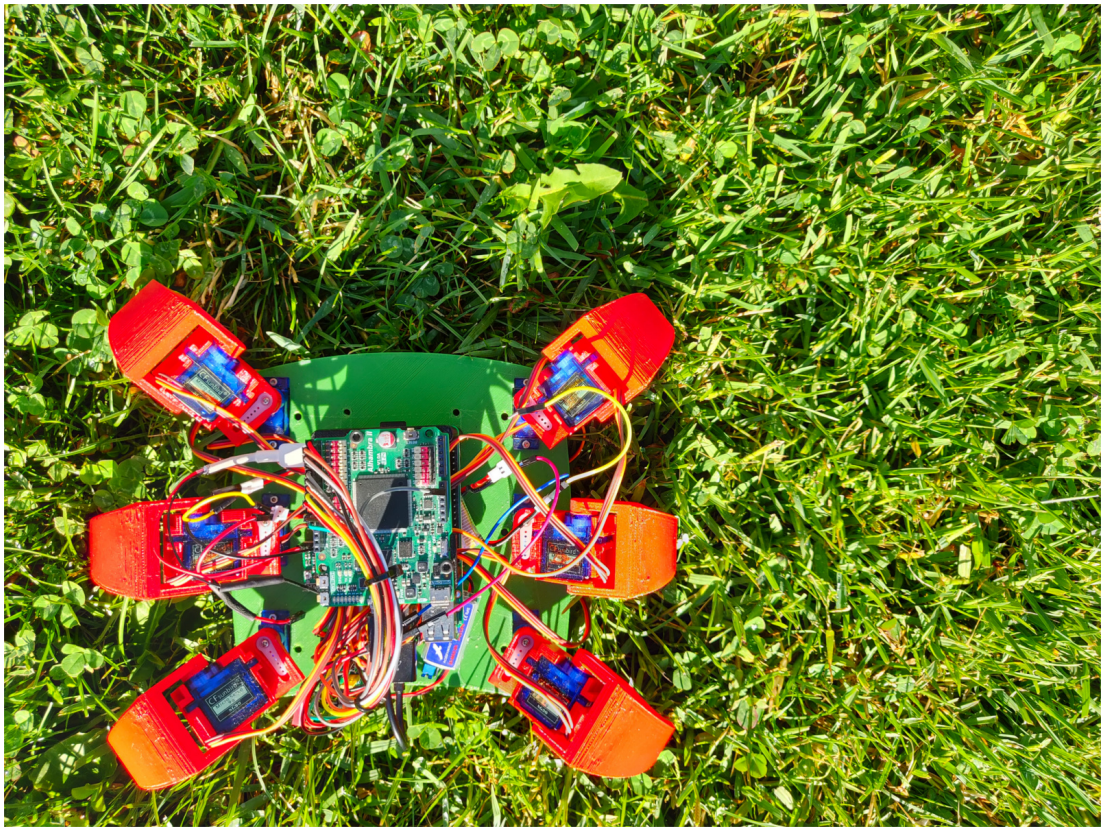


Fig. 5.31: Crabdle can be seen as another hexapod robot, but its bio-inspired control architecture using Open Source FPGAs makes him something special.

As we will see the robot is modular by design. The legs can be changed to two to three degrees of freedom in the future and can be assembled on different bases adding or removing legs without changes in their design. *Crabdle* is the final prototype of this work, and resumes and integrates on it all the technologies and experience learned during the development of it. Furthermore, it is the base for future robots that will expand the bio-inspired architecture.

5.3.1 Design of the bio-inspired architecture

Crabdle is a hexapod robot with a size of approximately 30 by 30 centimeters. It has six legs with two degrees of freedom, with a total of twelve *DOF* directly controlled by twelve servo motors. Although the hexapods usually have another *DOF* per leg to even add more freedom of movement, the locomotion possibilities of *Crabdle* are far superior from *Doodle*, being able to move forward with different locomotions, move backward, make turns, lateral movements and much more.

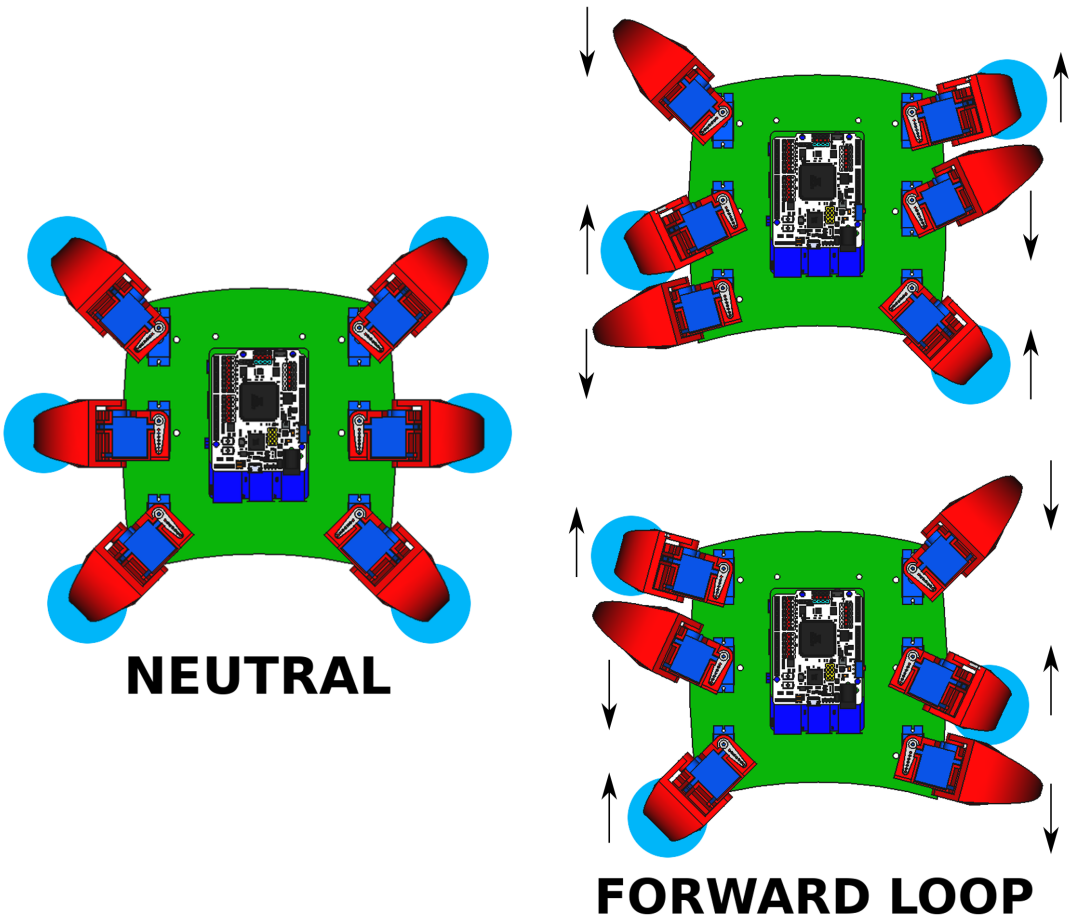


Fig. 5.32: Crabdle forward locomotion. The blue dots represent the legs that are touching the floor. The arrows indicate the next movement direction of the leg.

As we can see in fig. 5.32, the *Crabdle* locomotion of a forward movement is more complex than in other robots used during this work. It works like most of the hexapods,

using a system of triangles where the two legs from the extremes of a side are coordinated with the central leg of the opposite side. Putting down the legs for doing the movement of advance and raising the legs to make the motion of putting the legs ready for the next movement of advance. The mechanism of *Crabdle* is easy to understand. Each leg is composed of two *DOF* controlled by two servo motors.

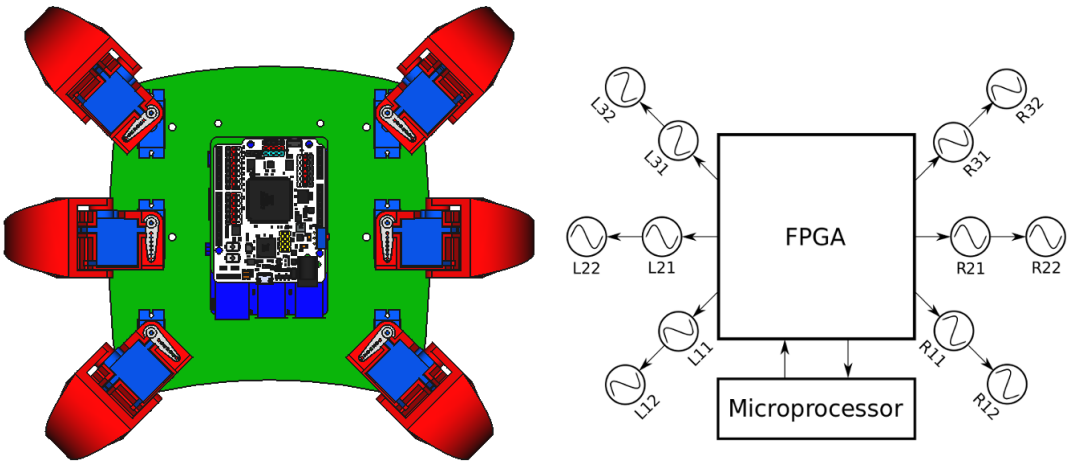


Fig. 5.33: Crabdle bio-inspired architecture. The high-level microprocessor does high-level tasks and give orders and receives data from the low-level system, which controls the complex locomotion of the robot using CPGs.

If we take a look at fig. 5.33, we can see that *Crabdle* has twelve *CPG* signals that corresponds to each motor of the robot. Each leg is named using a particular nomenclature where the first letter indicates the side, the first number the leg, and the second number the part of the leg, being the number one the motor that moves the leg in horizontal and the number two the motor who raise and lower the leg.

In robots like *Crabdle*, where there are a lot of degrees of freedom and many locomotion possibilities, traditional controls using only microprocessors for the movements cannot be enough. With a high number of motors to control, the microprocessor has to do a loop to give the right orders to each motor one by one, in a sequential order, loosing time and having a small delay between each order that can be increased more and more if the microprocessor is busy trying to do some important tasks at the same time. Although in a robot with twelve *DOF* the microprocessor, with some small exceptions, can do the job, in the case of *Crabdle* this problem is just trivial due to the proposed control architecture. Thanks to the control of the locomotion by the *low-level*

system using hardware, all the control circuits of each motor of each leg are moving in sync and at the same time without any loose of performance and without drawing resources from the high-level *microprocesor*.

If in the case of a simple robot like *Doodle* or *Randofo* the use of *CPGs* were a big step forward, in a robot like *Crabdle* the advantage is even more obvious. The movements are soft and take full advantage of the contact with the floor, resulting in a more natural movement and better locomotion.

Even more, the change of paradigm in the way a locomotion is designed, playing with the parameters of the waves, like frequency, amplitude, offset and phase, instead of thinking in a series of keyframes for each motor, it can be a little bit hard at the beginning, but after some practice, it is a way of creating locomotions more faster, and that requires less effort than other paradigms.

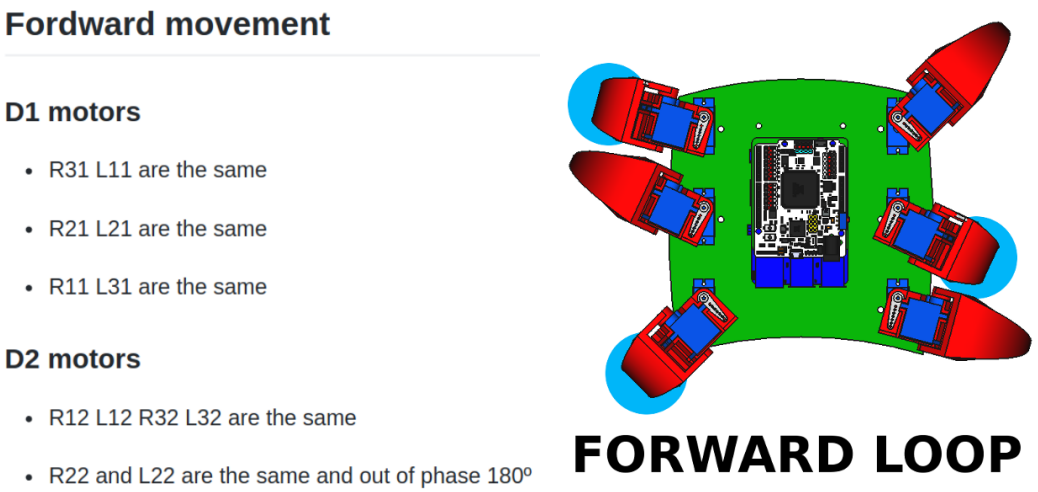


Fig. 5.34: Crabdle locomotion forward example. Thanks to the CPGs control paradigm, most of the motors share the same signal for this locomotion, controlling the 12 motors of the robot using only 5 different signals.

Thanks to its size, *Crabdle* has more space to incorporate differents low-level and high-level sensors and play with more advanced high-level tasks, being the perfect platform to play with the bio-inspired architecture of this work.

5.3.2 Design of the electronics

Crabdle has a more complicated circuit than *Doodle* for two reasons: the update from the *Icezum Alhambra I* to the *Icezum Alhambra II* with the differences in power that these boards have, and the fact that, even with all the improvements of the version II that allows more *amps* per pin, the board is not capable of powering twelve motors.

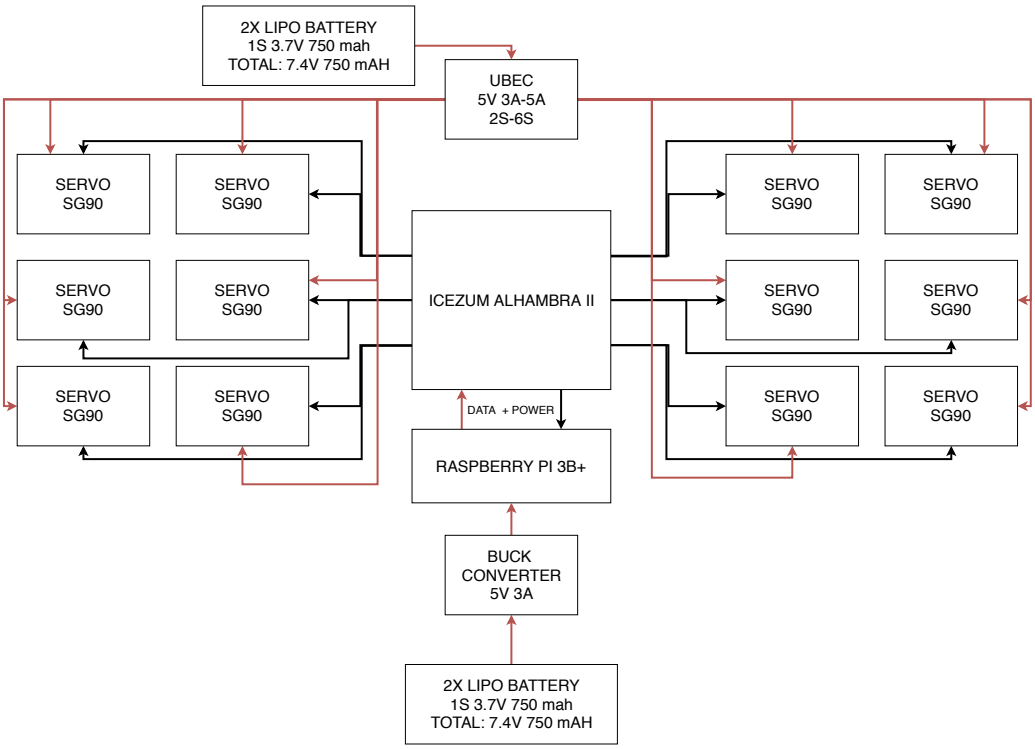


Fig. 5.35: Crabdle electronic diagram. The power of the boards and the motors are split into two different circuits. Red arrows represent power connections. Black arrows data.

If we take a look at fig. 5.35, we can see that there are two main power circuits. Both of them are powered with two groups of two *Lipo* batteries in series with a total of 7.4V and an autonomy of 750mAh. Due to the restrictions in the input voltage that the *Raspberry Pi 3B+* and the *Icezum Alhambra II* have, the first circuit transforms the 7.4V of the *Lipo* batteries to 5V using a *DC Buck converter*. This converter power the *Raspberry pi* that, at the same time thought an *USB to micro-usb* cable, power the *Icezum Alhambra II*, using the same cable for powering the *FPGA* and communicating the high and low level.

The second circuit powers the twelve servo motors of the robot. The 7.4V of the *Lipo* batteries are converted to 5V using an *UBEC* voltage regulator. This *UBEC* is typically used in model aircraft with an excellent performance, and it is able to give 3A with peaks up to 5A.

5.3.2.1 The eternal problem: too many wires

Creating a big robot with more degrees of freedom means more motors, that implies more cost, the necessity of more complicated power circuits due to the increase in power consumption, and more wires. In the case of the *SG90* servo motors, the cable is a three-pin jumper connector, very useful for connecting the motors directly to a board. Unfortunately in this case, only the signal pin is connected to the board, being the power and ground pins connected to the external power circuit that we mentioned above. The result using discrete prototyping elements like *breadboards* and jumper wires is a mess that cannot be properly integrated on the robot.

The best approach to solve this problem is creating a custom *PCB* board with the purpose of reducing the number of wires and simplify all the connections between the servo motors, the *UBEC* voltage regulator and the *FPGA* (100) (Fig. 5.36).

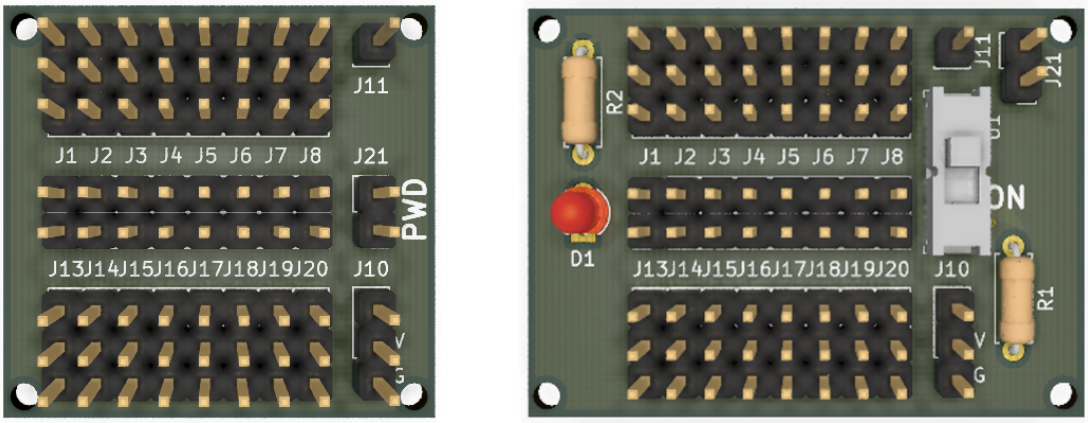


Fig. 5.36: The two versions of the External Power Adaptor boards created to simplify the connections. The servomotors are connected on the sides, the UBEC in the bottom right connector. The central pins are the signal pins of the motors that are connected with the FPGA using jumper cables.

The board is a simple two layers *PCB* with two rows of three-pins connectors to connect up to 16 motors to the chosen *UBEC* connector or another power source. In the center, two rows of pins allow connecting the signal pins of the motor to the *FPGA* (Fig. 5.36). An additional more complex board with power control thought a switch and a *TH LED* has also been created. With this board, the wire problem is solved, reducing as possible all the wires and connectors and having a good solution compatible with future robots (Fig. 5.37).

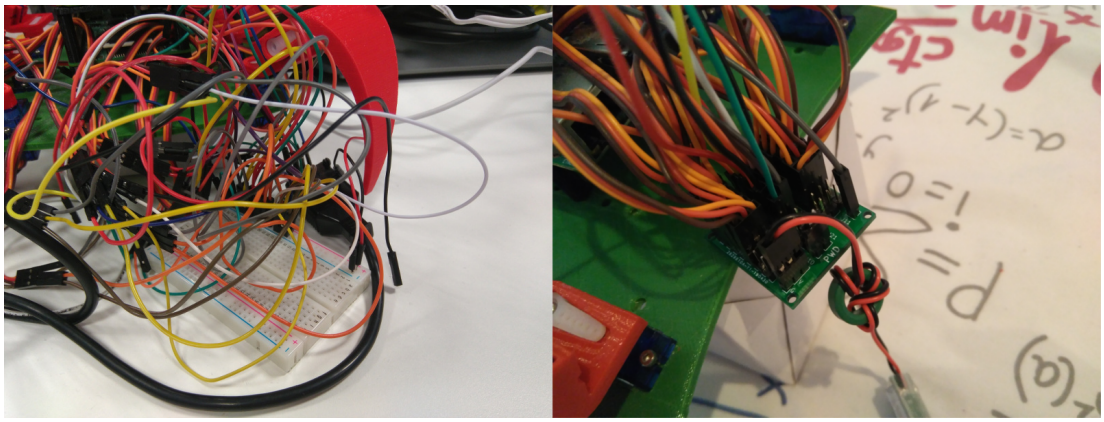


Fig. 5.37: Cables using a breadboard and jumpers cables (left) and cables of the robot using the designed board (right).

Apart from the power setup, *Crabdle* uses common and affordable electronics components composed of the following elements:

- **Raspberry Pi 3B+:** The *Raspberry Pi 3B+* is a great microcomputer that typically runs *Linux*. It has an integrated Wifi and Bluetooth connections, a handy *GPIO* pins and four *USB* ports perfect for connecting and powering different peripherals, like the *Icezum Alhambra FPGA*. The board can be powered thought a *micro-usb* or connecting the 5V and ground pins of the *GPIO* powers pins.

  The board is more powerful than the *Raspberry Pi Zero W* used in *Doodle*, but it also consumes more power. At the time of writing this document, the new *Raspberry Pi 4* has been released, with an impressive RAM memory of *4 gigabytes*. Some test has been done using *Apio* and this new *Raspberry Pi*, resulting in an impressive performance that is able to create, synthesize, and upload circuits like an standard laptop: in less than 20 seconds. Unfortunately, the board is even

more power drawing, requiring more future tests to know if it is possible to use this board in the robot.

- **Icezum Alhambra II:** This second version of the board has some interesting differences respecting the first one. The main improvement is the change of the size of the *ICE40 FPGA* from 1K to 8K, having more space for more complicated circuits, and even soft-cores like the *RISC-V* (85). The second difference is an improvement in the power that the board can offer per pin, improving the performance and allowing to power more peripherals using the board's pin. Unfortunately, these changes also mean the loss of the barrel jack, and more importantly, the loss of the original power regulator that allowed to power the board directly using 7.4V. Now we have to use an external *buck* converter to power the board to 5V.

- **12x SG90 servo motors:** As we said in the electronic of *Doodle*, the *SG90* are widely used in the *Maker* community. They have a torque of 2.5kg/cm that, with a clever design, is enough to move a robot like *Crabdle*. They are small and easy to control thought *PWM* signals and more importantly in a robot like *Crabdle* that uses a high number of them, they are very cheap, around three dollars per motor nowadays.

- **Short USB to micro-USB cable:** To connect the *Raspberry Pi 3B+* and the *Icezum Alhambra II*. Due to the *Icezum Alhambra II* does not power the servo motors and only drives the signals of them, it is possible to power the board using the same cable of data that we use to communicate the high-level system (*Raspberry Pi*) with the low-level system (*Icezum Alhambra).*

- **2x (2x 1S Lipo batteries):** As we explained before, we use two groups of two 1S *Lipo* batteries connected in series to power the boards on one side, and the servo motors on the other side. Each group of batteries has a voltage of 7.4V that have to be converted to 5V and an autonomy of 750mAh. That gives an autonomy of around 45 minutes for the servo motors and 1 hour and a half (best case scenario) for the boards. Although the autonomy of the robot can be improved putting better batteries, is out of the scope of the thesis of this work,

being this autonomy more than enough for the tests and experiments related with the bio-inspired architecture.

- **DC-DC Buck converter:** A cheap and simple *buck* converter that takes the 7.4V of the batteries and convert it to a constant 5V voltage to power the *Raspberry Pi + Icezum Alhambra*. The converter is able to give up to 3 Amps, enough in most of the cases to power these boards.

- **UBEC voltage regulator:** This voltage regulator has an impressive power performance. It can convert up to 6S *Lipo* cells, (around 22.2V) to a constant 5V voltage and can give 3A with peaks up to 5A. It is an excellent option for powering a high number of servo motors without experimenting power problems.

Fig. 5.38: Crabdle UBEC allocated in the body of the robot and connected to the external power adaptor of the servo motors.

In this case, the cost of the *Crabdle* robot is higher than the *Doodle* one. The *Raspberry Pi 3B+* is more expensive than the *zero* version, having a total of 80€ between both boards. If we add the value of the servo motors, having a total of twelve servo motors plus replacements with a cost around 2€ per motor, we have a cost of approximately 30€, adding the cost of the *Lipo* batteries, electronic converters, custom boards, plastic and screws we have a total cost around 170€. Not a bad price for a robot with twelve motors and two control boards. The detailed budget of *Crabdle* can be found at the Appendix A.

5.3.3 Design of the body

Crabdle is a robot created with a modular design in mind. Modular designs allow us to create robots with parts that can be easily modified and used in robots with different configurations and shapes. Saving time and avoiding problems.

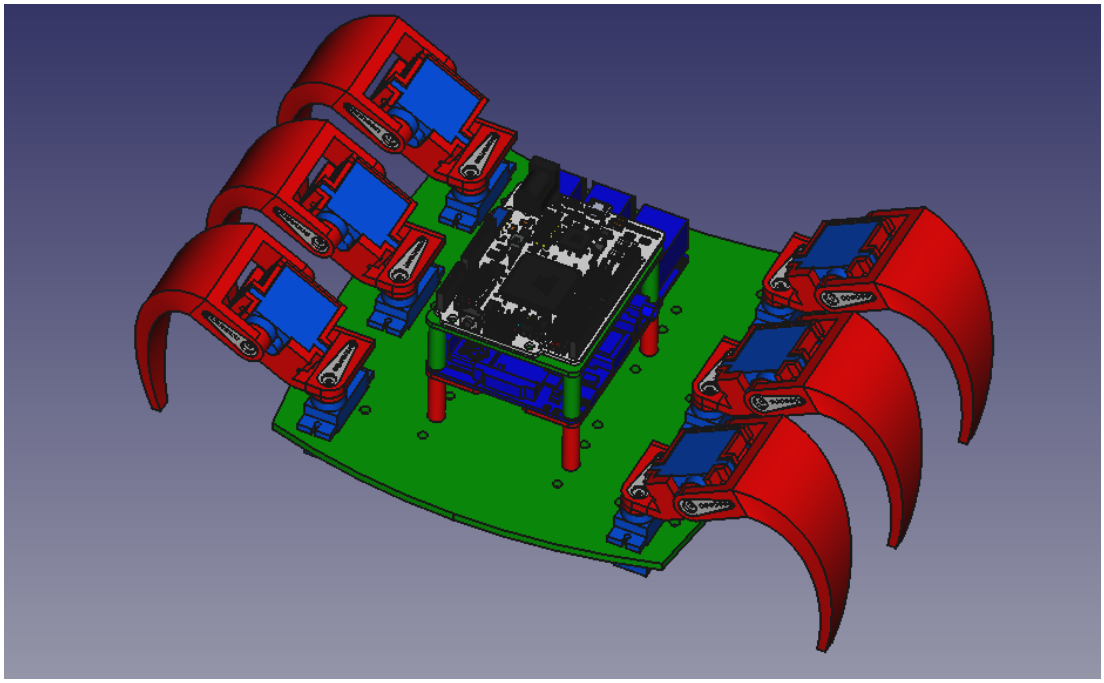


Fig. 5.39: Crabdle Freecad design.

One of the main problems found designing *Doodle* was the size of the robot. *Doodle* is small and its mechanism takes all the available space across the base, resulting in a robot where is difficult to put the batteries, new sensors or even do a proper mass balance to have a safe leg locomotion. In the the case of *Crabdle*, all the design lessons learned with *Doodle* has been taken into account. The robot has been designed having in mind the available digital manufacturing tools, having parts with simple shapes that are easy to 3D print with a minimal use of supports. The components have been placed in a configuration where all the heavier parts, like the two boards and batteries, are placed on the same vertical on the center of the robot, providing an stable locomotion. All the design has been created with a modular idea in mind: the future ampliation of the robot to connect bigger bases with eight legs per base to create a centipede robot.

For designing *Crabdle*, the best approach was to firstly design the base, and after that, design the legs parts taken into account the height and shape of the base. The minimum base iteration were the two parts to stack the *Raspberry Pi* and the *Icezum Alhambra* together. The design allows free access to the *GPIO* pins of the *Raspberry Pi* and enough space to put the batteries below the two boards, keeping the center of masses near the center of the robot (Fig. 5.40).

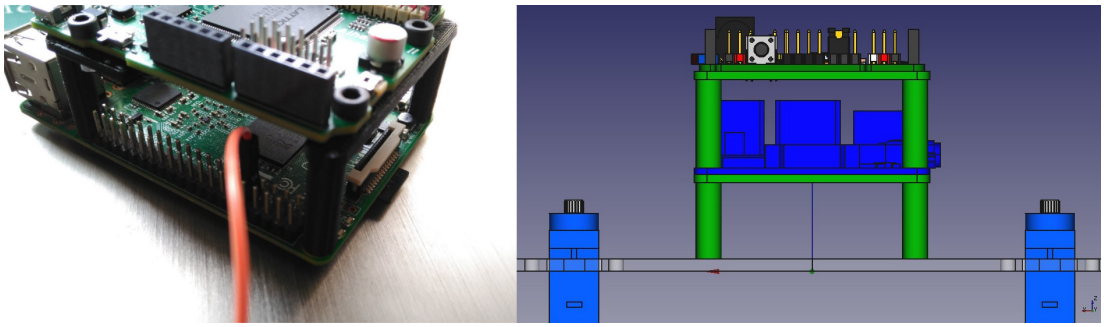


Fig. 5.40: Crabdle boards holders design.

The base has been designed taking into account the maximum printing volume of the typical 3D printers and following the same strategy as *Doodle*: design half of it and mirror the other half. The base is flat with the purpose of being capable of manufacture it using 3D printers, laser cutting machines, or *CNCs*. The rows of holes are for putting the boards in different positions to play with the center of mass if necessary.

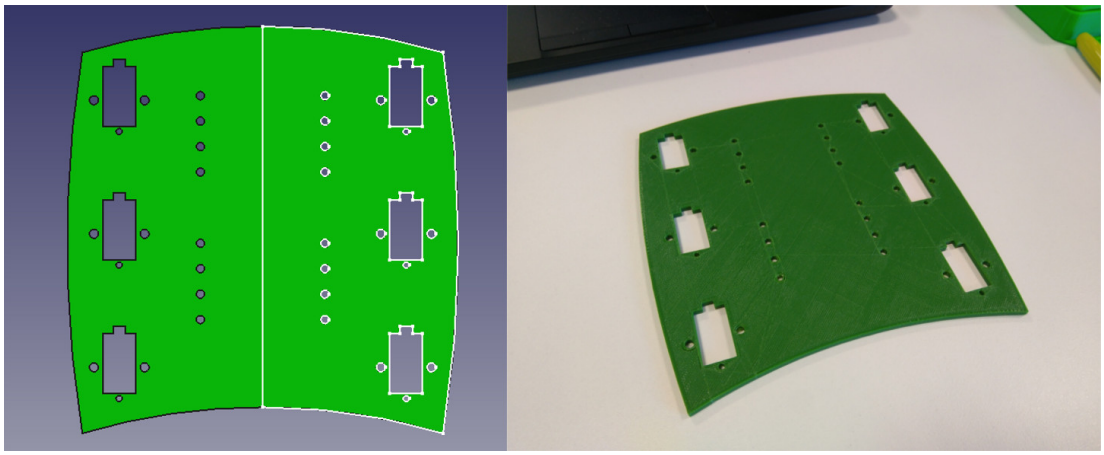


Fig. 5.41: Crabdle base design. The design has been created following the strategy of creating just half of the base (white lines) and mirror the other half.

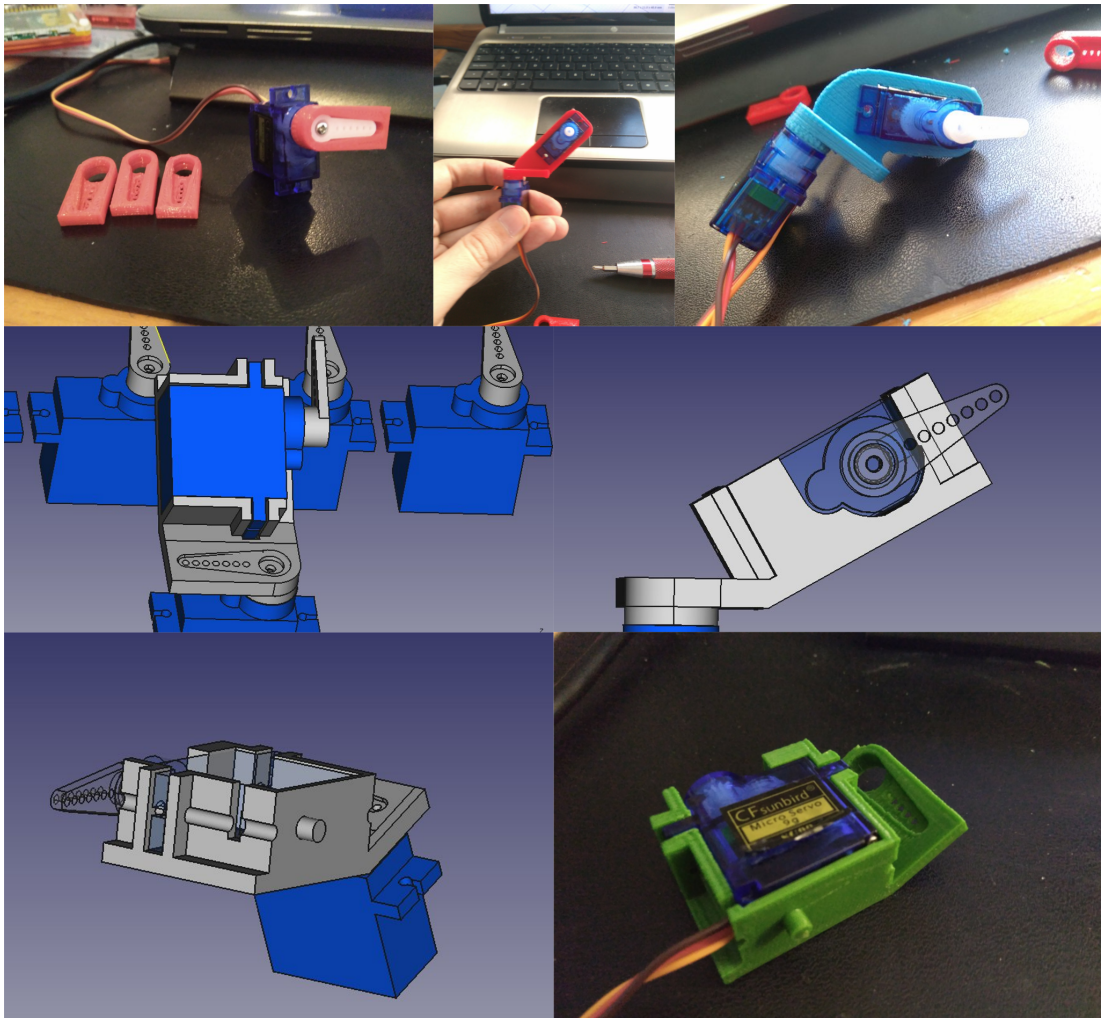


Fig. 5.42: Q1 leg iteration design. The first part is attached to the first servomotor horn and holds the second servomotor in an angle of 30°. A male axis formed by the horn of the servo motor and a pillar on the other side serves as rotation axis for the next part: Q2.

Once we have the base of the robot, it is time to create the design of the legs. In *Crabdle* each leg has two *DOF*, two parts named *Q1* and *Q2*. In this design, the first motor is attached to the base, and the second motor is attached to the leg *Q1* part. All the pieces created for this robot are designed following a methodology of iterations with physical test parts. In fig. 5.42 we can see some of the iterations done to reach the last design of the *Q1* leg part. In the end, the *Q1* design holds the second motor in a 30° angle. The most interesting thing about this part is that thanks to its design, the second motor does not need any screw to be attached to the *Q1* due to its *"click*

pressure system" and can be easily removed thanks to a hole in the bottom of the piece. This design greatly simplifies the assembly of the robot.

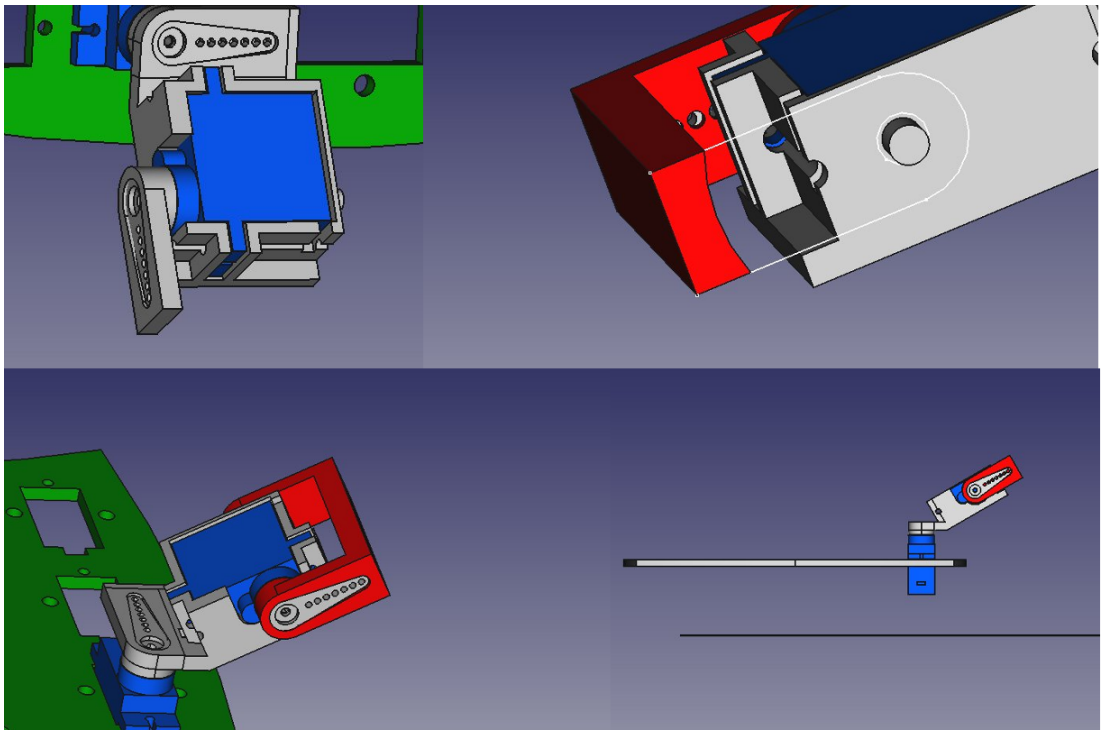


Fig. 5.43: Q2 leg design of the first part of the piece. To attach the Q2 part of the leg to the servo motor, we use the same strategy than Q1. Once we have the minimal design, it is time to design the shape of the leg, taking into account the height between the base, the servo motor, and the floor.

For the design of the *Q2* leg part there are two critical iterations. The first one (Fig. 5.43) is the attachment between the *Q1* part and the *Q2* part. Following the same strategy as the *Q1* attachment with the first servomotor, the *Q2* part is attached to the horn of the second servo motor in one side and attached using a hole to the male axis of *Q1* on the other side. With this iteration, the servo motor can freely move the future leg in the horizontal and vertical plane. Once we have the first iteration of the *Q2* part, it is essential to place all the components in their correct position before continuing with the design. The base has been placed at the desired height and the *Q1* part, the second servo motor and its horn form an angle of 30º with the floor. The height between the first part of *Q2* and the floor is key to create the rest of the leg.

Once we have the desired positions and height, the second iteration of *Q2*, the leg

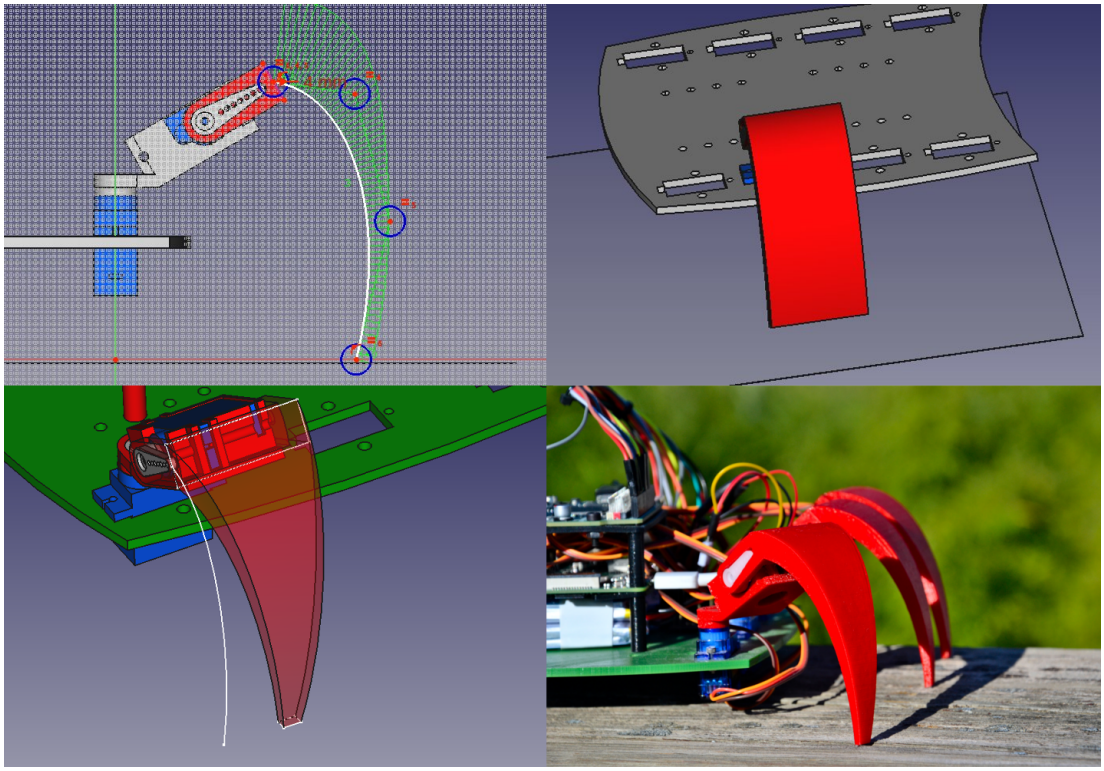


Fig. 5.44: The leg is created using bezier curves and an extrusion using the curve as the trajectory of the extrusion. For the final shape, two additional rectangle sketches mark the beginning shape and final shape that has to follow the extrusion of the leg.

itself, is created (Fig. 5.44). The leg is the most critical part of the whole design. It marks the correct locomotion of the robot and at the same time, the personality of it. For these reasons, the design has to be as clever as possible, combining a good mechanical design and an attractive appearance. In the case of *Doodle* the personality was marked by the insect hairs of the leg, for *Crabdle*, the personality is marked using a closed and soft curve that goes from the upper part of the servomotor to the floor. To create this part, *Bezier* curves have been used for creating the continuos and soft curve. Once the curve is created, it is necessary to use two more rectangles sketches that mark the beginning and final shape of the leg. With all the sketches defined, we can do an extrusion that starts with the big rectangle initial shape, follows the trajectory of the bezier curve, and ends the extrusion in the final small rectangle that touches the floor, resulting in a leg that reminds the legs of the arthropods.

One of the more interesting things about the *Q2* leg is that the part fits in the

axis of *Q1* by pressure, not being necessary the use of the horn screw to attach it to the servo motor. Thanks to the designs of *Q1* and *Q2* avoiding additional screws, the robot only uses 3 screws per leg using a total of 22 screws instead of 40.

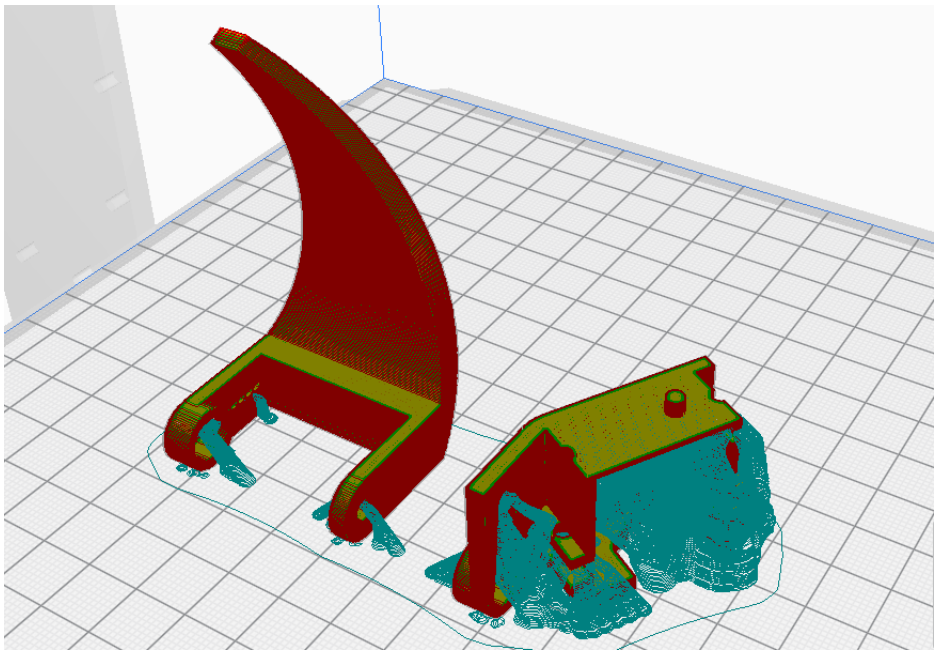


Fig. 5.45: The Q1 and Q2 legs parts are fast and easy to print, being possible to have a complete leg in 90 minutes. The use of tree supports improves the post-processing time.

The modular design also helps to improve the printing time. As we can see in fig. 5.45, a complete leg can be printed in 90 minutes. If we add the time of printing the base: 6 hours, we find that *Crabdle* can be printed in 15 hours. In the case that we create the base using a laser cutting or *CNC* to have a base made in wood or acrylic, the total time will be only 9 hours and a half.

The design of *Crabdle* has a lot of good things. The modular design allows us to re-use its legs in future robots, just changing the base. The base is easy to modify thanks to its flat design and can be manufactured using different technologies. The use of plastic is minimal for this kind of robot, the total printing time is acceptable, and the post-processing is just trivial. *Crabdle* is an excellent starting point to design more advanced robots in the future.

5.3.4 The low-level nervous system

Crabdle has a *low-level* system based on the experience earned developing *Doodle* and other experiments. The circuits are based on the principle of creating locomotions thought *CPGs* that are stored in *ROM/RAM* memories and implementing *low-level* sensors which behaviour and response usually are directly connected with the *low-level* actuators.

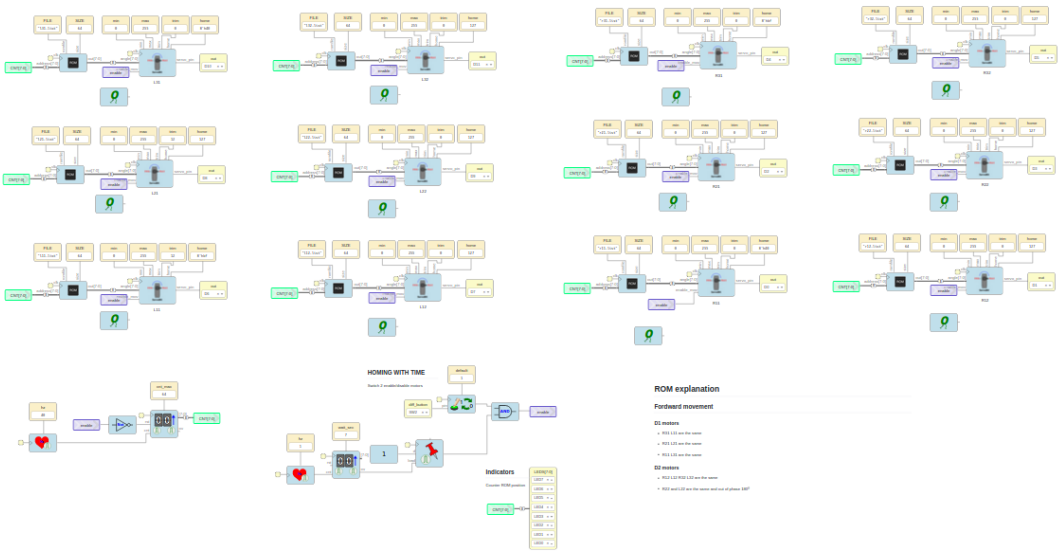


Fig. 5.46: Basic locomotion behaviour circuits in Crabdle using Icestudio.

As we did with *Doodle*, the circuits have been developed first with *Icestudio* (Fig. 5.46), being ported to pure *Verilog* code once the system properly works. We are going to use the graphical circuits of *Icestudio* to explain the *low-level* circuits.

As we can see in fig 5.47, the homing and initial circuit (right) are similar to the *Doodle* one, having an initial waiting time in the home position of the robot and manual homing control to stop the robot using the switch button 2 of the *Icezum Alhambra*. The *Crabdle* circuit is a more complicated circuit than the *Doodle* one, for this reason in this case connection labels are used instead of normal cables to connect the different modules, having a visually clean circuit easy to expand and edit. The speed control speed circuit is the same as *Doodle*, a counter in *Hertzs* control the speed at which every slot of the *ROM* memory is read, controlling the velocity of the robot. The counter can be reset by the homing module. The output of the counter (CNT[7:0]) will

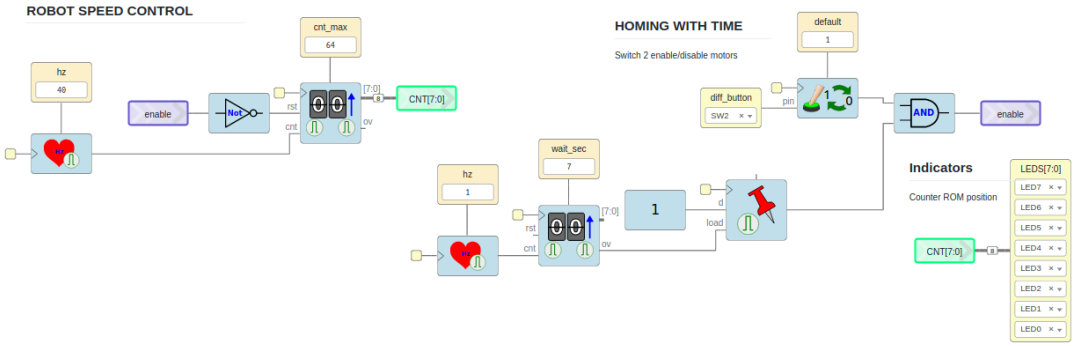


Fig. 5.47: Speed circuit (left), initial and homing circuit (center) and LEDs indicators connected to the speed ROM counter of the robot (right).

be connected to each leg module and the array of LEDs of the *Icezum Alhambra* to visualize the velocity of execution.

However, the core of the circuits are the leg modules (Fig. 5.48). The module takes the counter position output and reads the correspondent position of the *CPG* signal stored in the *ROM* module. In this case, the robot is bigger and the leg movements have to travel more space and for this reason, we need more resolution in the *CPG* signals to achieve a soft and continuous robot movement, having a *ROM* of 64 slots instead of 32 that store the positions using one byte.

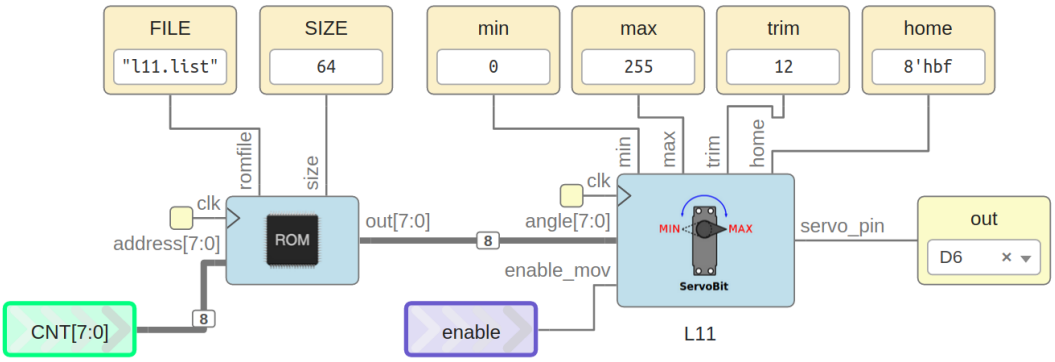


Fig. 5.48: Crabdle leg module. The module reads the CPG signal stored in the ROM memory and passes each position of the signal to the PWM servo motor circuit that has been improved with a trim calibration control.

The *CPG* signals are loaded to the *ROM* using the *romlist* files. These signals and files are generated using the already mentioned *oscillator rom generator* script (88) by a human or the *high-level* system of the robot.

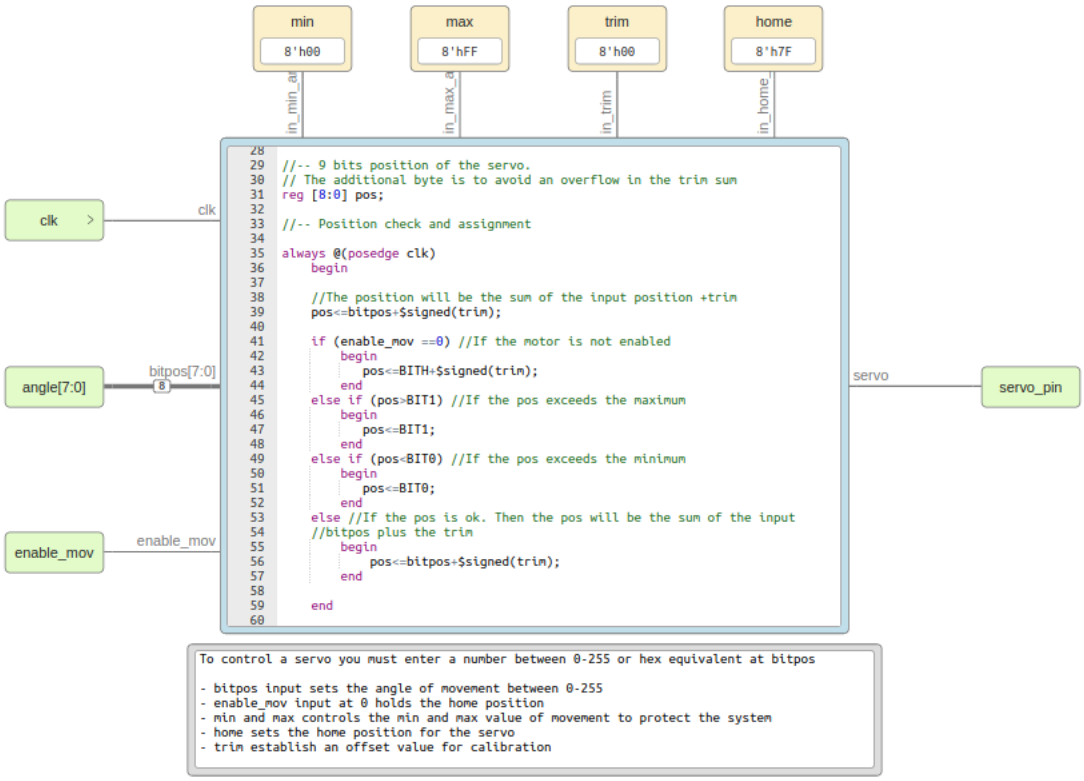


Fig. 5.49: Inside of the new PWM servo motor module. The trim position is added to the desired position. A register of 9 bits is used instead of 8 to avoid overflow problems.

A significant part is the new PWM servo motor control block (Fig. 5.49). The block has been improved with a new parameter called *trim*. Due to the complexity of *Crabdle* and the mechanical resolution of the cheap *SG90* servomotors, the mechanical calibration of each motor (basically putting the horn in the right position to match the control position) is not enough to achieve a good calibration in *Crabdle* legs. For this reason, a new trim position calibration has been added. This trim offset allows to properly calibrate the motor without the need to offset the real *CGP* signal. With the *min* and *max* parameters to avoid mechanical problems, the *home* position and the *trim* calibration, the new block is perfect for robotics applications, avoiding errors and saving a lot of time.

Once the basic behavior of *Crabdle* is working properly and the basic locomotions work, the circuits created with *Icestudio* have been ported to pure *Verilog* code.

```
module crabdle(
  input wire CLK,
  input wire SW1,
  input wire SW2,

  //Motors wires pinout
  output wire D0,
  output wire D1,
  output wire D2,
  output wire D3,
  output wire D4,
  output wire D5,
  output wire D6,
  output wire D7,
  output wire D8,
  output wire D9,
  output wire D10,
  output wire D11,


  //LEDs wires pinout
  output wire LED0,
  output wire LED1,
  output wire LED2,
  output wire LED3,
  output wire LED4,
  output wire LED5,
  output wire LED6,
  output wire LED7
);
```

```
//-- Robot speed or rhythm --//

//Heartrate
heartrate_hz #(.HZ(40))
  main_heartrate(
    .clk(CLK),
    .o(out_main_heartrate)
  );

//Modular up counter of 8 bits
counter_8_bits #(.M(64))
  counter_roms(
    .clk(CLK),
    .rst(~out_enable),
    .cnt(out_main_heartrate),
    .q(out_counter_roms)
  );

//-- HOMING WITH INITIAL TIME --//
homing_with_time #(.wait_seconds(7))
  home_and_enable(
    .clk(CLK),
    .in_enable(SW2),
    .enable(out_enable)
  );
```

```
//-- LEGS CONTROL --//

//R11
rom_servo_crtl #(
  .ROMFILE("./romlists/r11.list"),
  .ROM_SIZE(64),

  .HOME(8'h40),
  .TRIM(0),
  .MIN(0),
  .MAX(255)
)
  r11(
    .clk(CLK),
    .position(out_counter_roms),
    .enable(out_enable),

    .servo_out(out_r11)
  );
```

Fig. 5.50: Main Verilog modules of the basic behaviour of the *Crabdle* low-level system.

In fig. 5.50, we can see some of the main modules of the robot. Naturally and following the modular design typical from *Verilog*, these modules, like the *homing with initial time* or the *leg control* are created from other small modules and circuits. Thanks to the design approach used in this work, using *Icestudio* for fast prototyping and converting it later to the pure *Verilog* code without too much effort, the circuits code are heavily modularised, being easy to change their main parameters, improve the modules and add new circuits and behaviors in an easy way not only for a human but also, as we are going to see in the following section, for the *high-level* system of the own robot.

The development of the first locomotion can be seen at (101), and the fully working robot locomotion at (102).

5.3.5 The high-level brain

The *high-level brain* of *Crabdle* is a full high-level control written mostly in *Python* that runs inside the *Raspberry Pi* of the robot, its *CNS*.

The main functions for the *high-level* system are all the abstract tasks related with the robot, the processing of information, and the managing of complicated sensors and actuators that have not been implemented in the *low-level* system. The *high-level* system is, in general, a traditional robot control system that can be even implemented in *ROS* (section 5.2.5). The general structure of the program can be seen in fig. 5.51.

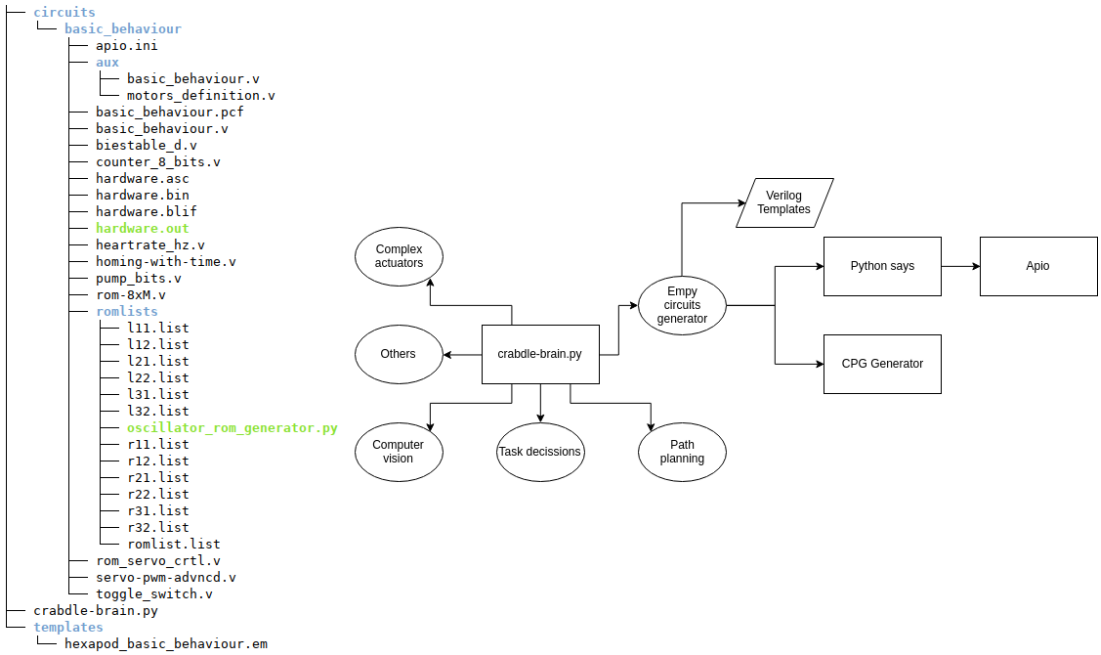


Fig. 5.51: Crabdle brain folder tree and a little example of the different tasks (circles) and programs (squares) used in the main program.

In the case of *Crabdle* a general *Python* program called *crabdle-brain.py* manages all the activities of *Cradble*, including the most interesting one, the control and new generation of circuits for the *low-level* system using the *high-level* program. For this purpose, there are two important folders: *templates* and *circuits*. The *templates* folder contains the verilog circuits templates that the *crabdle-brain* uses to generate new circuits using the library *Empy*. The *circuits* folder is where the generated circuits are stored, including the auxiliar files like used verilog modules, apio init files. auxiliar files

generated synthetizing circuits using *Apio* and the binaries that will be uploaded to the *FPGA*. Inside the case of the *basic_behaviour* circuit, there is a sub-folder called *romlists* that contains the *CPG* signals that will be loaded into the *ROMs*.

5.3.5.1 The Empy Verilog templates

As we said in *Doodle* (section 5.2.1.5), the *high-level* brain program uses the library *Empy* to generate *Verilog* circuits for the nervous system of the robot. *Empy* (91) uses files with the extension *.em* that are divided in two main parts: the python code execution (using the special character @ and brackets) and the verilog code of the template. At the beggining there is always a group of *Python* code that takes the string received by the *crabdle-brain* program and converts the data into *Python* variables.

```
1  //PYTHON CODE
2  @{
3  arguments=empy.argv.split(";")
4  HZ=arguments[0]
5  init=arguments[1]
6
7  # Motors names to build the verilog circuit
8  motors_names=["r11","r12","r21","r22","r31","r32",
9                "l11","l12","l21","l22","l31","l32"]
10
11 # Motor homes parameters for each motor
12 motors_home=["8'h40","127","127","127","8'hbf","127",
13               "8'hbf","127","127","127","8'h40","127"]
14
15 # Motor trims
16 motors_trim=["0","0","0","0","0","0","12","0","12","0","0","0"]
17
18 motors=arguments[2].strip(']['').split(', ')
19
20 print("//Circuit arguments")
21 for element in arguments:
22   print("//Element: "+str(element))
23 }@
```

Listing 5.3: basic_behaviour.py initial Python code

In the case of the *basic_behaviour.em* of *Crabdle*, the *crabdle-brain* program decides the velocity of execution of the robot (*Hz*), the implementation or not of the init module, and which motor modules are going to be created and which, for whatever

reason, not. These arguments are going to be used by the *Empy* library to generate the correspondent *Verilog* circuits. The template is full of special execution code for generating different parts of the circuits, but probably the most interesting case is the leg modules generation.

```
//-- LEGS CONTROL --//
@{
for i in range(len(motors)):
  motor_module="//"+motors_names[i]+"\n"+\
  "rom_servo_crtl #(\n\
  .ROMFILE(\"./romlists/"+motors_names[i]+".list\"),\n\
  .ROM_SIZE(64),\n\
  \n\
  .HOME("+motors_home[i]+"),\n\
  .TRIM("+motors_trim[i]+"),\n\
  .MIN(0),\n\
  .MAX(255)\n\
)"+"\n"+\
" "+motors_names[i]+"(\n\
    .clk(CLK),\n\
    .position(out_counter_roms),\n\
    .enable(out_enable),\n\
    \n\
    .servo_out(out_"+motors_names[i]+")\n\
);\n"
  if (motors[i]=='1'):
    print(motor_module)
}@
```

Listing 5.4: basic_behaviour.py leg modules generation

In this case there is an *Empy* python code execution between the @{...} that will control the *Verilog* code generated. A *for* loop iterates in each motor of the legs, and generates the leg modules control with the custom parameters, like the *romlist* file for the *CPG* signal, the *trim*, *homing* and *min* and *max* params, and the name of the module. The *if* conditional at the end of the code decides if the motor module is going to be implemented in the *Verilog* file or not, depending on the decision passed by the crabdle-brain program. The code is short and powerful, being able to generate all the *Verilog* leg modules that we want with just a bunch of *Python* arrays.

5.3.5.2 The Crabdle Brain program

The Crabdle main program execution *crabdle-brain.py* is a *Python* program that can run different programs and task inside the *Raspberry Pi*. The most interesting part of the program developed for this work is the control of the program of the *low-level system* or *PNS*. In the last section, we have seen how the *Verilog* circuits are created using the templates files and the library *Empy*. In this section, we will see how the main program passes the desired arguments to the *Empy* library and how the generated circuits are verified, synthesized, and uploaded to the *FPGA*.

```
# Import python says
import pythonsays as pysy

class HexapodPNS (pysy.VerilogBlock): #Hexapod PNS control

    def __init__ (self,circuits_path,template_name,argv,output_file):
        super().__init__(circuits_path,template_name,argv,output_file)

        # Default circuit options
        self.HZ=60
        self.init_circuit=True
        self.motors=[
            1,1, 1,1, 1,1, # R11,R12, R21,R22, R31,R32
            1,1, 1,1, 1,1  # L11,L12, L21,L22, L31,L32
        ]

        # Circuit setup
        if(self.argv==""): #If arguments empty->default values
            self.gen_circuits_options()

    def gen_circuits_options(self): # Argms separator: ";"
        self.argv=str(self.HZ)+";"+str(self.init_circuit)+";"+str(self.
    motors)


    def gen_romlist(self,signal,min,max,size,path):
        romlist=pysy.rom_generator.RomGenerator(signal,min,max,size,path)
        romlist.build_save()
```

Listing 5.5: crabdle-brain.py HexapodPNS class

As we can see in the code above, we have created a *Python* class named *HexapodPNS* to control the *low-level* system of *Crabdle*. This class inherits from the class *Verilog-*

Block that is present in the *Python module* that we have created to deal with all the tasks related with the generation, verification, sinthesis and upload of circuits using *Open Source FPGAs*: *PythonSays* (90). The attributes of the class are the atributtes inherited from the *Python says* class and the attributes that are going to be useful to describe *Crabdle*: robot speed (*HZ*), the implementation or not of the initial homing circuit, and the motors list (*self.motors*) that controls with motors controls circuits are going to be described in the low-level and which not. The *gen_circuits_options* function add the parameters to the *self.argv* function, that will be used by *Empy* to build the circuits, and the function *gen_romlist* call the *PythonSays* functions dedicated to generate the *CPGs* signals romlist files from the *oscillator_rom_generator.py* functions (88).

```
def basic_behaviour():
    hz_list=["20","40","60","80"] # Hz

    # Legs lists
    r11_list=[[45,75],[35,75],[25,75],[15,75]] # Step amplitude
    l31_list=r11_list

    r31_list=[[105,135],[105,145],[105,155],[105,165]] # Step amplitude
    l11_list=r31_list

    while(True):
        for i in range(len(hz_list)):
            print("\n\n<-----Creating a new circuit with a value of "+
    hz_list[i]+"----->")

            # Generation of the low level
            hexapod_pns=HexapodPNS("./circuits/basic_behaviour","./
    templates/hexapod_basic_behaviour.em","","basic_behaviour.v")

            hexapod_pns.HZ=hz_list[i]
            hexapod_pns.init_circuit=True
            hexapod_pns.gen_circuits_options()

            # Generation of Rom list that defines the leg movements
            hexapod_pns.gen_romlist("sin",r11_list[i][0],r11_list[i
    ][1],64,"./circuits/basic_behaviour/romlists/r11.list")
            hexapod_pns.gen_romlist("sin",l31_list[i][0],l31_list[i
    ][1],64,"./circuits/basic_behaviour/romlists/l31.list")
            hexapod_pns.gen_romlist("sin",r31_list[i][0],r31_list[i
    ][1],64,"./circuits/basic_behaviour/romlists/r31.list")
```

```
26              hexapod_pns.gen_romlist("sin",l11_list[i][0],l11_list[i
    ][1],64,"./circuits/basic_behaviour/romlists/l11.list")
27
28              #Clean, verify, build and upload of new circuits
29              hexapod_pns.generate()
30              hexapod_pns.to_fpga(clean=True)
31              time.sleep(60)
```

Listing 5.6: crabdle-brain.py basic_behaviour test class

For testing the correct implementation of the *high-level* system control of the *low-level* circuits, the function *basic_behaviour* has been created. As it can be seen in the code above, the function create circuits with different speeds (*hz_list*) and different legs *CPG* signals with different leg step amplitudes (*r11_list*). The function makes a *for* loop with the different parameters described in the list, creating the *HexapodPNS* object, putting the desired values of speed and others, and generating the different *CPG* signals used for each leg. After that, the function calls the functions inherited from *PythonSays* to generate the new circuit and verify, synthesize, and upload the circuit to the *FPGA*. Finally, the program waits one minute and implement the next new circuit with the new elements and *CPG* signals in the *Open Source FPGA*.

With the general architecture of the *high-level* system implemented, and able to control and create new circuits for the *low-level* system without human intervention, the *Crabdle* implementation of the bio-inspired control architecture proposed in this work is complete. *Crabdle* is an easy to manufacture and build robot perfect for many experiments related to the proposed bio-inspired architecture and the use of *CPG* signals for locomotion.

Thanks to *Crabdle*, the viability of the proposed bio-inspired control system has not only been demonstrated, it also has been successfully applied and integrated into a complex bio-inspired robot, being *Crabdle* the base in which new robots will be developed in the future to implement new features and do new experiments with this bio-inspired robotics architecture.

5.4 Spreading the knowledge

During the development of this project, different activities and publications have been done. The idea was to make a contribution to the *Open Source* and *Maker* community and expand and build new tools for the use of everyone. Trying at the same time to make a small contribution in the research field of robotics sharing the work developed during this master thesis.

5.4.1 The Maker community: the informal way

Fig. 5.52: Different workshops, talks, and events during the development of this project.

As we said before, this work was planned from the beginning as an *Open Source* project with the goal of contribute to the *Open Source* and *Maker* community. The most important action was sharing the sources of the project (81). But we have also spread the knowledge in different ways like participations in Maker faires like Madrid (103)

and Oshwdem (104), the talks about this thesis titled *"Robots, gatitos y FPGAs libres"* in Arduino Day Zaragoza (105) and Maker Faire Galicia (106), and some workshops about building *Doodle* in Oshwdem (86) and building an *Enigma Machine* using *Open Source FPGAs* in T3chfest (107). Having also some related informal interviews (108) and (109). Sharing this work with different people from different fields, like biology, mechanics, general public and *Makers* has been amazing and priceless, having the opportunity to learn a lot from all of them and apply this knowledge into the project.

5.4.2 Sharing with other professionals: the formal way

During the development of the project, part of this master thesis was presented and published as a paper in the *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018)* (6). The conference was an incredible opportunity to share with robotics professional with an impressive knowledge of their fields the ideas developed in this master thesis and learn many things about some of the fields related with it, like bio-robots and soft-robots, control systems and leg locomotions.

Fig. 5.53: IROS 2018 Conference presentation of the paper based on this work.

6

Conclusions, results and future

In this master thesis we have presented an alternative to the traditional robotics control architectures. The alternative is a bio-inspired control system inspired in the common animal nervous system, where there is a clear división between the *high-level system*: the brain and spinal cord (*CNS*), and the peripheral nervous system (*PNS*), the *low-level system*. After studying the actual state of the art in the bio-inspired robotics field, the use of *Central Pattern Generators* (*CPG*) and the use of *FPGAs* in robotics, we have proposed the idea of a *bio-inspired* control architecture with a traditional *high-level* control using microprocessors and an innovative *low-level* control based on the use of *CPGs* for general locomotion and the new possibilities that the *Open Source FPGAs* offers to the robotics field.

The architecture proposed in this work is based on the general idea of splitting the tasks of the robot in *high-level* and *low-level* tasks. While the *high-level* tasks are more general and abstract, needing great power processing and are not being easy to model even in software, the *low-level* tasks are more simple, repetitive and concise tasks that are common in the robot and does not usually need too much power processing, but need great execution speed and should be executed in parallel. The *high-level*, based on the traditional approach of using one or more microprocessors and develop programs through software, is very powerful and requires relatively little effort to implement a new task, but it is not really prepared to work in parallel with multiple tasks and the same time without finally losing performance in most of them. On the other hand, the *low-level* system is based on the idea of modeling the tasks using digital circuits, hardware, described inside an *FPGA*. It requires more effort than the traditional software approach

but at the same time allows the robot to execute critical tasks as fast as possible and in real parallelization without losing performance in any moment, no matter the number of tasks. Both approaches have advantages and disadvantages; however, if we combine them, we can use the great advantages of each level control to create robots with an incredible level of performance. During this work different experiments to test the viability and find the situations where this bio-inspired architecture is more useful than a more traditional approach has been done. The robot *Doodle* has demonstrated the full viability of the control system, while the robot *Crabdle* has successfully implemented the architecture in a more complex robot. After designing, building, programming and creating circuits for these robots, we have found that the advantages of using *Open Source FPGAs* for the *low-level* system are relevant and it should be taken into account in the design of a robot.

The velocity of a digital circuit created inside an *FPGA* is key to create better locomotions, especially in complicated robots with a lot of degrees of freedom, where the movement of multiple motors in synchrony at the same time is critical. The sensorization in the *low-level* system is also a great approach that allows the robot to have more sensors, take better decisions and react faster, without saturating the *high-level* microprocessor. With the implementation of the different robots of this work, we have discovered that, with the proposed bio-inspired architecture, we can create bio-inspired robots where adding more and more legs and controlling them without losing performance is just trivial, as long as there is still space in the *FPGA*, being almost the same controlling a robot with 3, 12 or 36 *DOF*.

During the development of the robots created for this project, the use of *Central Pattern Generators* or *CPGs* has been crucial. The use of *CPGs* improves a lot the locomotion of the robots, having a better grip and bigger distance step for each leg, and improving the softness and precision of the movements. Also, the use of *CPGs* is fascinating when we design the different locomotions of the robot. Although controlling the legs changing the different parameters of a signal can be tricky and complicated to understand at the beginning, after some time this locomotion paradigm is revealed as a great, simple and straightforward system to control robots with a lot of *DOF* and complicated locomotion with less effort than other traditional approaches. However, during the development of the work, only relatively simple signals, like square, triangular and sinusoidal have been used. As the state of the art shows, the implementation of *CPGs*

in robots and their presence in animals have more complex and combined signals that require more mathematical study and effort, being implementing more complex *CPGs* one of the future research jobs to do out of this work.

Another exciting field of study, discovered during the implementation of this work, has been the creation by the *low-level* system of reflex actions. The use of reflexes actions in the implementation of *Doodle* and the *mechatronic eye* has shown and interesting feature of the proposed bio-inspired architecture. Creating reflex actions using the hardware of the *low-level* system is an interesting approach that should be studied deeper in the future. The *low-level* sensors can be directly connected to the actuators, reacting in almost real time to events, and without the *high-level* intervention, improving the speed and acting in parallel without worrying about the number of sensors or in which moment or task we are. The implementation of the bio-inspired architecture with the reflex movements is an interesting approach to implement more complex robotics nociception systems and improve the proprioception of the robots, being a future work the expansion and implementation of more reflex actions using the proposed bio-inspired architecture of this work.

The use of *Open Source FPGAs* open new possibilities in the robotics field. Thanks to the new *Open Source* tools, we can create robots that can modify their own nervous system almost in real time and without human intervention. This interesting feature has been implemented in the robots *Doodle* and *Crabdle*, where the *high-level* system can create new circuits for the *low-level* system and verify, synthesize and upload them to the *low-level* system *Open Source FPGA*. This fascinating workflow of creating and uploading new circuits in real time without human intervention takes around 15 seconds in a normal laptop, and around 90 seconds in the *Raspberry Pi* boards used in this work. Although it could be better to have the time of 15 seconds in the robots created on this work, it is just a hardware limitation that can be solved with more powerful micro-computer boards and the progressive improvement in performance of the *Open Source* tools. This feature of creating new circuits and automatically upload them was not possible using the traditional tools of the main *FPGA* manufacturers. With the new tools developed by the community, it is possible to develop custom tools for new research fields where the *FPGAs* were limited or were not normally used. In this sense this work has contributed to improve and expand some of the actual *Open Source*

FPGA tools, being a future work to continue to experiment and expand the possibilities of the *Open Source FPGAs* in the robotics field.

In this work, the viability of the bio-inspired proposed architecture control has been tested using different robots and experiments. However, there is still too much work to do. Future works related to this master thesis are the implementation of more complex *CPGs* in the architecture or the expansion of the architecture, adding more complex tasks to the *high-level* and *low-level* systems. Another interesting point to expand further is the massive use of *low-level* sensors to create complex reflex movements with interesting implementations and interactions. Other is the use of *machine learning* in the *high-level* control to create robots able to learn from new situations, modifying in real time their own nervous system to successfully solve new problems and adding the power of a robotics cloud to expand this learning getting situations already experimented by other robots to apply to their own nervous system.

The base for all of these future lines of work are already tested, implemented, and are working in the robots and experiments developed during this work. For these future works, there is also another line of working, the implementation of the bio-inspired architecture in more robots, and the creation based in *Crabdle* of a new one robot. This robot will be strongly based in the modular design of *Crabdle*, creating modules with legs than can be connected together to have a caterpillar-like robot, with a lot of legs and sensors to control in parallel by the low-level system using *CPGs* signals that spread across the legs of the robot. An additional feature of the robot will be the possibility of separate the different modules to work independently, or connect the modules to work together.

As a final conclusion, the proposed hybrid bio-inspired robot architecture of this work is an interesting and real alternative to more traditional controls. The possibility of putting the repetitive and simple tasks in the *low-level* system to work faster and in parallel, freeing up the *high-level* system for more critical and abstract tasks, is an interesting approach that already works in nature. The use of *CPGs* is fundamental and will be crucial in future leg locomotions and the use of *Open Source FPGAs* open new possibilities in the robotics field. For all the reasons presented during this work, we truly believe that future robots will have bio-inspired architectures like the proposed in this thesis. Combining as nature does, the best things of the abstract and the physical world, the best things of software and hardware, the best things of us.

Appendix A

Legal-ethical impact, WBS structure, Gantt diagram and Budget

In this appendix we will talk about some extra but important aspects of the developed project, such as the legal and ethical impact, and the professional responsibility. Taking into account ecological, economical and social aspects, negatives and positives.

Every project needs a good planning and prevission of time and effort. It is very important to clearly identify the goals of the project and, based on these goals, create the different tasks and sub-task to reach them. In this sense, the *WBS* and the *Gantt* diagrams are very useful to really understand the scope and effort of the project.

Finally, the budget is shown. The budget is divided into the different robots and experiments developed in this work. It assumes that for each robot and project, the materials and components are new and not reused. This is useful to estimate the real cost of each robot and experiment. However, the reality has been different. In most of the cases, the main components of the robots were already acquired or have been reused between the different robots. Other materials, like *PLA* plastic, have been obtained cheaper than the commercial price reflected in the budget.

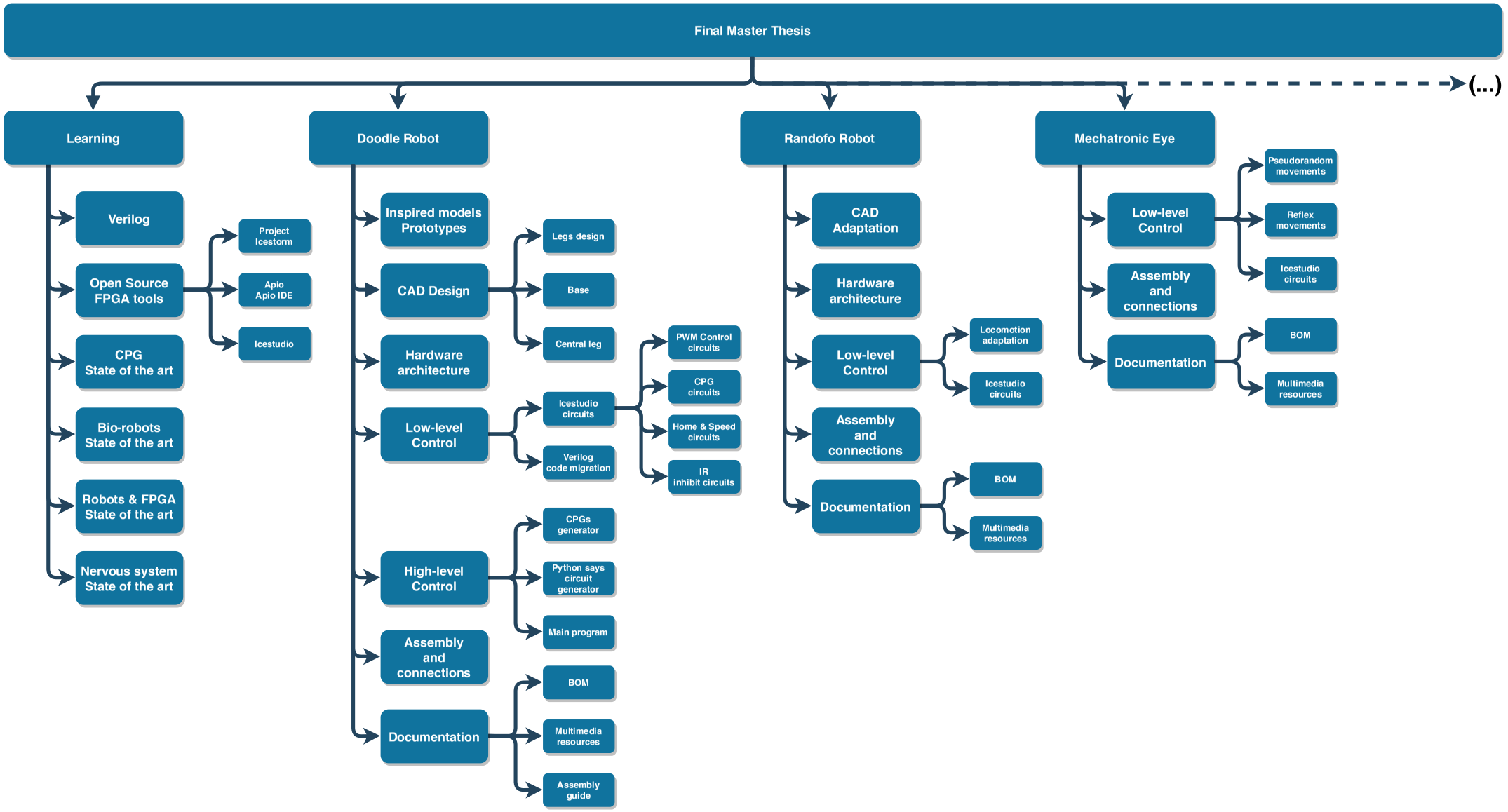
Every project and research work has a legal and ethical impact related to its development. In the case of this project, there is an ecological impact, a possible ethical problem, and a positive impact in the form of divulgation and research support.

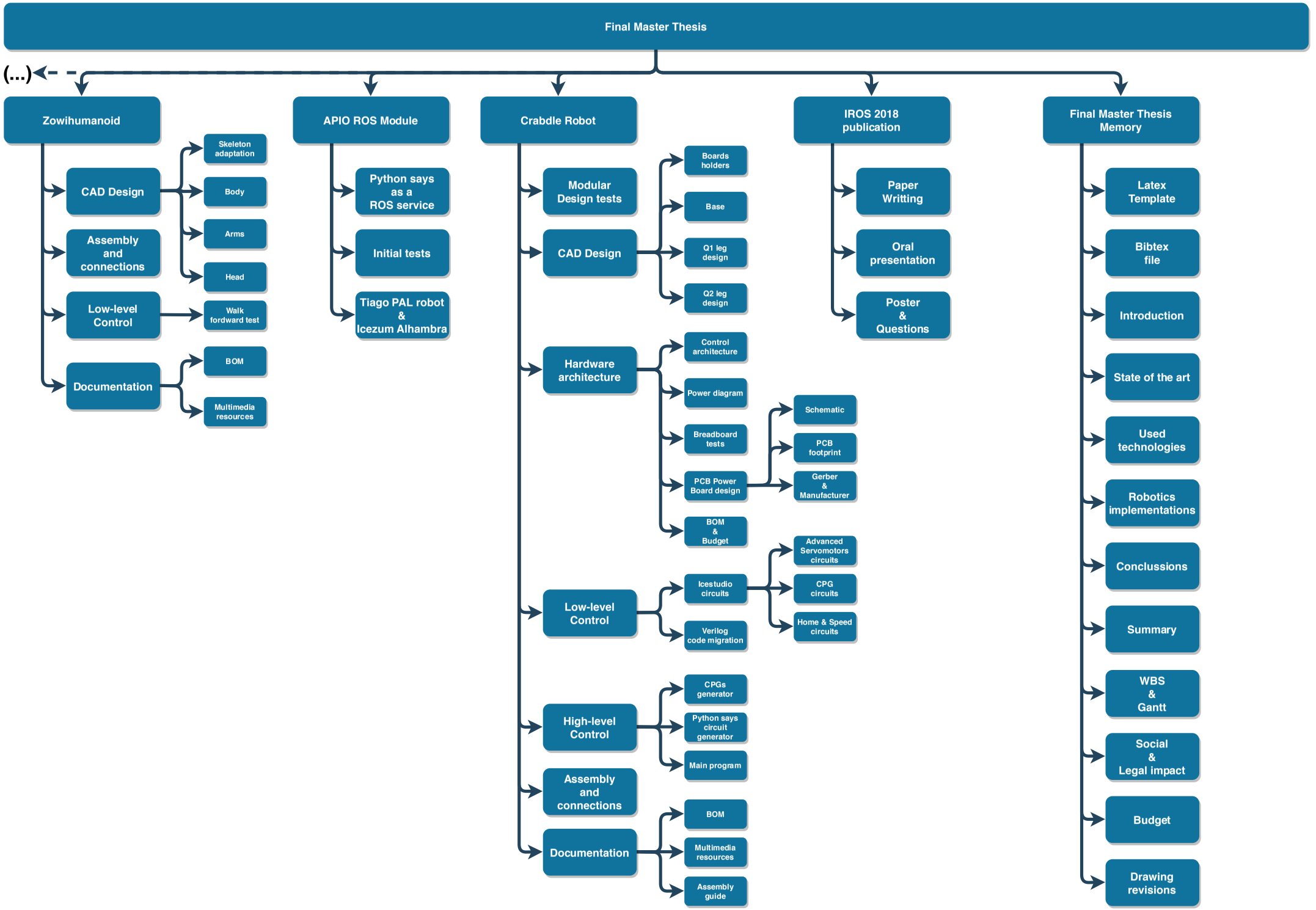
The ecological impact of the project can be considered minimum but significant and has to been taken into account. Most of the development of this project has been done in larger hours of computer power and, more importantly, digital fabrication using tools like 3D printers with a relevant consumption in electricity. The parts of the robots and experiments made with 3D printers have been manufactured using *PLA* plastic. The PLA is a biodegradable plastic that requires a proper and advanced recycling system. The electronics and mechanical components, on the other hand, will be reused for new robots and projects and properly send to a recycling station at the end of their life. Multiple trips around the country, generally on train, has been done to spread the project, with the ecological impact that implies.

We consider that there are not critical ethical problems in the development of this project. Although it is possible to defend that the creation and use of *Open Source* tools in the *Lattice ICE40 FPGAs* can have a negative economic impact in the sales of the company *Lattice*, the reality is quite the opposite. Thanks to the creation and use by the community of *Open Source Tools* and the design and fabrication of new boards with these *ICE40 FPGA*, *Lattice* has sold a lot of more units of this model, and a discrete but new blue ocean related with the hobbyist and research has been created.

On the other hand, the development of this project has been extremely positive in many aspects, being an *Open Source* project with robots designed with a strong *STEM* and educational aspect. A lot of new tools and materials have been created, and some of them are used for educational purposes by the community. The non-formal divulgation of the project, in the form of workshops and talks, has contributed to the spread of the *Open Source* community and the creation of new tools and educational materials.

Besides, the diffusion of this research project has the aim of being a base to new related research in the robotics field using *Open Source FPGAs*. The desire of the author is that this work will soon be out of date because of the new tools and works developed from it.


Final Master Thesis
(...)
Learning
Verilog
Open Source FPGA tools
Project Icestorm
Apio Apio IDE
Icestudio
CPG State of the art
Bio-robots State of the art
Robots & FPGA State of the art
Nervous system State of the art
Doodle Robot
Inspired models Prototypes
CAD Design
Legs design
Base
Central leg
Hardware architecture
Low-level Control
Icestudio circuits
PWM Control circuits
CPG circuits
Home & Speed circuits
IR inhibit circuits
Verilog code migration
High-level Control
CPGs generator
Python says circuit generator
Main program
Assembly and connections
Documentation
BOM
Multimedia resources
Assembly guide
Randofo Robot
CAD Adaptation
Hardware architecture
Low-level Control
Locomotion adaptation
Icestudio circuits
Assembly and connections
Documentation
BOM
Multimedia resources
Mechatronic Eye
Low-level Control
Pseudorandom movements
Reflex movements
Icestudio circuits
Assembly and connections
Documentation
BOM
Multimedia resources


Final Master Thesis
(...)
Zowihumanoid
CAD Design
Skeleton adaptation
Body
Arms
Head
Assembly and connections
Low-level Control
Walk fordward test
Documentation
BOM
Multimedia resources
APIO ROS Module
Python says as a ROS service
Initial tests
Tiago PAL robot & Icezum Alhambra
Crabdle Robot
Modular Design tests
CAD Design
Boards holders
Base
Q1 leg design
Q2 leg design
Hardware architecture
Control architecture
Power diagram
Breadboard tests
PCB Power Board design
Schematic
PCB footprint
Gerber & Manufacturer
BOM & Budget
Low-level Control
Icestudio circuits
Advanced Servomotors circuits
CPG circuits
Home & Speed circuits
Verilog code migration
High-level Control
CPGs generator
Python says circuit generator
Main program
Assembly and connections
Documentation
BOM
Multimedia resources
Assembly guide
IROS 2018 publication
Paper Writting
Oral presentation
Poster & Questions
Final Master Thesis Memory
Latex Template
Bibtex file
Introduction
State of the art
Used technologies
Robotics implementations
Conclussions
Summary
WBS & Gantt
Social & Legal impact
Budget
Drawing revisions

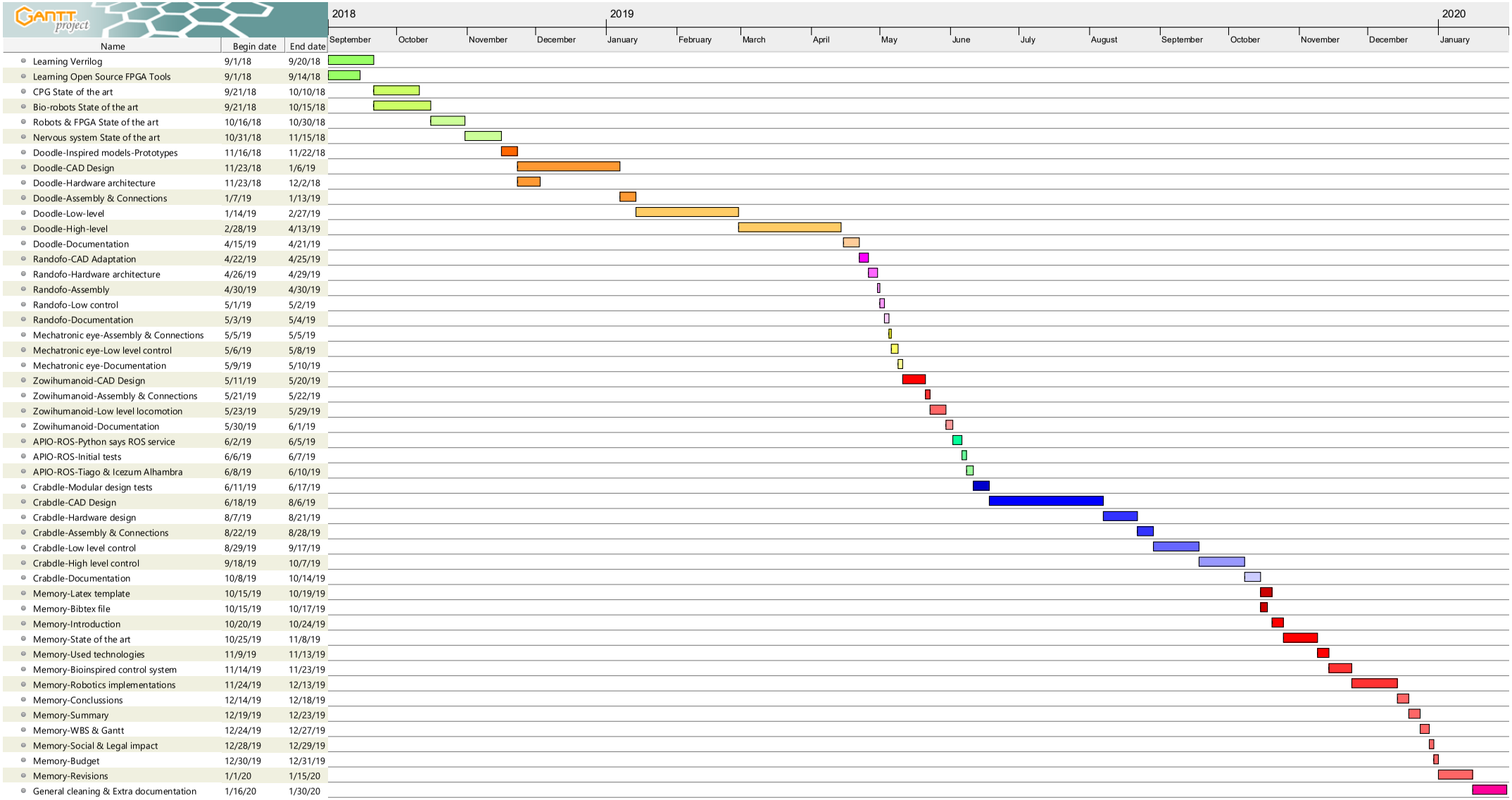
Tasks

Name	Begin date	End date
Learning Verrilog	9/1/18	9/20/18
Learning Open Source FPGA Tools	9/1/18	9/14/18
CPG State of the art	9/21/18	10/10/18
Bio-robots State of the art	9/21/18	10/15/18
Robots & FPGA State of the art	10/16/18	10/30/18
Nervous system State of the art	10/31/18	11/15/18
Doodle-Inspired models-Prototypes	11/16/18	11/22/18
Doodle-CAD Design	11/23/18	1/6/19
Doodle-Hardware architecture	11/23/18	12/2/18
Doodle-Assembly & Connections	1/7/19	1/13/19
Doodle-Low-level	1/14/19	2/27/19
Doodle-High-level	2/28/19	4/13/19
Doodle-Documentation	4/15/19	4/21/19
Randofo-CAD Adaptation	4/22/19	4/25/19
Randofo-Hardware architecture	4/26/19	4/29/19
Randofo-Assembly	4/30/19	4/30/19
Randofo-Low control	5/1/19	5/2/19
Randofo-Documentation	5/3/19	5/4/19
Mechatronic eye-Assembly & Connections	5/5/19	5/5/19
Mechatronic eye-Low level control	5/6/19	5/8/19
Mechatronic eye-Documentation	5/9/19	5/10/19
Zowihumanoid-CAD Design	5/11/19	5/20/19
Zowihumanoid-Assembly & Connections	5/21/19	5/22/19
Zowihumanoid-Low level locomotion	5/23/19	5/29/19
Zowihumanoid-Documentation	5/30/19	6/1/19
APIO-ROS-Python says ROS service	6/2/19	6/5/19
APIO-ROS-Initial tests	6/6/19	6/7/19
APIO-ROS-Tiago & Icezum Alhambra	6/8/19	6/10/19
Crabdle-Modular design tests	6/11/19	6/17/19

Tasks

Name	Begin date	End date
Crabdle-CAD Design	6/18/19	8/6/19
Crabdle-Hardware design	8/7/19	8/21/19
Crabdle-Assembly & Connections	8/22/19	8/28/19
Crabdle-Low level control	8/29/19	9/17/19
Crabdle-High level control	9/18/19	10/7/19
Crabdle-Documentation	10/8/19	10/14/19
Memory-Latex template	10/15/19	10/19/19
Memory-Bibtex file	10/15/19	10/17/19
Memory-Introduction	10/20/19	10/24/19
Memory-State of the art	10/25/19	11/8/19
Memory-Used technologies	11/9/19	11/13/19
Memory-Bioinspired control system	11/14/19	11/23/19
Memory-Robotics implementations	11/24/19	12/13/19
Memory-Conclussions	12/14/19	12/18/19
Memory-Summary	12/19/19	12/23/19
Memory-WBS & Gantt	12/24/19	12/27/19
Memory-Social & Legal impact	12/28/19	12/29/19
Memory-Budget	12/30/19	12/31/19
Memory-Revisions	1/1/20	1/15/20
General cleaning & Extra documentation	1/16/20	1/30/20

Gantt Chart



Name	Begin date	End date
Learning Verrilog	9/1/18	9/20/18
Learning Open Source FPGA Tools	9/1/18	9/14/18
CPG State of the art	9/21/18	10/10/18
Bio-robots State of the art	9/21/18	10/15/18
Robots & FPGA State of the art	10/16/18	10/30/18
Nervous system State of the art	10/31/18	11/15/18
Doodle-Inspired models-Prototypes	11/16/18	11/22/18
Doodle-CAD Design	11/23/18	1/6/19
Doodle-Hardware architecture	11/23/18	12/2/18
Doodle-Assembly & Connections	1/7/19	1/13/19
Doodle-Low-level	1/14/19	2/27/19
Doodle-High-level	2/28/19	4/13/19
Doodle-Documentation	4/15/19	4/21/19
Randofo-CAD Adaptation	4/22/19	4/25/19
Randofo-Hardware architecture	4/26/19	4/29/19
Randofo-Assembly	4/30/19	4/30/19
Randofo-Low control	5/1/19	5/2/19
Randofo-Documentation	5/3/19	5/4/19
Mechatronic eye-Assembly & Connections	5/5/19	5/5/19
Mechatronic eye-Low level control	5/6/19	5/8/19
Mechatronic eye-Documentation	5/9/19	5/10/19
Zowihumanoid-CAD Design	5/11/19	5/20/19
Zowihumanoid-Assembly & Connections	5/21/19	5/22/19
Zowihumanoid-Low level locomotion	5/23/19	5/29/19
Zowihumanoid-Documentation	5/30/19	6/1/19
APIO-ROS-Python says ROS service	6/2/19	6/5/19
APIO-ROS-Initial tests	6/6/19	6/7/19
APIO-ROS-Tiago & Icezum Alhambra	6/8/19	6/10/19
Crabdle-Modular design tests	6/11/19	6/17/19
Crabdle-CAD Design	6/18/19	8/6/19
Crabdle-Hardware design	8/7/19	8/21/19
Crabdle-Assembly & Connections	8/22/19	8/28/19
Crabdle-Low level control	8/29/19	9/17/19
Crabdle-High level control	9/18/19	10/7/19
Crabdle-Documentation	10/8/19	10/14/19
Memory-Latex template	10/15/19	10/19/19
Memory-Bibtex file	10/15/19	10/17/19
Memory-Introduction	10/20/19	10/24/19
Memory-State of the art	10/25/19	11/8/19
Memory-Used technologies	11/9/19	11/13/19
Memory-Bioinspired control system	11/14/19	11/23/19
Memory-Robotics implementations	11/24/19	12/13/19
Memory-Conclussions	12/14/19	12/18/19
Memory-Summary	12/19/19	12/23/19
Memory-WBS & Gantt	12/24/19	12/27/19
Memory-Social & Legal impact	12/28/19	12/29/19
Memory-Budget	12/30/19	12/31/19
Memory-Revisions	1/1/20	1/15/20
General cleaning & Extra documentation	1/16/20	1/30/20

Budget						
Project/Component	Concept	Model	Nºparts/grams	Description	Unit prize (€)	Total (€)
Doodle	Open Source FPGA Board	Icezum Alhambra model I	1	Open Source FPGA- Arduino UNO factor	50	50
	Mini computer	Raspberry PI zero W	1	Mini computer	11	11
	Servomotor	SG90	3	Cheap servomotors for Doodle robot	1,5	4,5
	Servomotor replacements	SG90	3	Replacements due to different problems	2	6
	IR Sensors	BQ Zum IR sensors	2	BQ Zum IR sensors	5	10
	Doodle base	BQ Green PLA (grams)	18	Doodle 3D printed base	0,02	0,36
	Doodle front legs x2	BQ Red PLA (grams)	4	Doodle 3D printed front legs	0,02	0,08
	Doodle back legs x2	BQ Red PLA (grams)	4	Doodle 3D printed back legs	0,02	0,08
	Doodle central leg	BQ Red PLA (grams)	3	Doodle 3D printed central leg	0,02	0,06
	Tests and plastic replacements	BQ PLA (grams)	58	Iterative 3D printed tests and replacements	0,02	1,16
	Screws	M3 Screws	20	Used screws and replacements	0,025	0,5
	Nuts	M3 nuts	20	Used nuts and replacements	0,025	0,5
	Screws	M2 Screws	5	Used screws and replacements	0,03	0,15
	Nuts	M2 nuts	5	Used nuts and replacements	0,03	0,15
	Googly eyes	Googly eyes stickers	2	Googly eyes for a happy robot	0,01	0,02
	Lipo battery (charger included)	Lipo battery 1S	2	Lipo batteries for powering the robot	3,6	7,2
	Micro USB to Micro USB short cable	Generic	1	Cable to communicate the boards	4	4
	Cables	Jumper cables	10	Cables and replacements	0,03	0,3
Total						€ 96,06

Budget						
Project/Component	Concept	Model	Nºparts/grams	Description	Unit prize (€)	Total (€)
Randofo	Open Source FPGA Board	Icezum Alhambra model I	1	Open Source FPGA- Arduino UNO factor	50	50
	Servomotor	MG996R	2	Control the pair of front and back legs	4	8
	Pencils with eraser	Cheap pencils with eraser	2	Pencils for the back legs with an eraser for better grip	0,2	0,4
	Pencils without eraser	Cheap pencils without eraser	2	Pencils for the back legs without an eraser for slipping	0,2	0,4
	Randofo body	BQ yellow PLA (grams)	91	Robot body	0,02	1,82
	Randofo front legs	BQ Green PLA (grams)	18	Randofo front legs	0,02	0,36
	Randofo front gear	BQ Green PLA (grams)	6	Randofo front gear for the front legs mechanism	0,02	0,12
	Randofo back legs	BQ Green PLA (grams)	17	Randofo back legs	0,02	0,34
	Battery holder	4xAA battery holder	1	Battery holder for the power of the robot	1	1
	Battery	AA battery	4	Batteries to power the robot	0,35	1,4
	Googly eyes	Googly eyes stickers	2	Googly eyes for a happy robot	0,01	0,02
	Screws	M3 Screws	9	Used screws and replacements	0,025	0,225
	Nuts	M3 nuts	9	Used nuts and replacements	0,025	0,225

Total						€ 64,31

Budget						
Project/Component	Concept	Model	Nºparts/grams	Description	Unit prize (€)	Total (€)
Crabdle	Open Source FPGA Board	Icezum Alhambra model II	1	Open Source FPGA- Arduino UNO factor	50	50
	Mini computer	Raspberry PI 3 B+	1	Mini computer	30	30
	Servomotor	SG90	12	Cheap servomotors for Doodle robot	2	24
	Servomotor replacements	SG90	5	Replacements due to different problems	1,2	6
	Raspberry Pi Camera	Raspberry Pi Camera 2.1	1	Raspberry PI Camera for computer vision	30	30
	Custom PCB Manufacturing	Power PCB	1	Power PCBs with shipment from China	1,5	1,5
	UBEC	5 V/6 V BEC 3A-5A	1	Power converter for servomotors	1,46	1,46
	Buck converter	5V-2A	1	Power converter for the boards	2,5	2,5
	Crabdle base	BQ Green PLA (grams)	55	Crabdle 3D printed base	0,02	1,1
	Crabdle Raspberry holder	BQ Green PLA (grams)	10	Crabdle Raspberry Pi holder	0,02	0,2
	Crabdle FPGA holder	BQ Green PLA (grams)	11	Crabdle FPGA holder	0,02	0,22
	Crabdle Q1 Leg x6	BQ Red PLA (grams)	48	Crabdle Q1 Leg	0,02	0,96
	Crabdle Q2 Leg x6	BQ Red PLA (grams)	84	Crabdle Q2 Leg	0,02	1,68
	Tests and plastic replacements	BQ PLA (grams)	44	Iterative 3D printed tests and replacements	0,02	0,88
	Screws	M3 Screws	5	Used screws and replacements	0,025	0,125
	Nuts	M3 nuts	5	Used nuts and replacements	0,025	0,125
	Screws	M2 Screws	20	Used screws and replacements	0,03	0,6
	Nuts	M2 nuts	20	Used nuts and replacements	0,03	0,6
	Lipo battery (charger included)	Lipo battery 1S	4	Lipo batteries for powering the robot	3,6	14,4
	USB to Micro USB short cable	Generic	1	Micro USB to Micro USB shorth cable	3	3
	Cables	Jumper cables	25	Cables and replacements	0,03	0,75

Total						€ 170,10

Budget						
Project/Component	Concept	Model	Nºparts/grams	Description	Unit prize (€)	Total (€)
Zowihumanoid	Open Source FPGA Board	Icezum Alhambra model I	1	Open Source FPGA- Arduino UNO factor	50	50
	Servomotor	HK15138	4	Foot and leg servomotors	3,6	14,4
	Servomotor	SG90	5	Servomotors for arms and head	2	10
	Servomotor replacements	SG90	2	Replacements due to different problems	1,2	2,4
	Zowihumanoid chassis	BQ RED PLA (grams)	34	Chassis of the robot	0,02	0,68
	Zowihumanoid head	BQ RED PLA (grams)	45	Head of the robot	0,02	0,9
	Zowihumanoid foots	BQ RED PLA (grams)	44	Foots of the robot	0,02	0,88
	Zowihumanoid legs	BQ RED PLA (grams)	20	Legs of the robot	0,02	0,4
	Zowohumanoid shoulders	BQ RED PLA (grams)	20	Shoulders of the robot	0,02	0,4
	Zowohumanoid arms	BQ RED PLA (grams)	30	Arms of the robot	0,02	0,6
	Zowohumanoid body	BQ RED PLA (grams)	59	Body of the robot	0,02	1,18
	Battery holder	4xAA battery holder	1	Battery holder for the power of the robot	1	1
	Battery	AA battery	4	Batteries to power the robot	0,35	1,4
	Screws	M3 Screws	18	Used screws and replacements	0,025	0,45
	Nuts	M3 nuts	18	Used nuts and replacements	0,025	0,45
	Screws	M2 Screws	27	Used screws and replacements	0,03	0,81
	Nuts	M2 nuts	27	Used nuts and replacements	0,03	0,81

Total						€ 85,14

Budget						
Project/Component	Concept	Model	Nºparts/grams	Description	Unit prize (€)	Total (€)
Mechatronic eye	Open Source FPGA Board	Icezum Alhambra model I	1	Open Source FPGA- Arduino UNO factor	50	50
	Servomotor	SG90	4	Servomotors for controlling the eye	2	8
	Servomotor replacements	SG90	1	Replacements due to different problems	1.2	1.2
	Universal joint	3mm-3mm joint	1	Universal joint for the eye movement	1.25	1.25
	Clips	Clips	5	Clips as wired movement transmissions	0.01	0.05
	Eye base	BQ RED PLA (grams)	32	Base	0.02	0.64
	Eye eyelid bottom + top	BQ RED PLA (grams)	4	Eyelids mechanism	0.02	0.08
	Eye eyeball	BQ WHITE PLA (grams)	3	Eyeball	0.02	0.06
	Screws	M3 Screws	5	Used screws and replacements	0.025	0.125
	Nuts	M3 nuts	5	Used nuts and replacements	0.025	0.125
	Screws	M2 Screws	14	Used screws and replacements	0.03	0.42
	Nuts	M2 nuts	14	Used nuts and replacements	0.03	0.42

Total		€ 61.53

RRHH Professional hours budget				
	Concept	Approximate Nºhours	Hour price	Total (€)
Professional hours	Estimated hours cost for a Junior Robotics Engineer at Spain	550	30	€ 16,500.00

Total		€ 16,500.00

Budget resume		
Robot		Total (€)
Doodle		€ 96.06
Randofo		€ 64.31
Crabdle		€ 170.10
Zowihumanoid		€ 85.14
Mechatronic eye		€ 61.53

Total		€ 477.14

Budget resume taking into account common components between the robots		
Robot		Total (€)
Doodle	Icezum Alhambra I included in Doodle,	€ 96.06
Randofo	Icezum Alhambra I of Doodle used in Randofo, Zowihumanoid and mechatronic eye	€ 14.31
Crabdle	Icezum Alhambra II included in Crabdle	€ 170.10
Zowihumanoid	Icezum Alhambra I of Doodle used in Randofo, Zowihumanoid and mechatronic eye	€ 35.14
Mechatronic eye	Icezum Alhambra I of Doodle used in Randofo, Zowihumanoid and mechatronic eye	€ 11.53

Total		€ 327.14

Bibliography

[1] Darpa. **Darpa Robotics Challenge**, 2015. Available from: https://archive.darpa.mil/roboticschallenge/. 1

[2] European Robotics League. **ERL-European Robotics League**, 2019. Available from: https://eu-robotics.net/robotics_league/. 1

[3] European Robotics League. **Sciroc**, 2019. Available from: https://sciroc.eu. 1

[4] Boston Dynamics Inc. **Spotmini Robot**, 2019. Available from: https://www.bostondynamics.com/spot-mini. 2, 10

[5] Ecole Polytechnique federale du Lausanne. **Biorobotics laboratory Biorob**, 2019. Available from: https://biorob.epfl.ch. 3, 22

[6] Julián Caro Linares, Antonio Barrientos, and Enric Mayas Márquez. **Hybrid Bio-Inspired Architecture for Walking Robots Through Central Pattern Generators Using Open Source FPGAs**. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7071–7076. IEEE, 2018. 4, 144

[7] Boston Dynamics Inc. **Alphadog Robot**, 2019. Available from: https://spectrum.ieee.org/automaton/robotics/military-robots/ls3-alphadog-robot-begins-outdoor-assessment. 10

[8] Florida institute for human and machine cognition. **Ostrich like robot**, 2019. Available from: https://youtu.be/Wa0amJmJ550. 10

[9] Boston Dynamics Inc. **Atlas Robot**, 2019. Available from: https://www.bostondynamics.com/atlas. 11

[10] Paul Hebert, Max Bajracharya, Jeremy Ma, Nicolas Hudson, Alper Aydemir, Jason Reid, Charles Bergh, James Borders, Matthew Frost, Michael Hagman, et al. **Mobile manipulation and mobility as manipulation—design and algorithms of RoboSimian**. *Journal of Field Robotics*, **32**(2):255–274, 2015. 12

[11] Toyotaka Kozuki, Hirose Toshinori, Takuma Shirai, Shinske Nakashima, Yuki Asano, Yohei Kakiuchi, Kei Okada, and Masayuki Inaba. **Skeletal structure with artificial perspiration for cooling by latent heat for musculoskeletal humanoid kengoro**. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2135–2140. IEEE, 2016. 12

[12] Harvard University and Trinity College Dublin. **Soft robotics toolkit**, 2014. Available from: https://softroboticstoolkit.com. 12

[13] Michael T Tolley, Robert F Shepherd, Bobak Mosadegh, Kevin C Galloway, Michael Wehner, Michael Karpelson, Robert J Wood, and George M Whitesides. **A resilient, untethered soft robot**. *Soft robotics*, **1**(3):213–223, 2014. 12

[14] Michael Wehner, Ryan L Truby, Daniel J Fitzgerald, Bobak Mosadegh, George M Whitesides, Jennifer A Lewis, and Robert J Wood. **An integrated design and fabrication strategy for entirely soft, autonomous robots**. *Nature*, **536**(7617):451, 2016. 13

[15] Festo Inc. **Aquajelly**, 2019. Available from: https://www.festo.com/group/en/cms/10227.htm. 14

[16] Claudio Rossi, J Colorado, W Coral, and A Barrientos. **Bending continuous structures with SMAs: a novel robotic fish design**. *Bioinspiration & biomimetics*, **6**(4):045005, 2011. 14

[17] Festo Inc. **Flyingfox Festo**, 2019. Available from: https://www.festo.com/group/en/cms/13130.htm. 14

[18] Alireza Ramezani, Soon-Jo Chung, and Seth Hutchinson. **A biomimetic robotic platform to study flight specializations of bats**. *Science Robotics*, **2**(3):Art–No, 2017. 14

[19] Festo Inc. **BionicAnt**, 2019. Available from: https://www.festo.com/group/en/cms/10157.htm. 15

[20] Alexis Lussier Desbiens, Yufeng Chen, and Robert J Wood. **A wing characterization method for flapping-wing robotic insects**. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1367–1373. IEEE, 2013. 15, 21

[21] Joohyung Kim, Alexander Alspach, and Katsu Yamane. **Snapbot: a reconfigurable legged robot**. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5861–5867. IEEE, 2017. 16

[22] Carlos S Casarez and Ronald S Fearing. **Dynamic terrestrial self-righting with a minimal tail**. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 314–321. IEEE, 2017. 17

[23] Backyardbrains. **Wirelessly Control a Cyborg Cockroach**, 2019. Available from: https://backyardbrains.com/experiments/roboRoachSurgery. 18, 21

[24] H Sato, Y Peeri, E Baghoomian, CW Berry, and MM Maharbiz. **Radio-controlled cyborg beetles: a radio-frequency system for insect neural flight control**. In *2009 IEEE 22nd International Conference on Micro Electro Mechanical Systems*, pages 216–219. IEEE, 2009. 18

[25] Dae-Gun Kim, Serin Lee, Cheol-Hu Kim, Sungho Jo, and Phill-Seung Lee. **Parasitic robot system for waypoint navigation of turtle**. *Journal of Bionic Engineering*, **14**(2):327–335, 2017. 18

[26] Kazuyoshi Wada, Takanori Shibata, Takashi Asada, and Toshimitsu Musha. **Robot therapy for prevention of dementia at home**. *Journal of Robotics and Mechatronics*, **19**(6):691, 2007. 19

[27] *Encyclopedia of Life Sciences*. Wiley, 2005. Available from: https://books.google.es/books?id=YmrBvgEACAAJ. 19

[28] Eve Marder and Dirk Bucher. **Central pattern generators and the control of rhythmic movements**. *Current biology*, **11**(23):R986–R996, 2001. 20

[29] Nathan Griffith. **Arduino UNO-compatible robotic simulation of the C. elegans nematode**, 2017. Available from: https://github.com/nategri/nematoduino. 21

[30] Sébastien Joucla, Matthieu Ambroise, Timothée Levi, Thierry Lafon, Philippe Chauvet, Sylvain Saïghi, Yannick Bornat, Noëlle Lewis, Sylvie Renaud, and Blaise Yvert. **Generation of locomotor-like activity in the isolated rat spinal cord using intraspinal electrical microstimulation driven by a digital neuromorphic CPG**. *Frontiers in neuroscience*, **10**:67, 2016. 21

[31] Auke Jan Ijspeert. **Auke Jan Ijspeert, head of the Biorob group from Lausanne**, 2019. Available from: https://biorob.epfl.ch/people/people-ijspeert/. 22

[32] Sten Grillner, Peter Wallén, Nicholas Dale, Lennart Brodin, James Buchanan, and Russell Hill. **Transmitters, membrane properties and network circuitry in the control of locomotion in lamprey**. *Trends in Neurosciences*, **10**(1):34–41, 1987. 22

[33] Melanie Falgairolle, Jean-Charles Ceccato, Mathieu Sèze, Marc Herbin, and Jean-René Cazalets. **Metachronal propagation of motor activity**. *Frontiers in bioscience : a journal and virtual library*, **18**:820–37, 06 2013. 22

[34] Auke Jan Ijspeert, John Hallam, and David Willshaw. **Evolving swimming controllers for a simulated lamprey with inspiration from neurobiology**. *Adaptive behavior*, **7**(2):151–172, 1999. 23

[35] Auke Jan Ijspeert and Jérome Kodjabachian. **Evolution and development of a central pattern generator for the swimming of a lamprey**. *Artificial life*, **5**(3):247–269, 1999. 23

[36] Auke Jan Ijspeert and Alessandro Crespi. **Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model**. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 262–268. IEEE, 2007. 24

[37] 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**(5817):1416–1420, 2007. 25

[38] Konstantinos Karakasiliotis, Robin Thandiackal, Kamilo Melo, Tomislav Horvat, Navid K Mahabadi, Stanislav Tsitkov, Jean-Marie Cabelguen, and Auke J Ijspeert. **From cineradiography to biorobots: an approach for designing robots to emulate and study animal locomotion**. *Journal of The Royal Society Interface*, **13**(119):20151089, 2016. 27, 28

[39] Biorob department of University of Lausanne. **Online Orobot simulation made in Webot.**, 2019. Available from: https://cyberbotics2.cyberbotics.com/orobot/simulation.php. 29

[40] John A Nyakatura, Kamilo Melo, Tomislav Horvat, Kostas Karakasiliotis, Vivian R Allen, Amir Andikfar, Emanuel Andrada, Patrick Arnold, Jonas Lauströer, John R Hutchinson, et al. **Reverse-engineering the locomotion of a stem amniote**. *Nature*, **565**(7739):351, 2019. 29

[41] Juan González Gómez, Houxiang Zhang, Eduardo I Boemo, and Jianwei Zhang. **Locomotion capabilities of a modular robot with eight pitch-yaw-connecting modules**. 2006. 30

[42] Alexander S Boxerbaum, Andrew D Horchler, Kendrick M Shaw, Hillel J Chiel, and Roger D Quinn. **Worms, waves and robots**. In *2012 IEEE International Conference on Robotics and Automation*, pages 3537–3538. IEEE, 2012. 30

[43] David Zarrouk, Moshe Mann, Nir Degani, Tal Yehuda, Nissan Jarbi, and Amotz Hess. **Single actuator wave-like robot (SAW): design, modeling, and experiments**. *Bioinspiration & biomimetics*, **11**(4):046004, 2016. 30

[44] Alexander Spröwitz, Alexandre Tuleu, Massimo Vespignani, Mostafa Ajallooeian, Emilie Badri, and Auke Jan Ijspeert. **Towards dynamic trot gait locomotion: Design, control, and experiments with Cheetah-cub, a compliant quadruped robot**. *The International Journal of Robotics Research*, **32**(8):932–950, 2013. 31

[45] Javier Isabel. *Locomoción basada en osciladores para robots con patas*. Master's thesis, Universidad Carlos III de Madrid, 2016. 32, 115

[46] Martin J Pearson, Anthony G Pipe, Chris Melhuish, Ben Mitchinson, and Tony J Prescott. **Whiskerbot: a robotic active touch system modeled on the rat whisker sensory system**. *Adaptive Behavior*, **15**(3):223–240, 2007. 33

[47] Juan Gonzalez-Gomez, E Aguayo, and E Boemo. **Locomotion of a Modular Worm-like Robot using a FPGA-based embedded MicroBlaze Soft-processor**. In *Climbing and walking robots*, pages 869–878. Springer, 2005. 34

[48] David Duff, Mark Yim, and Kimon Roufas. **Evolution of polybot: A modular reconfigurable robot**. In *Proc. of the Harmonic Drive Intl. Symposium, Nagano, Japan*, 2001. 34

[49] Alex Golovinsky, Mark Yim, Ying Zhang, Craig Eldershaw, and David Duff. **Polybot and polykinetic/spl trade/system: a modular robotic platform for education**. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, **2**, pages 1381–1386. IEEE, 2004. 34

[50] Erick Israel Guerra-Hernandez, Andres Espinal, Patricia Batres-Mendoza, Carlos Hugo Garcia-Capulin, Rene De J Romero-Troncoso, and Horacio Rostro-Gonzalez. **A FPGA-based neuromorphic locomotion system for multi-legged robots**. *IEEE Access*, **5**:8301–8312, 2017. 35

[51] Derek Chiou. **The microsoft catapult project**. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pages 124–124. IEEE, 2017. 36

[52] Clément Farabet, Cyril Poulet, Jefferson Y Han, and Yann LeCun. **Cnp: An fpga-based processor for convolutional networks**. In *2009 International Conference on Field Programmable Logic and Applications*, pages 32–37. IEEE, 2009. 37

[53] Arduino Inc. **Arduino**, 2005. Available from: https://www.arduino.cc. 42

[54] Espressif. **ESP32 board**, 2017. Available from: https://www.espressif.com/en/products/hardware/esp32/overview. 42

[55] Raspberry Pi Foundation. **Raspberry Pi Foundation**, 2012. Available from: https://www.raspberrypi.org. 43

[56] Aldebaran-Softbank. **Naoqi Framework**, 2019. Available from: http://doc.aldebaran.com/1-14/dev/naoqi/index.html. 45

[57] ros.org. **ROS: Powering the world's robots**, 2019. Available from: http://www.ros.org. 45

[58] OpenCV dev team. **Python OpenCV**, 2014. Available from: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_setup/py_intro/py_intro.html#intro. 46

[59] The FreeCAD Team. **Freecad: Your own 3D parametric modeler**, 2014. Available from: https://www.freecadweb.org. 48

[60] Autodesk. **Autodesk Inventor**, 2014. Available from: https://www.autodesk.com/products/inventor/overview. 48

[61] RepRap Community. **RepRap**, 2007. Available from: https://www.reprap.org/wiki/RepRap. 48

[62] Kicad Developers tema. **Kicad EDA**, 1992. Available from: http://kicad-pcb.org. 49

[63] Microsoft. **Visual Studio Code**, 2019. Available from: https://code.visualstudio.com. 50

[64] Sublime HQ Pty Ltd. **Sublime text IDE**, 2019. Available from: https://www.sublimetext.com. 50

[65] Github Inc. **Atom IDE**, 2019. Available from: https://atom.io. 50

[66] PlatformIO. **Plafformio**, 2014. Available from: https://platformio.org. 51

[67] Alec AK Nielsen, Bryan S Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A Strychalski, David Ross, Douglas Densmore, and Christopher A Voigt. **Genetic circuit design automation**. *Science*, **352**(6281):aac7341, 2016. 56

[68] C Wolf. **Claire Wolf personal webpage**, 2010. Available from: http://www.clifford.at. 58

[69] Lattice. **Icestick**, 2013. Available from: https://www.latticesemi.com/icestick. 58

[70] C Wolf and M Lasser. **Project icestorm**, 2015. Available from: http://www.clifford.at/icestorm/. 58

[71] FPGAWars Community. **FPGAWars-Explorando el lado libre de las FPGAs**, 2019. Available from: http://fpgawars.github.io. 60, 88

[72] Clone wars Community. **Clone Wars- A Reprap community**, 2019. Available from: https://www.reprap.org/wiki/Proyecto_Clone_Wars. 60

[73] Daniel C Wolf and Eduardo. **icoBoard: An FPGA based IO board for RaspberryPi**, 2016. Available from: http://icoboard.org. 60

[74] 1 Bit Squared. **iCEBreaker: The first open source iCE40 FPGA development board designed for teachers and students**, 2019. Available from: https://www.crowdsupply.com/1bitsquared/icebreaker-fpga. 60

[75] Luke Valenty. **TinyFPGA**, 2019. Available from: https://tinyfpga.com. 60

[76] Sutajio Kosagi. **Fomu: An FPGA board that fits inside your USB port**, 2019. Available from: https://github.com/im-tomu/fomu-hardware. 60

[77] Eladio Delgado Mingorance and Juan González Gómez. **IceZUM Alhambra: an Arduino-like Open FPGA electronic board**, 2018. Available from: https://github.com/fpgawars/icezum/wiki. 61, 95

[78] Jesús Arroyo and Juan Gonzalez Gomez. **Apio IDE: Developing hardware for open FPGAs easily**, 2019. Available from: https://github.com/fpgawars/apio-ide. 63

[79] Jesús Arroyo Torrents and Juan González Gómez. **Icestudio: A real gamechanger in the world of Open Source FPGAs**, 2019. Available from: https://icestudio.io. 65

[80] Luba Elliott. **AI Art Gallery**, 2018. Available from: http://www.aiartonline.com. 72

[81] Julián Caro Linares. **FPGA-BIOROBOTS-Hybrid bio-inspired architecture for robots using open source FPGAs.**, 2019. Available from: https://github.com/jcarolinares/fpga-biorobots/. 88, 143

[82] Julián Caro Linares. **Doodle the robot**, 2019. Available from: https://github.com/jcarolinares/fpga-biorobots/tree/master/robots/doodle-robot. 90, 96

[83] Julián Caro Linares. **Doodle robot using Oscillators in an open source FPGA**, 2019. Available from: https://youtu.be/HQzxtiUjpWk. 93

[84] Lattice Inc. **ICE40 LP/HX/LM**, 2019. Available from: https://www.latticesemi.com/Products/FPGAandCPLD/iCE40. 95

[85] Juan Gonzalez Gomez C Wolf. **RISC-V: A picorv32 inside an Icezum Alhambra**, 2019. Available from: https://github.com/FPGAwars/Alhambra-II-FPGA/tree/master/examples/picorv32/picosoc. 95, 125

[86] Julián Caro Linares. **Taller de Robótica y FPGAs libres: Construyendo a Doodle**, 2018. Available from: https://oshwdem.org/taller-de-robotica-y-fpgas-libres-construyendo-a-doodle/. 96, 144

[87] Carlos García Saura. **Printed Micro-Hexapod**, 2019. Available from: https://www.thingiverse.com/thing:34796. 97

[88] Julián Caro Linares. **Oscillator ROM generator script**, 2019. Available from: https://github.com/jcarolinares/fpga-biorobots/blob/master/python_says/oscillator_rom_generator.py. 105, 135, 141

[89] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986. 107

[90] Julián Caro Linares. **Python says - A high level generator of Verilog circuits for open source FPGAs**, 2019. Available from: https://github.com/jcarolinares/fpga-biorobots/tree/master/python_says. 109, 141

[91] Erik Max Francis. **Empy-A powerful and robust templating system for Python.**, 2019. Available from: http://www.alcyone.com/software/empy/. 110, 138

[92] Julián Caro Linares Randy Safaran. **Randofo-Arduino UNO version**, 2019. Available from: https://github.com/jcarolinares/fpga-biorobots/tree/master/robots/randofo_robot_futaba_arduino_version. 111

[93] Randy Safaran. **Randofo: 3D Printed Robot**, 2017. Available from: https://www.instructables.com/id/3D-Printed-Robot/. 111

[94] Julián Caro Linares. **Randofo video. A simple robot controlled with an Open Source FPGA**, 2017. Available from: https://youtu.be/ET-EtgHS6wI. 112

[95] Will Cogley. **3D printed animatronic eye mechanism**, 2019. Available from: http://www.instructables.com/id/3D-Printed-Animatronic-Eye-Mechanism-on-the-Cheap/. 113

[96] Asrob UC3M. **Zowi Humanoids Mods**, 2017. Available from: https://github.com/asrob-uc3m/zowi/tree/master/mods/Zowi_Maker_faire_robot. 115

[97] Javier Isabel Hernandez and Julián Caro Linares. **Zowi Maker Faire Robot**, 2019. Available from: https://github.com/asrob-uc3m/zowi/tree/master/mods/Zowi_Maker_faire_robot. 115

[98] Julián Caro Linares. **ROS Apio Module: A ROS module to verify, synthesize and upload circuits to Open Source FPGAs**, 2019. Available from: https://github.com/jcarolinares/ros-apio. 116

[99] Julián Caro Linares. **Crabdle: An Open-Source robot with Brain and Nervous system using Open-Source FPGAs**, 2019. Available from: https://github.com/jcarolinares/fpga-biorobots/tree/master/robots/crabdle-robot. 117

[100] Julián Caro Linares. **Servomotors external power adaptor board**, 2019. Available from: https://github.com/jcarolinares/servomotors_ext_power_adaptor. 123

[101] Julián Caro Linares. **Crabdle : Development of its first locomotion**, 2019. Available from: https://youtu.be/f76K9b2BC2k. 136

[102] Julián Caro Linares. **Crabdle: A Bio-Inspired Hexapod Robot that uses Open-Source FPGAs**, 2019. Available from: https://youtu.be/_nxg-Jf_gdw. 136

[103] Julián Caro Linares. **Biorobótica y FPGAs libres. Sintetizando el futuro.**, 2017. Available from: https://madrid.makerfaire.com/makers-exhibits/. 143

[104] Julián Caro Linares. **Robots bioinspirados y FPGAs Libres**, 2017. Available from: https://oshwdem.org/makers-oshwdem-2017/. 144

[105] Julián Caro Linares. **Robots, gatitos y FPGAs libres.**, 2018. Available from: https://arduinodayzgz.es/evento_2018.html. 144

[106] Julián Caro Linares. **Robots, gatiños e fpgas libres**, 2018. Available from: https://galicia.makerfaire.com/2018/10/17/la-bio-robotica-de-julian-linares-en-la-maker-faire-galicia-2018/. 144

[107] Julián Caro Linares. **¡Monta tu propia máquina Enigma en una FPGA Libre!**, 2019. Available from: https://t3chfest.uc3m.es/2019/learn-with-t3chfest/monta-propia-maquina-enigma-una-fpga-libre/. 144

[108] Julián Caro Linares. **Entrevista a Julián Caro Linares: Robótica y FPGAs libres**, 2019. Available from: https://programarfacil.com/podcast/robotica-fpgas-libres/. 144

[109] Julián Caro Linares. **Hablamos con Julián Caro. FPGAs en biorobótica y semáforos**, 2018. Available from: https://youtu.be/4iaPBEIK1U0. 144