ENSTA Bretagne
2, rue François Verny
29806 BREST cedex
FRANCE
Tel +33 (0)2 98 34 88 00
www.ensta-bretagne.fr

Report

ROB 2024

September 30, 2023

Engineering Assistant Internship

RITMO Centre for Interdisciplinary Studies
in Rhythm, Time and Motion
University of Oslo

Supervisor: Alexander SZORKOVSKY

Optimization of sensory inputs for synchronizing robots

Marguerite MIALLIER
Engineering Student in Autonomous Robotics
marguerite.miallier@ensta-bretagne.org

Contents

Résumé

Dans le cadre de ma formation d'ingénieur, j'ai effectué un stage de recherche de 16 semaines à RITMO, centre de recherche dépendant de l'Université d'Oslo. Supervisée par Dr. Alexander Szorkovosky, j'ai travaillé avec l'équipe du projet COROBOREES sur un hexapode autonome. Ce robot, contrôlé par un CPG (Central Pattern Generator) stimulé par une entrée audio, était fonctionnel en simulation. Le principal objectif de ce stage était de rendre le vrai robot fonctionnel.

Durant les trois premiers mois du stage, nous avons assemblé toutes les pièces, remanié certaines parties du robot pour rendre le rendre plus facile d'utilisation. Dans le même temps, j'ai implémenté à l'aide de ROS2 les codes permettant l'acquisition des données de la centrale inertielle, des capteurs de pression des pieds du robot, et du microphone externe.

Pendant le dernier mois, j'ai travaillé sur la comparaison des performances des différents CPG entre la simulation et la réalité, ainsi que sur l'évaluation des CPG dans leur capacité à adapter leur comportement au tempo de l'entrée audio.

Abstract

As part of my engineering training, I did a 16-weeks research internship at RITMO, at the University of Oslo. I worked on an autonomous hexapod with the COROBOREES team, under the supervision of Dr. Alexander Szorkovsky. This robot is controlled by a CPG (Central Pattern Generator) triggered by an audio input. At the beginning of the internship, the simulation of the robot was working, and my main goal was to make the real version of the robot work.

During the first three months, we assembled all the parts, redesigned some of them to make the robot easier to use. At the same time, I implemented the programs to get the data from the IMU, the foot contact sensors and the microphone with ROS2.

During the last month, I worked on the comparison of the performances of the robot between the simulation and the reality, and on the evaluation of the CPGs in their ability to adapt their behavior to the tempo of the audio input.

Keywords

Entrainment, Sensory input, ROS 2, Evolutionary Robotics, Hexapod, Central Pattern Generator

Thanks

First, I would like to thank Alexander Szorkovsky and Kyrre Glette for allowing me to participate in such an interesting project, and for all their help and support during this internship.

I would also like to thank the RITMO and ROBIN teams for their warm welcome and all the varied and enriching discussions we were able to have.

Then, thanks to the Senior Engineer Kayla Burnim for her precious help, without whom we could not have used the MoCap Lab.

Finally, I would like to thank Timothé Rivier, who has been a great working partner during this project, and Gauvain Thomas for his knowledge in mathematics and programming and for helping me to clarify all my ideas.

1 Introduction

1.1 RITMO and IFI

During this internship, I was affiliated to RITMO Centre for Interdisciplinary Studies in Rhythm, Time and Motion, which is a research center that regroups international researchers working on psychology, informatics, robotics, musicology and philosophy. The lab is divided in four main clusters : Interaction and pleasure, Interaction and robotics (of which I was a part), Structure and aesthetics, and Structure and cognition. This wide variety of research subjects was enriching and made it possible to meet people from all walks of life. The lab organized several events like conferences, meetings, but also hikes and activities that allowed me to meet everyone during the first weeks of the internship.

I also spent half of my time at IFI, the Department of Informatics at the University of Oslo, with the ROBIN (Robotics and Intelligent systems) research team. There we had easy access to the tools, materials and machines to build the robot, like soldering tools or CNC machines. Both at IFI and RITMO, we could have the precious help and advice of the lab engineers.

1.2 The COROBOREES project

Context

During this internship, I worked with the "Synchronized Robotics" team on the project "COllective ROBOtics through Real-time Entrainment of Evolved dynamical Systems" (COROBOREES), which uses evolutionary methods to create complex non-linear dynamical systems that allow flexibility while retaining stability. These are used as Central Pattern Generators (CPGs) with tunable social responsiveness in legged robots. The project seeks to identify :

- Minimum requirements for rhythmic entrainment
- How rhythmic complexity affects entrainment
- Conditions for synchronization between multiple agents

Evolutionary robotics is an embodied approach to AI where robots are automatically designed following the rules and principles of natural selection. The controller is optimized according to a desired behavior (e.g : be as stable as possible). During the evolution part, thousands of controllers are created and tested in simulation. As the algorithms are often probabilistic, the controllers that have a lower fitness are more likely to be deleted and those with a higher fitness are more like to reproduce (with mutation and/or combination with others). Filters are also evolved to process the sound. These CPGs and filters are described by arrays of integers, and they all can be used on the same physical robot by changing the corresponding arrays in the program parameters.

Entrainment is a term used to describe the adjustment and alignment of periodic signals in domains including sensorimotor and interpersonal coordination, oscillatory neural activity, and robotic interactions.

The theme of entrainment is a subject that is studied in depth at RITMO, as it allows a better understanding of the various processes that come with it. Entrainment is often presented from a physiological, sociological or philosophical point of view, and this project brings another perspective by combining robotics and bio-inspired neural networks.

The project began in 2021, and will end by the end of 2023. When I arrived in May 2023, almost all the objectives had been achieved in simulation with four- and six-legged robots, so my job was to make it work for real.

Objectives

The main challenge of my work was to make the physical robot operational, so that we could :

- evaluate the gap between the simulation and the reality
- interact with the robot (e.g by clapping hands)
- present a demo of the robot during the Entrainment Workshop (RITMO, August 17 & 18)

To achieve this goal I would have to :

- build the robot
- get the data from the different sensors
- create communication between the sensors and the controller

1.3 Stakes

This internship presented various stakes, both for me and for the COROBOREES team. For me this internship had four main stakes :

1. A human and cultural stake: This was the first time I had the opportunity to live in a foreign country. Almost everyone in Norway speak English, so I saw how much progress I had to make in this language to speak as well as them. A lot of things are different between Norway and France, as the currency or the food, and I had to adapt my habits to better integrate.
2. A financial stake: Norwegian is a very expensive country, so this internship was a real challenge for me and my family. Finding an accommodation under 600 euros per month is difficult, and the food is also expansive. I was fortunate to receive funds from ERASMUS+, but also from the University of Oslo, which is not always the case in foreign universities.

3. A professional stake: This internship was my first experience in a research group, so I had to adapt my working habits to this new context.

4. A technical stake: During this internship, I was able to get more familiar with ROS2, have a first experience with motion capture, and discover the challenges coming with the communication between hardware and software.

There was also a technical stake for the project team. Indeed, the majority of them didn't have experience with ROS2, and with this knowledge I was able to be effective in helping them carry out this project. As the project was supposed to end at the end of August, it was very important for them to have someone who would work on the robot.

2 Bringing the robot to life

2.1 Working dynamics

We were two french interns working on the COROBOREES project: Timothé, who worked on how to improve the simulation to make it closer to the real robot, and I. During the first two months of the internship, we had weekly meetings with our two supervisors. The goal of these meetings was to recap all that we had done since the previous meeting. Each time we prepared a few slides that presented the work done, the work in progress, the problems encountered, what we were going to do following the meeting and our possible questions. These meetings were very important for everyone. It allowed our supervisors to follow our progress and give us advice if needed, and that way we could make sure that we were going in the right direction. Between the meetings, we also could send them emails or messages through Mattermost (an open source platform for team collaboration) if we had any questions or problems.

During the last two months, we had only a few meetings, because Alexander and Kyrre went in vacation and to some conferences. However, we still could reach them through Mattermost, which was very helpful because I faced some issues with my code during this period.

2.2 Building the robot

The robot we used for the project is an hexapod for which the design is open source. Each leg is made of three servomotors, carbon rods, 3D printed parts and a silicone foot. All the legs are held together by a body made of two polycarbonate plates. We needed to adapt some parts of the robot to make it easier to use.

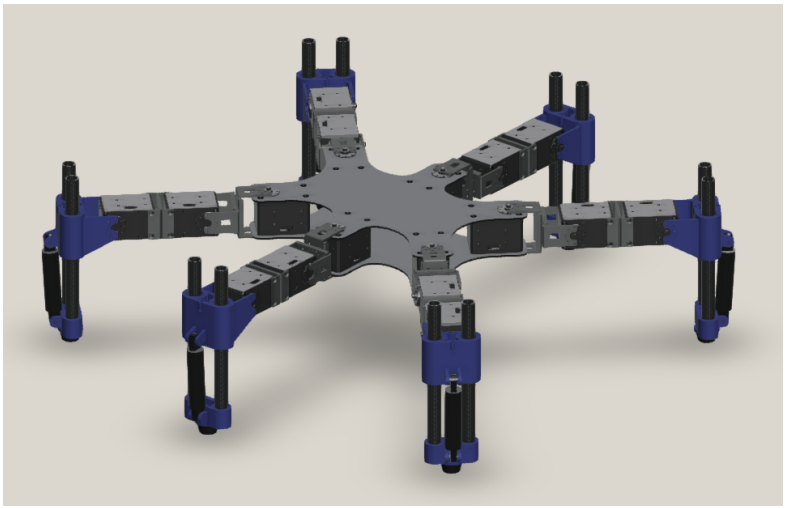
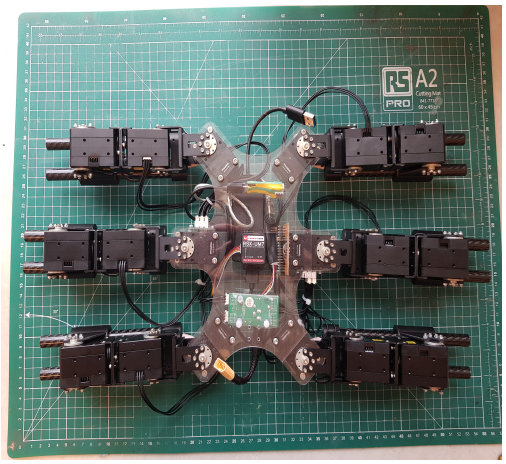


Figure 1: Open source design of the robot (https://github.com/adaptive-intelligent-robotics/Hexapod_Design_V2)

Mechanical part

When my internship started, the robot was in pieces. I participated in assembling the legs and attaching them to the body. It was a very long work because there are more than 200 screws on the robot. It took several days to have the robot completely built.

(a) Legs and body of the robot



(b) The final robot

Figure 2: Robot building steps

Electronic part

The robot contains :

- 18 servomotors (Dynamixel XM430-W350) powered by a power board (openCM 485 EXP)
- U2D2 USB communication converter to communicate with the servos
- 1 IMU (UM7)
- 6 FSR sensors
- 1 USB3 hub
- 1 Arduino Micro

A microphone RODE-NT USB is also plugged directly in the computer.

Figure 3: RODE NT-USB microphone (source: RODE website)

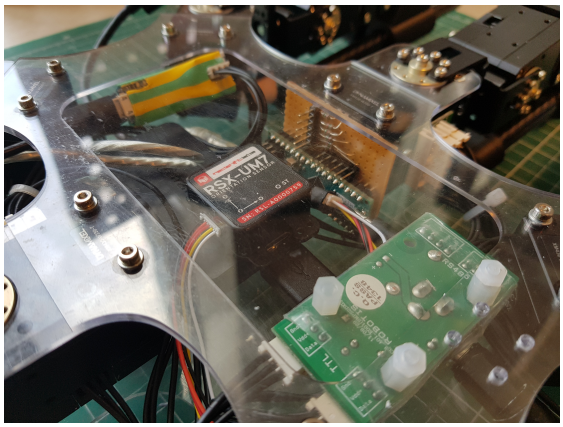


Figure 4: Robot electronics

The part concerning the servomotors was already working, and I added the sensors part. I soldered the FSR sensors to the Arduino board and the first microphone we were supposed to use. When all the parts were soldered and ready to use, we put everything in the robot between the two plates. The robot is wired and can move on the ground, so we needed a 5m long USB3 cable. As it is a bit expensive, we reused one that was on a robot built for a previous project. The robot is powered by a 12V 12.5A DELL power adapter.

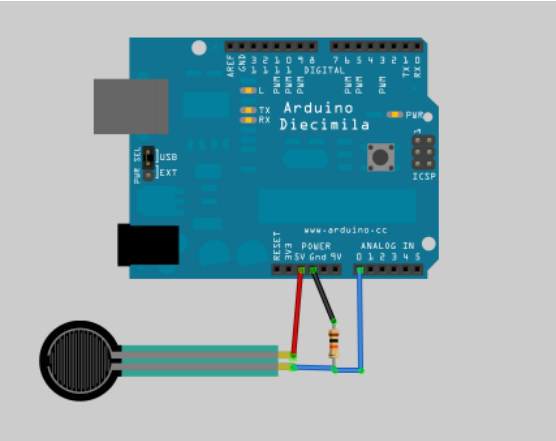


Figure 5: Connection of a FSR sensor to an Arduino board (source: Adafruit)

2.3 Obtaining sensor data

As mentioned before, the robot uses three main types of sensors :

1. An **IMU**, from which the front tilt and side tilt are obtained. It allows the robot to stabilize.
2. A **microphone**, that allows to get the amplitude of the sound to which the robot will react.
3. Six **FSR sensors**, that measure the force applied by the ground on each foot. It will be used later as an input by the controller.

The figure 6 illustrates the global architecture of the system, whose parts will be explained in this section.

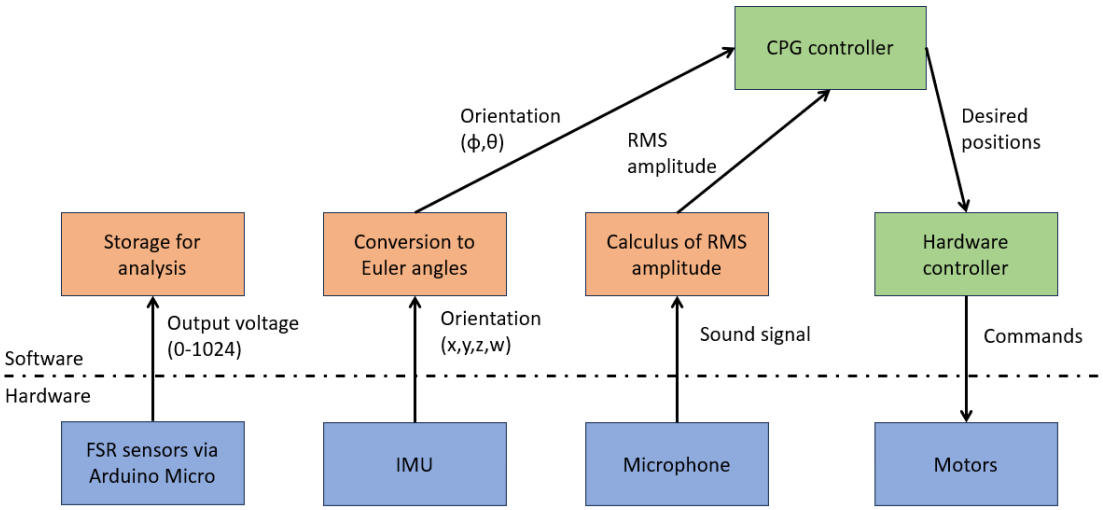


Figure 6: Architecture of the system

ROS 2 general architecture

In this project, **ROS 2** is used to establish communication between the different programs used to control the robot.

According to the *ROS 2 documentation*: "The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project."

The hardware interface part, as well as the base of the CPG part was done by Lars Kristian Bårdevik, research assistant working with the team until early June.

For this project, we use one package and one node per part of the robot : one for each type of sensor, one for the controller and one for the motor commands. Each node publishes on one topic. What was very important was to be able to test each part of the robot separately. The sensors nodes can be run independently of others, and if only one sensor is plugged in, getting the data is possible without changing any part of the code. This was very important because it allowed me to work on the sensors when the robot wasn't even built, and it also made debugging easier.

All the nodes are intended to run on a 15ms loop as in simulations.

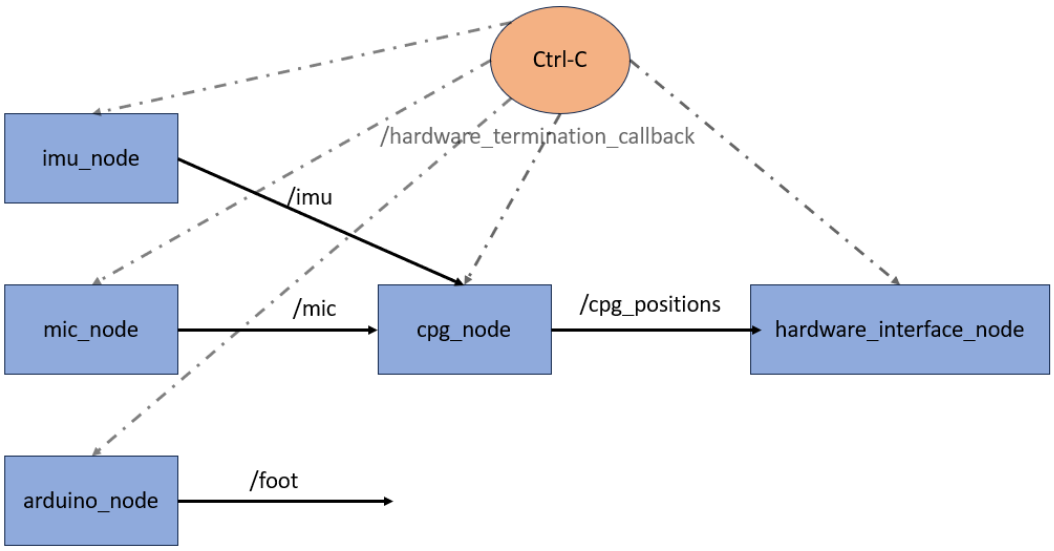


Figure 7: ROS 2 node graph

Now that we provided an overview of the structure of the system, we can focus more on the packages I created or modified.

IMU module

The main sensor the robot needs is the IMU. Here we use a **UM7 IMU**. I couldn't find the official drivers for this model, so I used a driver made by Isaac Jesus Da Silva, implemented in **C++**. I had to modify it to make it work with ROS2. To do that, I created the class `ImuNode`, and added to it all the functions that were in the original driver. I then adapted the main loop to transform it into a ROS 2 callback. My first trials didn't work, because with just reproducing this loop into the callback, the sensor would just be reconfigured each time the function is called, which takes a lot of time. To solve this problem, I just created a new boolean variable `configured`, that is set to `true` when the sensor is configured, and to `false` if an exception occurs and the port has to be closed and reopened.

Data collected

This node collects the data measured by the IMU :

```
auto message = sensor_msgs::msg::Imu();
message.orientation.x = IMU_QUAT_X;
message.orientation.y = IMU_QUAT_Y;
message.orientation.z = IMU_QUAT_Z;
message.orientation.w = IMU_QUAT_W;
message.angular_velocity.x = IMU_GYRO_X;
message.angular_velocity.y = IMU_GYRO_Y;
message.angular_velocity.z = IMU_GYRO_Z;
message.linear_acceleration.x = IMU_ACCEL_X;
message.linear_acceleration.y = IMU_ACCEL_Y;
message.linear_acceleration.z = IMU_ACCEL_Z;
```

As the IMU already contains a Kalman filter, there is no need to filter this data.

The orientation is given in the form of a quaternion [2], i.e. a number of the form

$$w + xi + yj + zk$$

with w, x, y, z real numbers and

$$i^2 = j^2 = k^2 = -1$$

.

Data published

The data is stored in a `sensor_msgs/msg/Imu` message, described in the ROS 2 documentation. They are then published on the `/imu` topic.

Data we're interested in

The robot only needs the **front tilt** and **side tilt** to work, so to calculate them, we just need the four quaternion coordinates x,y,z and w. They need to be converted in Euler angles [3], that describe the rotation of a rigid body with respect to a fixed coordinate system, as needed by the controller. It will be done in the CPG node, as explained in CPG module.

Arduino module

This module is used to get the data from the **FSR sensors**. It was also supposed to get the data from an Arduino microphone, but this one was not designed for long range audio detecting, so we switched to a USB microphone.

The node uses the library `serial` to collect data from the serial port (here `/dev/ttyACM0`).

Data is obtained as lines of `'bytes'` in the `timer_callback` function, then we split them in the `data_processing` function, which returns six `int` corresponding to each foot sensor.

I created a custom message to publish this data, defined in the file `Foot.msg`. It is made of six `Int32`.

Each sensor value ranges from 0 to 1024 and increases with the force applied to the sensor, corresponding to a voltage output from 0 to 5V.

The data is published on the `/foot` topic as a `Foot` message.

Arduino code

As we just need to get data from the six FSR sensors, the code in the Arduino board is very simple. Each sensor is connected to one analog pin, so they are read one by one. The data is sent to the serial port in the following format :

```
data = String(analogRead(foot0)) + " "
+ String(analogRead(foot1)) + " " + String(analogRead(foot2)) + " "
+ String(analogRead(foot3)) + " " + String(analogRead(foot4)) + " "
+ String(analogRead(foot5));
```

where `footX` is the name of the pin corresponding to the foot X.

Microphone module

Our robot uses a microphone input that is sent to the CPG, so that it can adapt his gait to the audio input. Here we use a Rode NT-USB microphone. To get the audio input, we use the PyAudio library.

After inconclusive tests with an Electret Microphone with the Arduino Micro (see Testing the robot) , we discussed the pros and cons and decided to use an USB microphone. Several options were considered to record data from the microphone, as trying to find an equivalent of the `audio_common` ROS package for ROS 2. The `PyAudio` library appeared to be the most practical solution, as it is relatively well documented and easy to use.

The node opens an audio stream using some parameters, whose :

- the **audio device**, selected by name in the list of the available devices
- the **sample rate** inherent to the device, here 44.1 kHz
- the **chunk**, which is the number of frames read each time the callback is called

Here, the chunk is set to 685, corresponding to around 15 ms, to fit the period of the callback.

We get the data as `bytes` that we convert to `int16`. Then we have a list of `chunk` amplitude values.

For our use, we need the robot to be able to hear sound coming for everywhere in a room, and not to be too sensitive to very loud sounds. To allow this, I implemented a **dynamic range compressor**. This is an audio signal processing operation that amplifies low sounds or reduces loud ones. In our case, the compressor does both. It takes an array of amplitude values and convert them to decibels :

$$x_{dB} = 20 \log |x|$$

where x is the input data and x_db the data converted in decibels.

The decibel values under a minimum threshold are set higher using :

$$x_{compressed} = k(x_{dB} - X_{min}) + X_{min}$$

where k is the compression factor, and X_min the minimum threshold.

The same formula is used for values upper a maximum threshold, changing X_{min} to X_{max}.

When the decibel values are modified, they are converted back to amplitude using

$$x = 10^{\frac{x_{compressed}}{20}}$$

The output of the compressor is then multiplied by a gain set to 0.01, because otherwise the amplitude values are too high for the controller - due to the training and evolution part - and the legs of the robot get stuck in high position.

After the compression is calculated the RMS (Root Mean Square) amplitude of the values :

$$x_{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

where n is the length of the data array x.

This RMS amplitude is what will be published, as an `std_msgs/msg/Int64` message, on the `/mic` topic.

CPG module

The CPG module contains all the functions necessary to control the robot. The CPG node uses a brain and a filter defined as lists of integers, specific to each CPG, and calculates the leg commands that will be sent to the hardware interface node, which will convert them to commands that can be given to the motors. The control part of the CPG node was already implemented, I only added the subscribers necessary to receive the data from the different sensors :

- `imu_callback`, that gets the message from `/imu`, uses a function to convert the quaternions to Euler angles using the following formula :

  Let x, y, z, w be the coordinates of the quaternion representing the orientation of the IMU. Then,

  $$t_0 = 2(wx + yz)$$
  $$t_1 = 1 - 2(x^2 + y^2)$$

  $$t_2 = \begin{cases} 1 & \text{when } 2(wy - zy) > 1 \\ -1 & \text{when } 2(wy - zy) < -1 \\ 2(wy - zy) & \text{otherwise} \end{cases}$$

  $$t_3 = 2(wz + xy)$$
  $$t_4 = 1 - 2(y^2 + z^2)$$

  $$\theta = \arcsin(t_2)$$
  $$\varphi = arctan2(t_0, t_1)$$
  $$\psi = arctan2(t_3, t_4)$$

  φ and ψ are then used as `sidetilt` and `fronttilt` by the control functions to stabilize the robot.

- `mic_callback` that gets the RMS amplitude published on `/mic`. The amplitude is also used by the same control functions to calculate the tempo of the music, so that the CPG can adapt his own period to the tempo.

Termination of the experiment

As the experiments involve a real robot, and we don't know exactly the duration we will need for each of them, we had to find a way to properly shut down the programs. Indeed, if we would just press `Ctrl-C`, the robot would stop, but in a position that could damage the motors, and we would lose all the data we would want to analyze.

To solve that problem, we implemented a `bash` function that publishes the message `'terminate'` on a topic when the user presses `Ctrl-C`. Each node mentioned before subscribes to this topic and launches the termination process when `'terminate'` is received :

- in the `hardware_interface_node`, all the positions of the motors are set back to their original position, so that the robot can stand.
- in the `cpg_node`, the CPG positions (commands sent to the hardware interface node), the front tilt and side tilt of the robot, previously stored in different arrays, are saved as `.npy` files.
- the same is done for the data collected in the `arduino_node`
- in the `microphone_node`, all the bytes frames stored in an array are written in a `.wav` file.

The data contained in the `.npy` files will be used for numerical analysis and graphical visualization. The `.wav` files can be read with any audio playback software, which allows us to replace every experiment in the audio context the robot was in.

2.4 Testing the robot

Each phase of the programming process needed some tests to check if what was implemented worked well.

Testing the IMU

To see if the IMU node worked, I ran it and printed in the terminal the Euler angles calculated as explained before. Moving around the IMU, I checked with a compass and an angle protractor that the angles measured were accurate. When that worked, I ran also the CPG node and the hardware interface node, connected to only one leg because the others weren't built. To check if the tilt measurements were taken in account by the control functions, I verified that the leg behaved differently when the IMU was in different positions.

First tests with the robot

When the robot was built, we built a stand on which we could put it to run the tests. This stand allowed us to prevent eventual damages if something went wrong. The robot was stable on the stand, without touching the ground. We ran first the hardware interface node and the CPG node, to see how the robot behaved without any sensory

input. As it worked well, we put it on the ground, first without then with the IMU input, and it appeared that the robot was very stable.

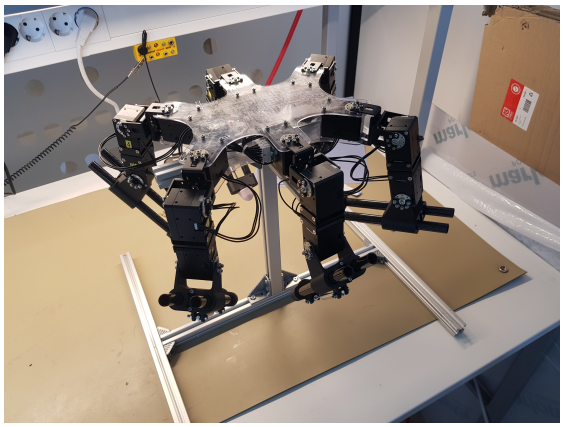


Figure 8: The robot on its stand

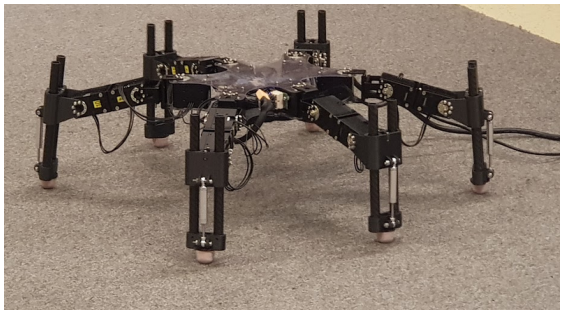


Figure 9: First experiment on the ground

Testing the FSR sensors

Before testing the Arduino node, we had to verify that there was no problem with the soldering or the sensors themselves. That's why, after soldering, we checked with a multimeter that the resistance at the sensors terminals varied when we pressed them.

We needed then to make sure the data was sent correctly by the Arduino board. To do it, I printed the data in the serial monitor on my computer, and pressed each sensor so see how the values varied. When that worked, I ran the Arduino node and printed the values received.

After that, I put the Arduino board in the robot, and try to find the best place for the sensors. Placing them between the silicone foot and the 3D printed part seemed a good idea, but the sensors often went away while the robot was walking on the ground, so I fixed them with hot glue. I ran the robot on the stand, pressed the feet one by one, and plotted the data collected during the experiment. It turned out that the data was very noisy, so it would need a filtering process to be able to use it as a feedback for the

control loop. As it wasn't necessary at that time to make the robot work, I chose to move to the microphone, which was more important for my experiments.

Testing the microphone

As mentioned before, our first idea was to connect a small microphone directly to the Arduino board, on the robot. I tested this idea plotting the output voltage of the microphone on my computer using different types of audio inputs like clapping hands, talking or music. It turned out that the microphone didn't get sound when the source was more than 20 cm away from it. As it wasn't adapted to our use, we changed for a RODE NT-USB microphone.

The tests with this second microphone went better than with the first one. At first, I just recorded some music played with my phone using a test Python code and listened to the recording to hear how it sounded, and also plotted the sound amplitude. The quality of the sound recorded was high, so the microphone seemed to be a good solution.

I tested then to run the robot with the audio input. At first, the legs got stuck in high position due to the high value of the amplitude, but adding a 0.01 gain solved the problem. We saw how the robot reacted to clapping hands and metronome, and it was obvious that the robot followed the rhythm. With music, it wasn't that clear because it seemed that the robot was reacted to every loud sound. Indeed, if the piece of music involved a singer, some syncopation, or for example drum fills, the robot reacted more to it than to the real tempo. As one of the goals of the project was to be able to interact with the robot and as it reacted well to clapping hands, we decided that it was enough for now.

3 Motion capture tests and results

As the robot was now working, we wanted to evaluate his performances in order to compare them to the simulation. This was done using motion capture.

Motion capture (mocap) is a technique that allows to capture positions and rotations of moving objects or limbs of moving objects. It can be done using a lot of various systems, but in our case we used infrared cameras and passive reflective markers. The position of the markers is calculated using geometrical calculus and interpolation. RITMO has a motion capture lab (MoCap Lab) equipped with a Qualisys motion capture system. With the help of Kayla Burnim, Senior Engineer at RITMO, we were able to conduct our experiments easily.

For our performances evaluation, we needed the global position of the robot in the room, we didn't need any data concerning the position of the legs, so we just needed to put three markers on the top plate of the robot. Having three non-aligned markers allows to get the position and the orientation of the robot in space without ambiguity.

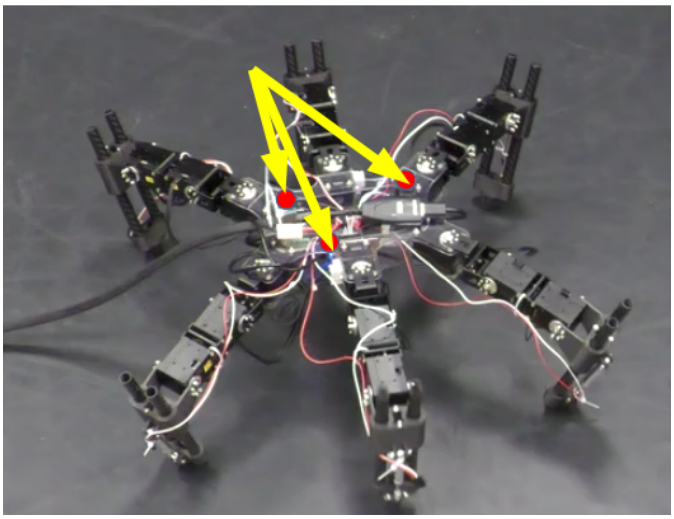


Figure 10: Position of the reflective markers

3.1 Distance comparison

During the internship, Timothé ran an evolution to get a new set of CPGs, optimized for distance and stability. He chose four of them to evaluate. He used them in the simulation and wanted to compare the distance traveled by the robot in simulation and in reality.

Setup

To compare these distances, we made each CPG run in simulation for ten seconds, got the position of the simulated robot during all the experiment, and saved it in separated

`.txt` files. After that, we ran the robot for ten seconds with the same CPGs while recording its position with the motion capture system, and saved it in `.tsv` files. For these experiments, the only sensory input was the one from the IMU. Indeed, the goal here was to evaluate the GPGs in their intrinsic behavior, with their intrinsic gate.

For each CPG we made two trials, to make sure that if a problem was encountered during one of them, we would have a backup. For example, during one of the trials, the robot lost one of its feet, so the friction with the ground was changed and we couldn't use the data got in this trial. We ended up with a total of eight position files for the mocap, and also eight for the simulation.

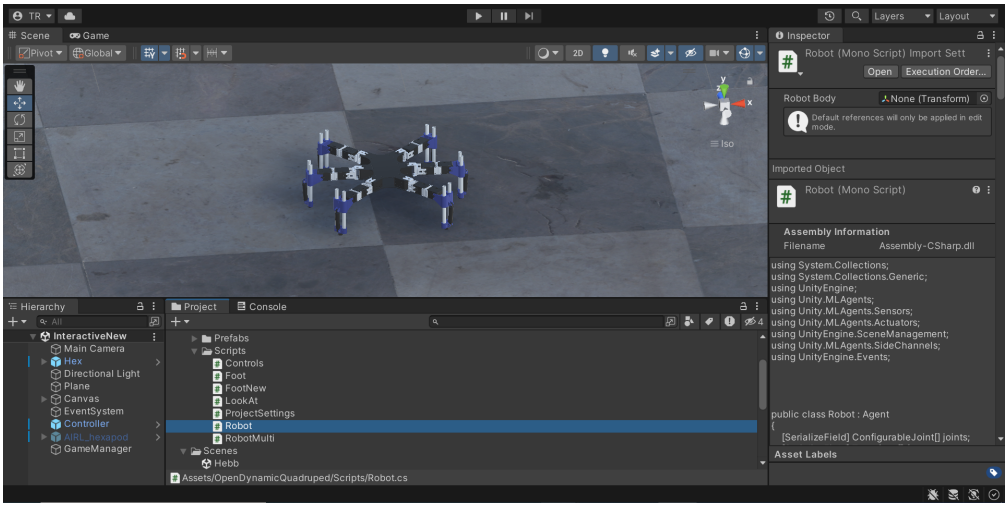


Figure 11: Robot running in simulation

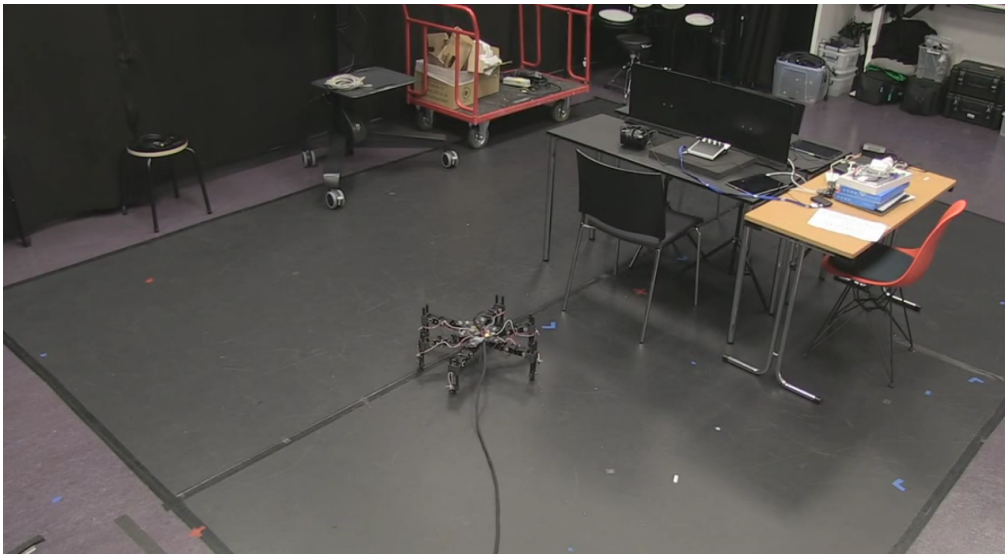


Figure 12: Robot running in the MoCap lab

Data analysis

Now that we had conducted our experiments and had a data set, we needed to analyze them. For that, I implemented two Python codes containing a set of functions. The first one can be used to analyze only one mocap data file at a time, the second can be used to compare two files, either from mocap or simulation recordings.

They can be used to :

- plot **X,Y or Z coordinates** of the three markers over time
- plot **2D trajectory** of the gravity center of the 3 markers
- plot **3D trajectory** of the gravity center of the 3 markers
- calculate the **total, projected and normal distance** traveled by the robot
- calculate the **total, projected and normal speed** of the robot

In the simulation, the robot was oriented along the X axis at $t = 0$, so the projected distance corresponds to the length of the trajectory projected on the X axis, and the normal distance to the length of the trajectory projected along the Y axis, and it works the same way for the projected and normal speeds. For the motion capture experiment, the starting orientation and position were different between each trial. To obtain the projected and normal distances, we therefore create a straight line passing through the robot's center of gravity and oriented according to its starting orientation. The leading coefficient of the projected line is the leading coefficient of the straight line between the front marker and the back marker, as they are aligned with the head of the robot. The trajectory is then projected on this line :

```
# three markers front, back and side
gravity_center_X = (frontX + backX + sideX)/3
gravity_center_Y = (frontY + backY + sideY)/3
coeff = (frontY[0] - backY[0])/(frontX[0] - backX[0])
ord_orig = gravity_center_Y[0] - coeff*gravity_center_X[0]
# straight line
X = np.linspace(0,1000,1000)
Y = coeff*X + org_orig

# projection on the line
X_proj = (coeff*gravity_center_Y + gravity_center_X - coeff*org_orig)
X_proj = X_proj/(coeff**2 + 1)
Y_proj = coeff*X_proj + org_orig

# normal projection
coeff_norm = -1/coeff
ord_norm = gravity_center_Y[0] - coeff_norm*gravity_center_X[0]
X_proj_norm = (coeff_norm*gravity_center_Y + gravity_center_X
```

```
                    - coeff_norm*ord_norm)
X_proj_norm = X_proj_norm/(coeff**2 + 1)
Y_proj_norm = coeff_norm*X_proj_norm + ord_norm
```

The distance (resp. projected distance, normal distance) is then calculated using :

$$d = \sqrt{(x_f - x_0)^2 + (y_f - y_0)^2}$$

where (x_0, y_0) and (x_f, y_f) are the coordinates of the starting and ending points of the trajectory (resp. projected trajectory, normal trajectory). The corresponding speed is then :

$$v = \frac{d}{t_f - t_0}$$

where t_0 is the moment when the robot starts to move and t_f when it stops.

The experiments lasted 10s, and the robot hasn't a constant speed, so the result of this calculation might not be representative of the instantaneous speed. This doesn't really matter because we are more interested in the average speed of the robot over a long duration.

Results

The CPGs numbered 600, 700, 800, and 900 have been tested during the experiments. We compared the forward (projected) and sideway (normal) distance covered by the robot in simulation and in reality in 10s.

	CPG 600		CPG 700		CPG 800		CPG 900	
	Simu	Mocap	Simu	Mocap	Simu	Mocap	Simu	Mocap
Forward speed (m/s)	0.034	0.037	0.720	0.126	0.925	0.073	0.864	0.179
Sideway speed (m/s)	0.199	0.125	0.225	0.294	0.633	0.504	3.124	0.014

Table 1: Comparison of the distance covered by the robot in simulation and reality

The speed is lower in reality than in simulation for most of the cases. As can be seen on 13 the two trajectories have almost the same shape but not the same size. That may be due in part to a lower adhesion of the real robot's feet on the ground.

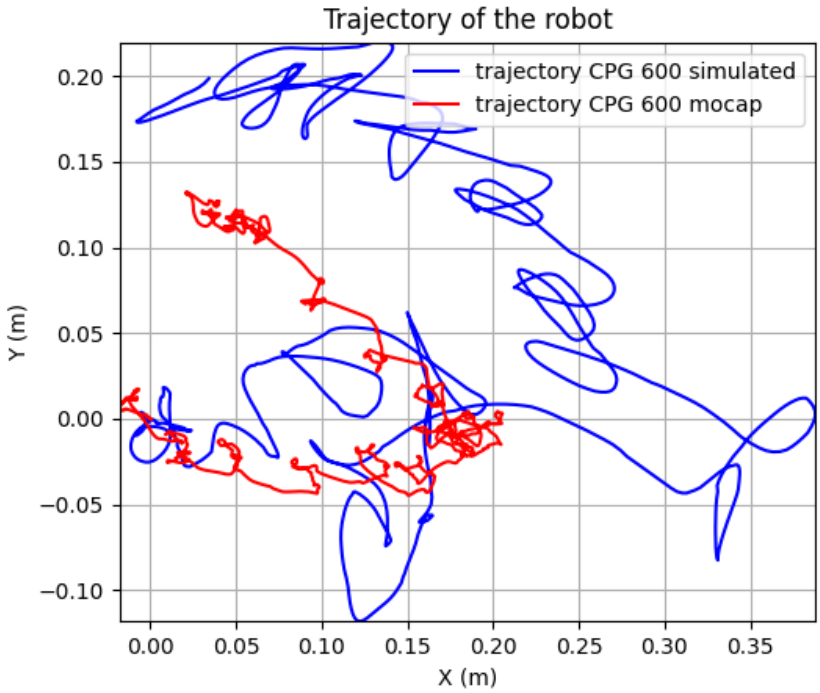


Figure 13: Trajectory of the robot in simulation and reality

3.2 Evaluation of the CPGs' ability to be entrained

This experiment aims to evaluate how the real robot reacts to an audio input. To run these tests, we used a set of 19 different CPGs, 18 of them resulting from the same evolution and the last one (named here "base") coming from a previous evolution.

Setup

The experiment was conducted in the MoCap lab at RITMO. The audio input we used was a metronome installed on my phone. As we didn't have much time to run this experiment, we chose to evaluate only the CPGs that scored the best in simulation (according to the score defined in). For each CPG that we wanted to evaluate, we made the robot run, listening to a metronome set to 80 bpm and then to 120 bpm. As five CPGs have been tested during this experiment, which provides us a set of 10 files to analyze.

Data analysis

We focused on three different points to evaluate the CPGs' performances :

- the score used to rank the CPGs in simulation
- the stability of the robot

- a qualitative evaluation based on how obviously the robot was entrained by the metronome

Score

This scoring method is used in [1]. It evaluates the ability of the robot to adapt its period to the input period. The higher the score

$$F_{fk} = H_{tot,k} \left(1 + \frac{1}{\epsilon} \left| \frac{2T_{out,k}}{T_{in,k}} - \left[\frac{2T_{out,k}}{T_{in,k}} \right] \right| \right)^{-1}$$

where k is the number of the experiment, $H_{tot,k}$ is the mean height of the robot during the experiment, $T_{in,k}$ is the period of the audio input, $T_{out,k}$ is the period of the CPG output and ϵ is a parameter that is set to 0.1.

To obtain $H_{tot,k}$, we calculate the mean height of the robot's markers gravity center using the MoCap data, as it was done in the first experiment. The height is in meters. For the real robot, the height is divided by 0.1 to fit better with the one in simulation.

The input period $T_{in,k}$ is here the time between two beats of the metronome, in seconds:

$$T_{in,k} = \frac{60}{F_{bpm}}$$

where F_{bpm} is the metronome tempo in beats per minute.

The output period $T_{out,k}$ is given by a function that uses auto correlation. In argument of this function, we give the array containing the input positions of the motors, and it returns the number of peaks in one output period. The number of peaks is then multiplied by the delay between to values of positions (here 0.015s) to obtain the output period.

Stability score

To evaluate the stability of the robot, the IMU data is needed.This is done by calculating the standard deviation of the front tilt (x) and side tilt (y), and then the square root of the sum of the squares of the standard deviations. The score is then calculated using the following formula :

$$F_{stability} = \frac{1}{\sqrt{\sigma_x^2 + \sigma_y^2}}$$

A high standard deviation means the orientation of the robot varies a lot. The lower the standard deviation, the more stable the robot, so the higher the score, the more stable the robot.

Results

The ranking for the score evaluation is as follows :

CPG	Score
11	0.6165
3	0.5420
8	0.4937
0	0.4763
base	0.4719

Table 2: Score of the different CPGs on the real robot

This ranking can be compared to the one in simulation :

CPG	Score
3	0.9626
0	0.8523
11	0.8474
base	0.8426
8	0.8120

Table 3: Score of the different CPGs in simulation

None of the CPGs has the same rank in reality and in simulation, and the scores with the real robot are lower than the ones in simulation. This may partly come from the measurement of the height. Indeed, the height measured in the MoCap lab is the height of the top plate of the robot, and in simulation it was the height of the gravity center of the body. The audio input is also not exactly the same, the tempi are different and the rhythm too.

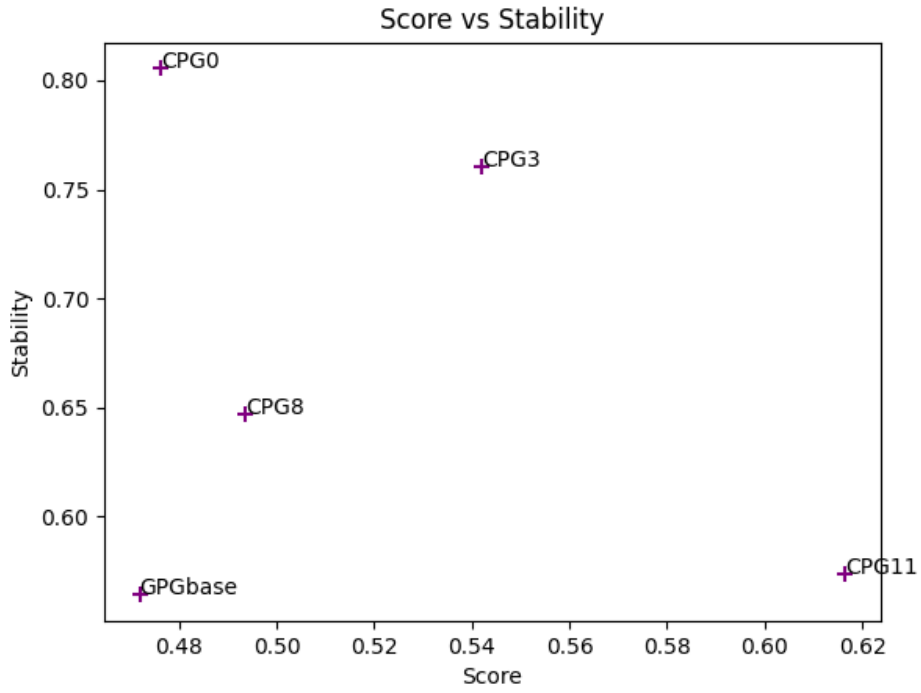


Figure 14: CPGs' stability vs score

The idea was also to evaluate the CPGs qualitatively, by looking at the behavior of the robot during the experiment. For some CPGs, it is obvious that the robot reacts to the audio input, but for others, it is not so clear, and that's what I evaluated here.

By watching the videos of the experiments, I ranked the CPGs from the one that is entrained the most obviously by the audio input to the one that doesn't seem entrained at all.

Rank	CPG
1	base
2	0
3	11
4	8
5	3

Table 4: Qualitative evaluation of the different CPGs with the real robot

What is interesting to notice is that the CPG that is the most obviously entrained (CPG base) is the ones that has the lowest score. The low stability is due to the fact that the robot raises four of his legs at the same time on the audio onset, which makes him unstable.

Also, the CPG 3 has a high stability and a high score compared to the other, but is the one that is the least obviously entrained.

To conclude, we observe that a high score doesn't necessarily mean that the entrainment of the robot is obvious.

A way to get a better score and behavior could be to improve the sound filter to make the tempo detection more precise and more adapted to complex music. Indeed, the filter has been developed and tested using very clear studio recordings, so trying to improve the filter using music recorded with the Rode microphone could make it more robust and help to improve the score.

Also, even if some of the CPGs have a lower stability than the others, they are still pretty stable when we look at the videos of the experiments. None of them fell during the experiments, and the robot was able to walk without any problem.

4 Work completed

During the internship, I contributed to improving and completing the GitHub repository of the project, giving the team an easy access to all the code I implemented. Moreover, I sent them a short report giving detailed explanations on how my code works, so that they will be able to reuse all of this, and modify it according to their needs. This report contains a detailed description of each important function, type of the data involved, topics the nodes subscribe to or publish on, and a troubleshooting section listing the problems and errors I encountered and some solutions.

The week before the end of the internship, RITMO hosted the Entrainment workshop, which aimed "to bring together an interdisciplinary group of researchers, all engaged in the pursuit of a better understanding of various processes and outcomes of entrainment" (source : RITMO web page of the event). During this workshop, we presented a live demonstration of the robot. We explained the goals of the project and an overview on how the robot works. We showed the simulation, and then asked the public to clap their hands, so that the robot could follow the rhythm. The demo went well and the robot behaved exactly as we expected. It was a great opportunity for us to present our work to interdisciplinary researchers, and to see the robot interact with so many people.

Figure 15: Presentation of the project during the Entrainment Workshop

5 Conclusion

During this 16-week internship in Norway, I was able to experience working in a international context. I learned how to start working on a project already in progress, using someone else's code as a basis, and producing something that has to be easily reusable for the team. From a technical point of view, I got more familiar with tools that we use in my engineering training, like ROS 2 or C++. I also improved my hardware skills by learning more about electronics and building a robot. All these skills will be useful for my future life as an engineer.

From a personal point of view, these four months abroad allowed me to learn more about myself and about how I work in a team, and it also helped me clarify my professional project. This internship will remain for me an essential moment of my engineering training.

List of Acronyms

ENSTA Bretagne École Nationale Supérieur des Techniques Avancées Bretagne

IMU Inertial Measurement Unit

FSR Force Sensitive Resistor

RMS amplitude Root Mean Square amplitude

CPG Central Pattern Generator

List of Figures

Bibliography

[1] Alexander Szorkovsky, Frank Veenstra, and Kyrre Glette. Central pattern generators evolved for real-time adaptation to rhythmic stimuli. 2023.

[2] John Voit. *Quaternion Algebras*. Springer Nature, 2021.

[3] Eric W. Weisstein. Euler angles. *MathWorld–A Wolfram Web Resource*.

Assessment Report

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* RITMO Centre, University of Oslo

Adresse / *Address* Forskningsveien 3A, 0373 Oslo

Tél / *Phone (including country and area code)* +47 2285 4489

Nom du superviseur / *Name of internship supervisor* Dr Alex Szorkovszky

Fonction / *Function* Postdoctoral Researcher

Adresse e-mail / *E-mail address* alexansz@ifi.uio.no

Nom du stagiaire accueilli / *Name of intern* Marguerite Miallier

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak).***

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? ☑ B C D E F
  Was the initial contract carried out to your satisfaction?
- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☑ non/*no*
  Was the intern lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

  ☑ B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?

Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

✔B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

_____

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? ✔B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

_____

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? ✔B C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____

_____

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : ✔B C D E F
Please evaluate the technical skills of the intern:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☐ oui/*yes* ☑ non/*no*

Fait à _____, le _____
In _____, on _____

Signature Entreprise _____ Signature stagiaire
Company stamp _____ *Intern's signature*
A. SZOKOVSZKY

Merci pour votre coopération
We thank you very much for your cooperation

scoring.py

```
import numpy as np
import matplotlib.pyplot as plt

pos_list = ['mocap_data/pos_Ant1_CPGbase_15082023_Filter80_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPGbase_15082023_Filter120_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG0_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG0_23082023_Filter120_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG3_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG3_23082023_Filter120_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG8_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG8_23082023_Filter120_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG11_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/pos_Ant1_CPG11_23082023_Filter120_config4_MoCap.npy'
            ]
mocap_list = ['mocap_data/Ant1_CPGBase_15082023_Filter80_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPGBase_15082023_Filter120_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG0_23082023_Filter80_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG0_23082023_Filter120_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG3_23082023_Filter80_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG3_23082023_Filter120_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG8_23082023_Filter80_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG8_23082023_Filter120_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG11_23082023_Filter80_Config4_MoCapTrial0001.tsv',
            'mocap_data/Ant1_CPG11_23082023_Filter120_Config4_MoCapTrial0001.tsv']

imu_list = ['mocap_data/imu_Ant1_CPGbase_15082023_Filter80_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPGbase_15082023_Filter120_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG0_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG0_23082023_Filter120_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG3_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG3_23082023_Filter120_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG8_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG8_23082023_Filter120_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG11_23082023_Filter80_config4_MoCap.npy',
            'mocap_data/imu_Ant1_CPG11_23082023_Filter120_config4_MoCap.npy']

# from MathUtils.py

def autocorr(allx, start, mindelay, maxdelay=-1):
    n_i = allx.shape[0]
    tt = allx.shape[1]
    tstart = round(tt*start)
    out = np.zeros([n_i, 2*(tt-tstart)-1])
    if maxdelay < 0:
```

```
        maxdelay = tt-tstart
    for i in range(n_i):
        x = allx[i, tstart:]
        out[i, :] = np.real(np.correlate(
            x - np.mean(x), x - np.mean(x), mode='full'))
        #new: remove limbs with no correlation peak
        if np.argmax(out[i,(len(x)+mindelay):(len(x)+maxdelay)]) == 0:
            out[i, :] = 0
    outtot = np.sum(out, axis=0)
    peak = mindelay + np.argmax(outtot[(len(x)+mindelay):(len(x)+maxdelay)])
    height = np.max(outtot[len(x)+mindelay:])
    if peak == mindelay:
        peak = 0

    return peak, height

def calc_H_tot(filename):
    data = []
    with open(filename,'r') as file:
        for line in file:
            l = line[:-1]
            l = l.split('\t')
            # l = [float(i) for i in l]
            data.append(l)
    data.pop() # in some files, the last line is equal to zero, which creates errors,
    # so we remove it
    data = np.array(data[12:])
    data = data.astype(np.float32)
    _, time, _, _, frontZ, _, _, backZ, _, _, sideZ = data.transpose()
    Z = (frontZ + backZ + sideZ)/3
    Z = Z/1000
    r = 0.01
    # find time of the fisrt movement and the last movement
    for i in range(len(Z)):
        if Z[i] > Z[0]+r:
            start = i
            break
    for j in range(len(Z)):
        if Z[-j] > Z[0]+r:
            end = len(Z)-j
            break
    Z = Z[start:end]
    H_tot = np.mean(Z)/0.1
    return H_tot

def score(cpg_file, mocap_file , period):
```

```
    print('cpg_file = ',cpg_file)
    if "Filter80" in cpg_file:
        bpm = 80
    else:
        bpm = 120
    print('bpm = ',bpm)
    cpg = np.load(cpg_file)
    H_tot = calc_H_tot(mocap_file)
    peak, height = autocorr(cpg.transpose(), 0, 7)
    print('peak = ',peak)
    T_out = peak*period
    print('T_out = ',T_out)
    T_in = 60/bpm
    print('T_in = ',T_in)
    eps = 0.1
    Q = 1/(1 + (1/eps)*abs(2*T_out/T_in - np.around(2*T_out/T_in)))
    H_tot = calc_H_tot(mocap_file)
    print('H_tot = ',H_tot)
    return H_tot*Q

def stability_score(filename):
    data = np.load(filename)
    data = data.transpose()

    std = np.std(data, axis=0, ddof=1)
    score = 1/np.sqrt(np.sum(std**2))

    return score


if __name__ == '__main__':

    score_list = []
    for j in range(len(pos_list)):

        score_list.append(score(pos_list[j], mocap_list[j], 0.015))
    # print(score_list)
    scorebase = score_list[0] + score_list[1]
    score0 = score_list[2] + score_list[3]
    score3 = score_list[4] + score_list[5]
    score8 = score_list[6] + score_list[7]
    score11 = score_list[8] + score_list[9]
    final_score = [scorebase, score0, score3, score8, score11]

    print(final_score)
    stability_score_list = []
```

```
for k in range(len(imu_list)):
    stability_score_list.append(stability_score(imu_list[k]))
print(stability_score_list)
print(np.sqrt(stability_score_list[0]))
stabilitybase = np.mean([stability_score_list[0],stability_score_list[1]])
stability0 = np.mean([stability_score_list[2],stability_score_list[3]])
stability3 = np.mean([stability_score_list[4],stability_score_list[5]])
stability8 = np.mean([stability_score_list[6],stability_score_list[7]])
stability11 = np.mean([stability_score_list[8],stability_score_list[9]])
stability_score_list = [stabilitybase, stability0, stability3, stability8,
                        stability11]



# plot the score and stability labelling each point with the name of the cpg
fig, ax = plt.subplots()
ax.scatter(final_score, stability_score_list,50, marker='+', color='purple')
for i, txt in enumerate(['GPGbase', 'CPG0', 'CPG3', 'CPG8', 'CPG11']):
    ax.annotate(txt, (final_score[i], stability_score_list[i]), fontsize=10)
plt.xlabel('Score')
plt.ylabel('Stability')
plt.title('Score vs Stability')
plt.show()

# results : a higher stability score means the robot is more stable
# the CPG with the best score is CPG11,
# but it is not the most stable (almost the worst)
# the CPG with the best ratio score/stability is CPG3
#(the second best score and the second best stability)
# CPGbase is the worst both in stability and score

# rank by qualitative evalutation :
# 1) CPG base
# 2) CPG 0
# 3) CPG 11
# 4) CPG 8
# 5) CPG 3

# score :
# [0.04719353169202806, 0.04763573881076731, 0.054202513464612745,
# 0.04937176192584246, 0.06165931383147828]
# rank :
# CPG 11 : 0.06165931383147828
# CPG 3 : 0.054202513464612745
# CPG 8 : 0.04937176192584246
# CPG 0 : 0.04763573881076731
```

```
# CPG base : 0.04719353169202806

 # score with htot/0.1 :
 # [0.47193531692028057, 0.4763573881076731, 0.5420251346461273,
 # 0.4937176192584246, 0.6165931383147829]
```

Sensors documentation

Technical specifications of the UM7 orientation sensor

- EKF estimation rate: 500 Hz
- Typical static pitch/roll accuracy: +/- 2°
- Typical dynamic pitch/roll accuracy: +/- 4°
- Typical static yaw accuracy: +/- 5°
- Typical dynamic yaw accuracy: +/- 8°
- Angle repeatability: 0.5°
- Angular resolution: 0.01°
- Input voltage: 5V
- Communication: 3.3V TTL UART, SPI bus
- Supported baud rates: 9,600, 14,400, 19,200, 38,400, 57,600, 115,200, 128,000, 153,600, 230,400, 256,000, 460,800, 921,600
- Power consumption: 50 mA @ 5V
- Operating temperature: -40 to +85°C
- Data output rates: 1 Hz to 255 Hz (binary packets), 1 Hz to 100 Hz (NMEA packets)
- Output data: orientation and heading (Euler Angles), attitude quaternion, GPS altitude, position, velocity (w/ external GPS), magnetometer, accelerometer, and gyro data
- Dimensions: 27 x 27 x 6.5 mm
- Weight: 11 g
- **Included: a 5-pin connection cable on one side, naked wires on the other side**

Figure 16: IMU-UM7 specifications (source: https://www.generationrobots.com/en/403281-um7-orientation-sensor.html)

Acoustic Principle	Pressure Gradient
Active Electronics	JFET impedance converter with bipolar output buffer, A/D converter 16bit 48kHz
Capsule	0.50"
Polar Pattern	Cardioid
Bit Depth	16-bit
Address Type	Side
Frequency Range	20Hz - 20kHz
Maximum SPL	110dB SPL
Power Requirements	USB
Output Connection	USB and 3.5mm headphone

Figure 17: RODE NT-USB specifications (source: https://rode.com/fr/microphones/usb/nt-usb)