UCLouvain

École polytechnique de Louvain

Practical Domotics Using Sensor Fusion: Responsibility Handover of a Mobile Robot in a Building

Authors: **Nicolas Daube, Thomas Vanbever**
Supervisor: **Peter Van Roy**
Readers: **Tom Barbette, Peer Stritzinger**
Academic year 2024-2025
Master [120] en sciences informatiques / Master [120] : ingénieur civil en génie de l'énergie

Abstract

Domotic systems are becoming increasingly prevalent in our daily lives, driven by both the evolution of connected devices and advances in new technologies. Businesses and consumers alike are increasingly encouraged to take advantage of these innovations, both to improve their performance and for the comfort they offer. These systems most often use multi-agent sensor networks that must operate reliably even in changing or constrained environments. In such contexts, sensor data fusion and dynamic management of information sources become major challenges for maintaining the accuracy of estimates and ensuring continuity of operations.

This master's thesis studies the integration of a dynamic sensor responsibility-transfer mechanism between data sources used for sensor-fusion. Using GRiSP2 prototyping boards and the Hera framework, we developed a multi-agent architecture enabling a self-balancing robot to navigate a scaled-down model of a multi-room building. The system combines ultrasonic sonars placed in each room with an onboard inertial sensor to perform real-time sensor fusion for estimating the position and orientation of our robot agent.

"The future of work is not about replacing humans with machines;
it's about augmenting human capabilities with technology"

— Satya Nadella [12]

Acknowledgment

We wouldn't have been able to finish this work on our own. We thus owe those people a thank you. We would specially like to thank :

***Prof. Peter Van Roy**, our supervisor, for his support, his guidance, his good advice and his enthusiasm for this project.*

*All the people at **Peer Stritzinger Gmbh** for their help on the usage of the GRiSP technology.*

***Cédric Ponsard** and **François Goens** for the help they provided in building our robot prototype and understand their design choices.*

***Simon De Jaeger** and **Thierry Daras** for their help and advices on all the hardware we built during this past year.*

***Vanessa Maons** for all the administrative work and the access to the Stévin open space.*

*Finally, all our **relatives**, **friends** and **families** for their support during this year and all our previous college years. We owe a special thanks to **Sylvie Baudine** for her proofreading of this document.*

AI Use Disclaimer

Large Language models were used throughout the work done on this master's thesis. We used ChatGPT to supercharge and accelerate documentation and API research and to help building scripts to accelerate our development. We also used tools like Deepl and Deepl Write to paraphrase some text in this document.

Contents

Part I

Practical Domotics Using Sensor Fusion: Responsibility Handover of a Mobile Robot in a Building

Chapter 1

Introduction

1.1 Context

In 1960, Rudolph E. Kalman introduced the first version of his algorithm to the Ames Research Center [3]. The aim of his algorithm was to infer an unknown state using available noisy data. This process is known as sensor fusion. This term is used for filters that use data from different sensors to improve the output data or to deduce new state variables, as in the case of the Kalman filter.

Later on, NASA's Dynamic Analysis Branch worked on the next iteration of the filter. [11]. Their work led to the separation of the algorithm into two separate steps: prediction and update. This separation allowed the asynchronous processing of measurements coming from the different sensors, as prediction steps could be done without new data.

Still, that evolved version does not have the means to choose the most relevant data in a list of available measurements, nor to switch dynamically between different sensors following their actual state estimation.

In today's world, where Internet of Things (IoT) and domotic systems are everywhere around our homes, this ability becomes critical. In a system where the environment in which a Kalman filter operates changes or is discontinued, some sources of data may become irrelevant or useless. Using irrelevant data in a Kalman Filter may produce estimates without any relation with the real-world state, which can be catastrophic.

1.2 Objectives

The goal of our master's thesis is twofold.

First, we aim to give a robot the ability to orient itself in a complex full scale building. Put simply, our goal is to make the robot aware of its location. In order to achieve this feat, we will create and implement an IoT system allowing a robot to move around in a reduced scale mockup of a house, using data coming from sonars placed in the rooms to deduce its position. This small scale test environment will give us a great starting point for the potential development of a full scale prototype as it takes into consideration all the constraints that a real-world building would impose.

Second, we will incorporate the possibility of the handover of responsibility between the sensors used by the sensor-fusion. In other words, we aim to demonstrate the feasibility for a system to dynamically reassign the responsibility to different data sources used for sensor fusion and adapt to the resulting change of responsibility.

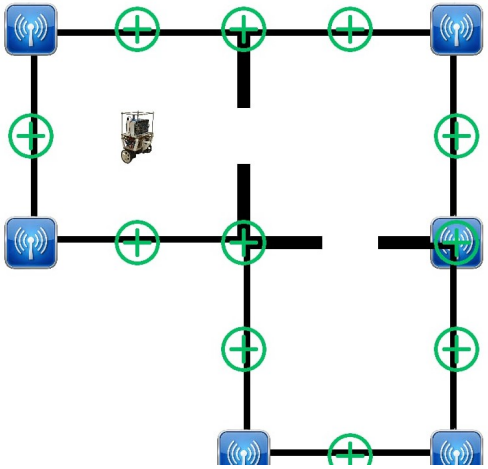
1.3 Contributions

We can identify the following list of contributions, resulting from the work done for this master's thesis:

- We refactor and extend the software of a robot built for a previous master's thesis and integrate it in a larger system.
- We create a multi-agent distributed system using GRiSP2 prototyping boards and sensor-fusion to allow a robot to determine its position. In other words, we add a new layer of awareness (location awareness) in the process of the robot.
- We demonstrate the feasibility for a robot to dynamically hand over its focus to different groups of sensors, modifying its sensor-fusion operations in the process.
- We demonstrate how useful the Hera framework is for the creation of an IoT distributed system.
- We add new functionalities to the Hera framework, including the possibility to handle the loss of connectivity, to create a publisher/subscriber system or to communicate with non-Erlang systems.

- We clarify the organization and document the Hera framework's API and inner works.
- We implement a time multiplexing like protocol for the prevention of cross talk interferences in the sonar sensors.
- We create a propagation protocol that allows communication in a discontinued environment.
- We implement an executive function above a timeclock in order to check whether a measurement is needed.
- We implement a Kalman filter using Hera to determine the position, angle and room in which an object is moving.
- We create a data representation of a mutli-room building. Figure 1.1a shows the physical representation we made of a three room building and Figure 1.1b shows its digital representation.

(a) Physical Mockup Representation of a Three Room Building



(b) Digital Representation of a Three Room Building

Figure 1.1: Building Representation

All the code developed for this master's thesis is available on the GitHub Grisp_robot repository[1], on the Hera repository[2] or in Appendix to the present document.

[1]https://github.com/Nicodaube/Grisp_robot
[2]https://github.com/Nicodaube/hera

1.4 Roadmap

This document is divided into five main parts:

- Chapter 2 summarizes all the background knowledge necessary to understand the following development. Section 2.1 explains all the fundamental knowledge and Section 2.2 details the three master's thesis on which this one builds.
- Chapter 3 presents the system that we designed. Section 3.1 explains the creation of the physical mockup and parts used to test the system and Section 3.2 explains the system's architecture and the different protocols designed for our system.
- Chapter 4 deals with the actual implementation. Section 4.1 details our work on Hera, Section 4.2 talks about the room agent, Section 4.3 shows our additions to the robot and Section 4.4 explains the use of the Laptop Controller.
- Chapter 5 explains how we tested each feature of our system and shows the according results. Section 5.1 talks about the errors we encountered in the sonar measures and how we got around them. Then, Section 5.2 shows the results of our Kalman filter tests. Following this, Section 5.3 shows the results of our tests on the handover protocol and on the propagation protocol.
- Lastly, Chapter 6 will finish by discussing our system, its limitations and the work that could further improve our present contributions.

Chapter 2

Background

2.1 Fundamentals

2.1.1 The GRiSP2 board

GRiSP2 are the heart of our system, they are the main computing unit used in the robot and the sole one in each of our sensors. At the maximum tested size, our system features seven of those boards.

The GRiSP2 Board [9] is a system on a module (SoM) prototyping platform developed by Peer Stritzinger GmbH [10]. Designed with IoT and distributed systems in mind, it is able to run an Erlang/OTP virtual machine (known as BEAM). This board features a large range of input/output ports, such as GPIO, UART, SPI, I²C, Ethernet, WiFi, USB and more.

GPIO (General Purpose Input Output) pins are the most basic wired way of interfacing with outside components. Every pin can individually be set either as an input or an output, making it very modular. The user can then use software to put its state as 1 or 0. This very manual usage makes it perfect for interfacing LEDs or buttons, but limits its use for communication purposes.

UART (Universal Asynchronous Receiver/Transmitter) are point-to-point serial links using a master transmitting to a slave. It can transmit data based on an agreed baud rate, common for both receiver and transmitter. It uses a set of three links, namely a GND (Ground) link, an RX (Receiver) and a TX (Transmitter) link.

SPI (Serial Peripheral Interface) is a high-frequency way of transmitting data from a master to a set of slaves and vice versa. It uses a CLK (Clock) pin, a MISO (Master In Slave Out) pin, a MOSI (Master Out Slave In) pin and a CS (Chip Select) pin for slave selection. It is ideal for high throughput needs.

I²C (Inter Integrated Circuit) is a multi-master connection, it is slower than SPI, but only uses two connections : a CLK wire (Clock) and an SDA wire for data. Its built-in addressing allows multi peripherals on the same bus.

Each GRiSP2 board also features five jumpers that can be set on or off.

GRiSP2 boards run the BEAM virtual machine thanks to an RTEMS (Real-Time Executive for Multiprocessor Systems) [15] instance. RTEMS is an open source version of the real-time operating system (RTOS). It supports multiple architectures and is used in a wide range of industries including the medical and aerospace sectors.

The GRiSP2 boards possess several Pmod enabled ports. These enable us to use Digilent Pmod sensors to extend the capabilities of the GRiSP2 boards. Those Pmod inputs are used in two critical components of the developed system, as explained below.

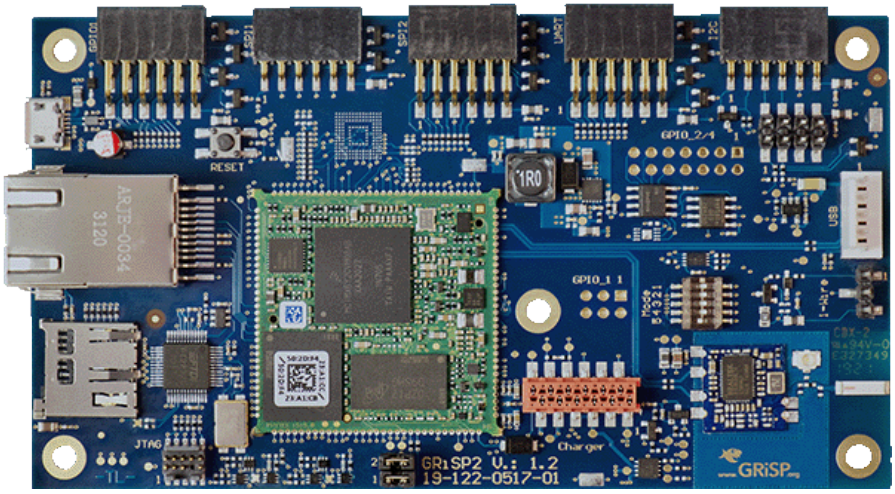


Figure 2.1: The Peer Stritzinger Gmbh GRiSP2 Board [9]

Pmod MaxSonar

The Digilent Pmod MaxSonar [8] is an ultrasonic range finder that can be connected to the UART port of the GRiSP2 board. It uses a burst of thirteen 42kHz impulses of varying width to measure ranges between 6 to 255 inches (15.24 to 648cm). This measure can be read every 50ms with a precision of around 2.54cm. It is mounted on each of the components we call "sensors" and used to determine the distance at which the robot is located.

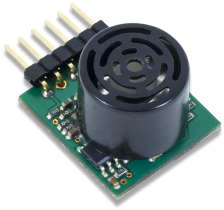


Figure 2.2: The Digilent Pmod MaxSonar [8]

Pmod Nav

The Digilent Pmod Nav [18] is a multi-sensor Pmod module composed of a 3-axis accelerometer, a 3-axis gyroscope, a 3-axis magnetometer and a barometer. It connects to the SPI Bus of the GRiSP boards and is extensively used in the robot in the two running Kalmans filters (stability and position).

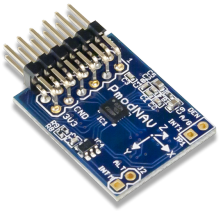


Figure 2.3: The Digilent Pmod Nav [18]

2.1.2 Erlang programming language

The Erlang programming language is a multi-agent message passing language. It was originally developed by the Ericsson Computer Science Laboratory, with telecommunication applications in mind

Erlang comes with a motto: "*Let it crash!*", meaning that programs are built to handle the possibility of failure. Erlang achieves its goal by creating a hierarchy tree of lightweight processes. The parents, called supervisors, are in charge of restarting faulty processes following a predetermined strategy. The available strategies for restart are one for one (only the faulty child gets restarted), one for all (when one child dies, all children are restarted), simple one for one (restart the faulty instance in a set of same processes) and rest for one (restart the faulty child and all the children started after this one).

Behaviours are an other important Erlang concept. The Erlang documentation [4] defines behaviours as a formalization of the common patterns occurring in different processes. For example, two supervisors only differ in their children and in their way to supervise them. We will bring back this concept when explaining how Hera works.

This multi-agent behaviour also helps with the development of distributed, concurrent and elastic systems. Its message-passing behaviour allows data to be passed between different Erlang processes. The programming language comes with OTP (Open Telecom Platform), a collection of Erlang libraries allowing for easier development.

To maximize availability, Erlang incorporates a "hot-loader" capability that allows users to update code without shutting down the whole application.

As far as its downsides are concerned, Erlang is not aimed at performing intensive computation tasks. This lack of performance may slow the programs down when heavy computations are required. To accelerate those computations or access lower components of the system, Erlang allows the integration of NIFs (Native Implemented Functions). These function blocks are written in C and can improve the performance of the system significantly. Of course, NIFs come with a tradeoff. When you enter a NIF, you leave the Erlang safe environment, abandoning the benefits of its fault tolerancy. NIFs thus need to be used with care as they induce BEAM VM crash possibilities in the system.

2.1.3 Signal processing filters

Low pass filter

The Low Pass Filter (LPF) is a well-known signal processing filter. It is used to smooth data measurement, filtering out the quick changes, often synonym of noise or errors. It works by giving a chosen weight to the previous measure relative to the new one. Using a parameter α we can define the new measure x at step k as:

$$x_k = \alpha * x_{k-1} + (1 - \alpha) * x_k$$

Hampel filter

The Hampel filter [14] is a filter using a sliding window to detect and filter anomalies or outliers in a series of data points. It works using the Median Absolute Deviation (MAD) [2]. It measures the amount of variation in a set of data, much like the variance but much less influenced by extreme values. The MAD of a list of points Y is defined as :

$$\tilde{Y} = median(Y)$$
$$MAD = median(|Y_i - \tilde{Y}|)$$

Let $\tilde{Y}$ be the median of the original Y list. The MAD is defined as the median of the list formed by the absolute difference between each value and this median $\tilde{Y}$.

The Hampel filter uses two different parameters :

1. The **Half window size** K: the size of the sliding window. The window length must be odd to allow for a true median. The true size of the window is thus $2K + 1$.
2. The **Threshold** n_σ: the strength of the filter determines when a data point is considered an outlier.

The algorithm is as follows [24] :

- First we add the sample to the window (buffer).
- Then we compute the median of the buffer (M) and the MAD (MAD) of the window
- At last, the output value (x) is M if

  $$|x - M| > n_\sigma * MAD$$

  else return the measure x.

2.1.4 Kalman filter

The Kalman filter is a mathematical algorithm that efficiently and recursively solves the problem of estimating the state of a dynamic system from noisy and incomplete measurements. It operates by maintaining a correction prediction cycle. First, it uses a mathematical model to predict the system's next state. It then corrects this prediction using real-world measurements as they become available.

Initially, the filter predicts the state of the system and the associated uncertainty. When a new measurement is received, the filter updates its prediction by calculating a weighted average between the predicted state and the new measurement, the weights being inversely proportional to their uncertainties. This correction reduces the estimation error by optimally combining model information and sensor data.

The Kalman filter assumes that all noise processes are Gaussian and that the system can be described by linear models. Under these assumptions, it produces an optimal estimate in the sense that it minimizes the mean squared error.

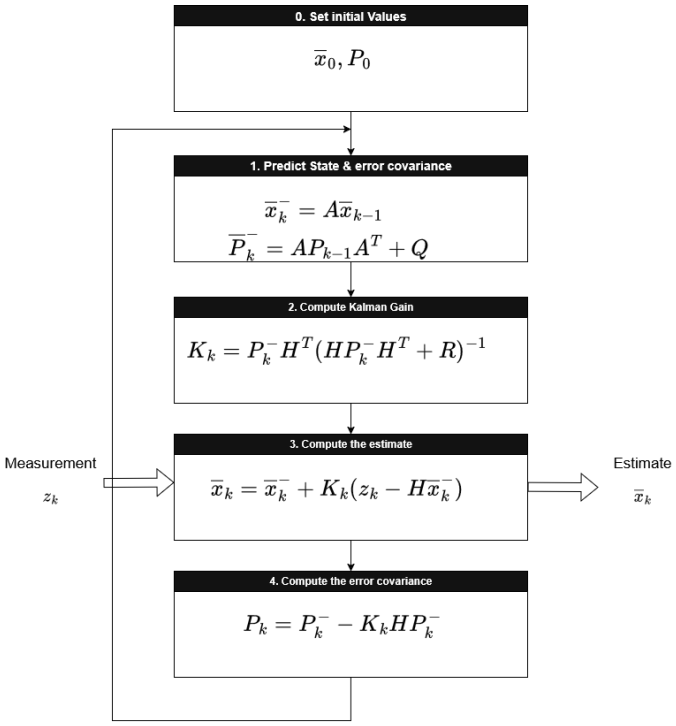


Figure 2.4: Kalman Filter Algorithm Flow Chart

As shown in Figure 3.4, the process comprises two main steps (prediction and update) repeated at each iteration :

- Prediction: The filter estimates the current state and the associated uncertainty based on the previous state and the system dynamics.
- Update: Upon receiving a new measurement, the filter updates the predicted state and uncertainty by incorporating the new information.

Each iteration of this filter evaluates a vector and a matrix.

- The vector X represents the state of the system
- The covariance matrix P of the error on the state X. It is used to measure the confidence level on the estimation of the vector X.

Prediction phase

This part is only based on the system's physical model and the command given to it.

$$\hat{x}_k = F_k x_{k-1}$$
$$\hat{P}_k = F_k P_{k-1} F_k^\top + Q_k$$

Update phase

The update step combines both the prediction and the measurement to improve the estimate of the state X and its associated uncertainty P.

$$K_k = \hat{P}_k H_k^\top (H_k \hat{P}_k H_k^\top + R_k)^{-1}$$
$$x_k = \hat{x}_k + K_k(z_k - H_k \hat{x}_k)$$
$$P_k = (I - K_k H_k)\hat{P}_k$$

Explanation of the different matrices and vectors in the Kalman calculation

- **F**: State transition model. This matrix describes the evolution of the state based on the mathematical model of the problem.
- **Q**: Covariance matrix of the evolution of the noise from the prediciton. It quantifies the noise introduced into P post-prediction, reflecting the model's inaccuracy.

- **R**: Covariance matrix of the evolution of the noise from the sensor. This term accounts for the noise and inaccuracies present in the measurements.
- **H**: Observation model : It is a mapping matrix to link the state of the system and sensor measurements.
- **z**: Sensor measurement vector.
- **K**: Kalman gain matrix : The Kalman gain weights the confidence in the sensor and the prediction in order to combine them in such a way that the variance of P is reduced after performing the update.

The most visual way to see the Kalman filter is that it calculates the product of two Gaussian curves. This calculation produces a new Gaussian curve that is less noisy.

2.1.5 Extended Kalman filter

The extended Kalman filter is the non-linear version of the Kalman filter, linearizing the model around the current estimate.

In the extended Kalman filter, the state transition and observation models do not have to be linear functions of the state, but can be differentiable functions.

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1} \tag{2.1}$$
$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{2.2}$$

Where f is a nonlinear function, $\mathbf{u}_{k-1}$ is a control input and $\mathbf{x}_k$ is the current state. the function h is the measurement function that relates the current state to the measurement $\mathbf{z}_k$. The Gaussian noises for the process model and the measurement model are $\mathbf{w}_{k-1}$ and $\mathbf{v}_k$ with covariance Q and R.

As with the basic Kalman filter, there is a prediction and an update step. This is how they are described in the extended Kalman filter:

Predict

Predicted state and covariance estimates:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_{k-1})$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_{k-1}$$

Update

Innovation or measurement residual and covariance:

$$\tilde{y}_k = z_k - h(\hat{x}_{k|k-1})$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k$$

Near-optimal Kalman gain:

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{2.3}$$

Updated state and covariance estimates:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

The state transition and observation matrices are defined by the Jacobian :

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}, u_{k-1}} \tag{2.4}$$

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}} \tag{2.5}$$

2.1.6 The LilyGo TTGO LoRa32

The LilyGo TTGO LoRa32 (named LilyGo later in this document) is a small unit equipped with an ESP32 micro controller capable of handling WiFi and Bluetooth connections. It features a LoRa chip and a small OLED display. LoRa (Long Range) is a wireless communication method focused on range and low power consumption. In the context of this project, the LilyGo is used by the Laptop controller to send commands to another LilyGo on the robot. That LilyGo is also in charge of forwarding commands from the GRiSP board to the 3 stepper motors, and of creating the WiFi access point used by the whole system.

2.2 Previous Work

The present work builds directly on three previous master's theses, all supervised by Prof. Peter Van Roy. Each of them contributed in components extensively used or improved in our work. They provided a solid foundation combining theoretical analysis and design decisions, offering a solid basis upon which to build, solve our own challenges and identify new opportunities. Here, we outline the key takeaways of these theses and outline how their findings have been essential to make decision for our own one.

2.2.1 The Hera framework

Sébastien Kalbusch and Vincent Verpoten [17], introduced the first version of Hera in their master's thesis in 2021. Built for the GRiSP board, this framework was created with the aim of facilitating the development of GRiSP IoT/Sensor fusion applications.

Hera aims at incorporating the storage, sharing and broadcasting of data to a set of distributed nodes through different interconnected processes. It allows framework users to develop a top level application seamlessly using its functionalities.

Every Hera process is built as a child of the hera_sup supervisor process. This allows for the restart of any faulty process and thus ensures a maximal availability of the framework's functionalities.

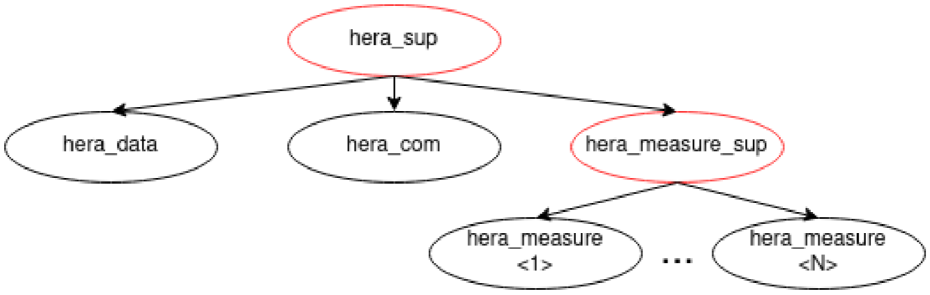


Figure 2.5: Hera Supervision Tree [17]

Originally, Hera was composed of 5 main modules :

1. **hera_com** is in charge of all the external communication and connection to the WiFi. hera_com is built to use UDP messages for data communication between nodes.

2. **hera_data** is the process in charge of storing and retrieving information, that is internally packaged in a big map data structure.

3. **hera_measure** defines a basic behaviour that can be extended by each user-defined application to periodically gather data. Its high level utilization is simple. Users only have to define two functions: *init/1*, that takes a facultative list as argument, and *measure/1*, that takes a State map as argument. The return of the init function optionally contains the name of the hera_measure instance, a timeout defined by the user, that corresponds to the time between two invocations of the measure function by Hera, and other options. Once the measurements are made, the user can return this measurement It will then be stored by hera_data and shared via hera_com.

4. **mat** defines all the necessary operations that can be executed on matrices.

5. **kalman** (now renamed hera_kalman in the newer version): defines all the computations (using the mat module) required for Kalman filters resolutions. Users only have to define the parameters of the filter and call the functions to solve it.

The main design principle behind Hera is that each node takes measurements, share them and use the gathered data to infer new information. Through a sequence number, each node has easily access to the most recent version of each type of measurement.

Since that initial framework was created, Hera has been expanded and improved by other students to match their master's theses needs. Our own contribution to this framework is explained in Chapter 4.1 of the present document.

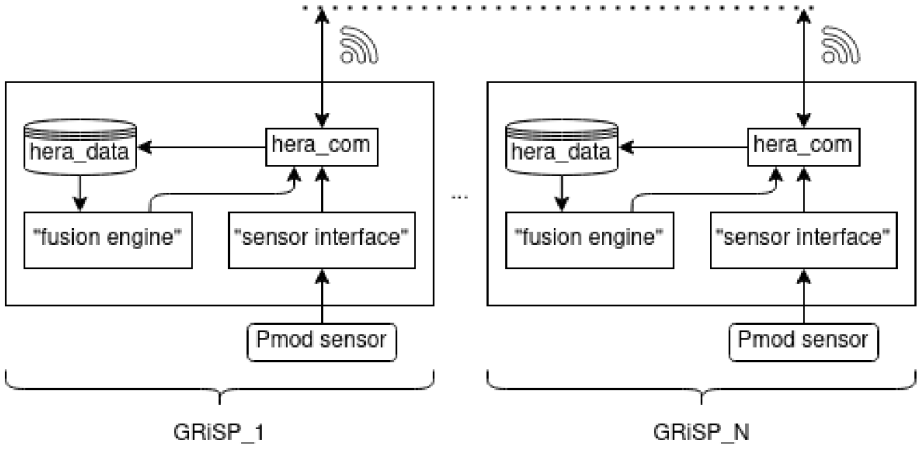


Figure 2.6: Hera Data Flow Chart [17]

2.2.2 The NumErl NIF

The second master's thesis was authored by Lucas Nils in 2023 [13]. The main objective of this thesis was to drastically speed up matrix computations within the GRiSP board by integrating the NumErl NIF to the Hera framework. This optimization is of crucial importance for our project, as our Kalman filters are fed in real time with data from various sensors.

Those filters require linear operations and matrix inversions to be performed almost instantaneously. Without this acceleration, the latency induced by pure Erlang calculations would compromise the responsiveness and accuracy of the state estimation.

Thanks to the work of Lucas Nils, we can now guarantee fast enough matrix processing and maintain sufficient performance for our Kalman filters.

2.2.3 Dynamically balanced robot

In 2024, Cédric Ponsard and François Goens provided the basic hardware and software utilized and expanded upon in our own work. [6]

The central objective of this master's thesis was to control the behaviour of an intrinsically unstable system using a GRiSP2 board. To this end, the authors set up a real-time dynamic control loop specifically designed to stabilize a self-balancing two-wheeled robot.

Initially, they integrated several inertial sensors, all packaged as a single unit in the now retired Pmod Nav. Those sensors measurements, naturally noisy, were filtered using a Kalman filter implemented by Cédric and Francois. By optimally merging the different data sources, this filter considerably improved the accuracy of the angle and speed estimates.

Additionaly, they designed and adjusted a PID controller (integrated in their new Hera version) to continuously correct position deviations and straighten the robot before it tipped over. The combination of the PID and the Kalman filter yielded a robust balance, even in the presence of disturbances or rapid changes of direction.

Their architecture can be divided in a succession of three abstract layers :

1. The deepest part is the **sensor-fusion layer**, using a Kalman filter to deduce the angle of the robot.
2. Then comes the **stability layer**, using this angle stabilize the robot by counterbalancing the falls.
3. Above these two layers is a **excutive control layer**. This layer uses variations in the target angle and the power supplied to the robot's stepper motors to control the robot. It also contains controls for the robot's central motor. This motor is responsible for lifting the robot and returning it to an upright position.

This architecture demonstrated a remarkable stability, resulting in a reliable device remotely controllable. This initial starting point proved to be perfect for us to be able to move through our test environment.

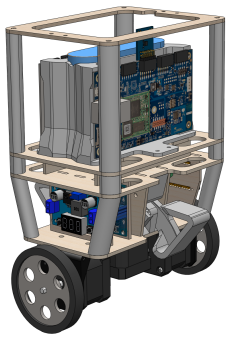


Figure 2.7: Screenshot from SolidWorks of the Balancing Robot [6]

Chapter 3

System Design

Based on the existing architecture, we can say that our work is to add a new abstract layer on top of the three layers already contained within the robot. We will call this layer "*Location awareness*". Conceptually, this means that the robot needs to know its two-dimensional position and its angle of orientation at all times. To enable it to choose which group of sensors it has to listen to, we incorporate a fourth value into this position: the room identifier. Figure 3.1 shows the abstract architecture of the robot system.

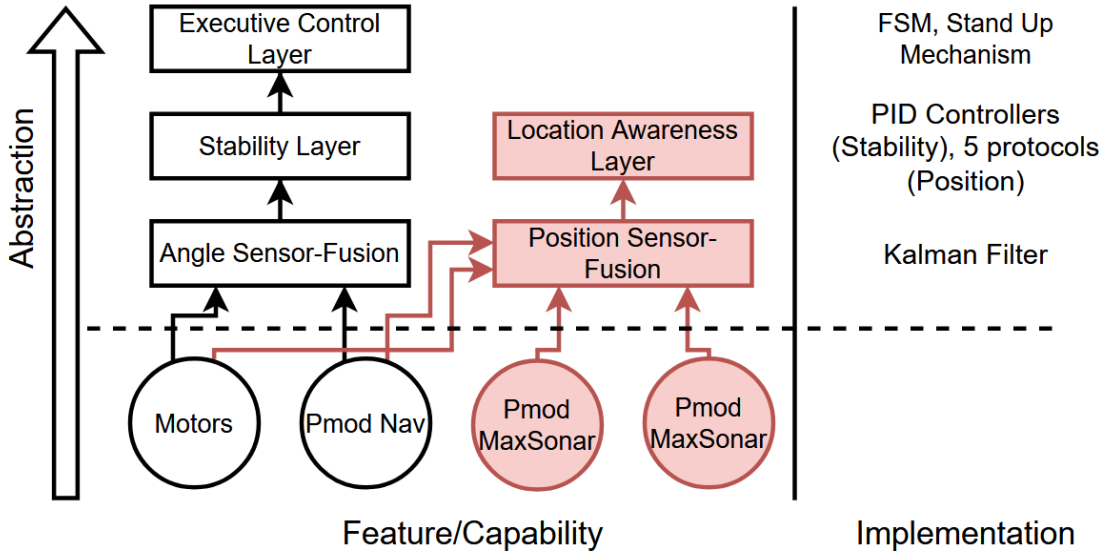


Figure 3.1: Abstraction Layers System Architecture

In black, Figure 3.1 shows the initial abstract layers already integrated to the robot. In red, we show all the logical layers we designed to incorporate the position awareness layer in the architecture.

- At the bottom, the environment interaction layer, it is the layer interfacing with the real world. It contains the different data sources used for sensor-fusion.
- This data is then used to infer the position of the robot.
- All of these previous steps allow for the top-level location awareness for the robot. This location awareness layer is made possible thanks to five protocols we designed. The handover protocol is one of the most important of those protocols. Section 3.2 details their utility.

In order for us to give the robot the knowledge of its location, we need to digitalize the real world knowledge needed about the building. We thus transform it as a graph where each node corresponds either to a room, connected to all adjacent rooms, or to the robot, which is linked both to the room it occupies and to any nearby rooms. Figure 3.2 shows this transition.

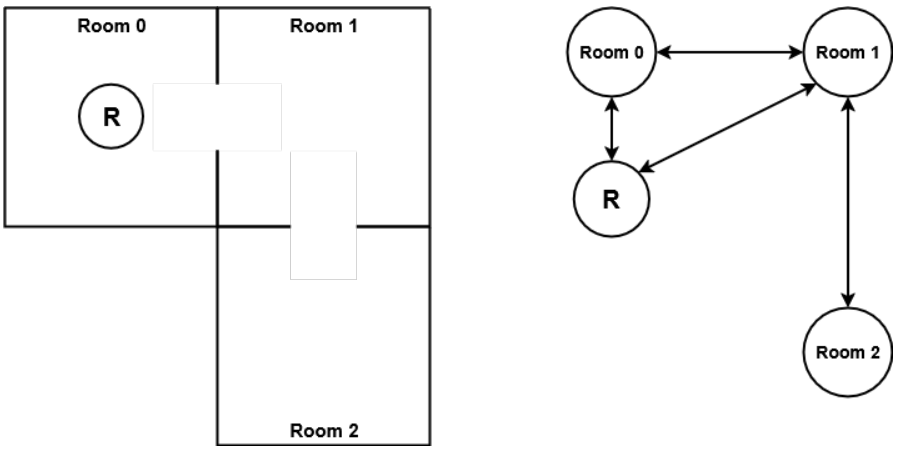


Figure 3.2: Building's Knowledge Graph

In the following sections, we will explain how we designed and built the test environment, the inner protocols used to create those layers and how they are organized as a cohesive system.

3.1 A House Mockup as a Test Environment

Since it was not possible to perform all our tests in a real-size building, we had to scale down the environment issue. As a consequence, we built a miniature mockup of a house to perform the tests required to validate our concepts. As we stayed in the real world, our miniature environment presented us with the same constraints as in a full building. Our solutions should thus remain mostly valid once scaled up to a full-size version in the future.

3.1.1 Rooms

Implications

To model the house, we needed a simplified version of the concept of room. In order to get a convincing model of a house, we built three modular wooden rooms. We also needed the structure to be easy to dismantle so that it could be stored and reassembled easily.

Each room is a parallelepiped with a square base of dimensions 1.14 [m] X 1.14 [m] X 0.61 [m] (Length, Width, Height). We chose those dimensions because they were the most easy to build with the largest wooden planks we found. This brings our mockup to be approximately a 1:8 or 1:9 model of a real room.

Due to its regular shape, our room induced a lot of strong acoustic resonance modes. The reflectiveness of the plywood used as walls and ground also induces very pronounced reflections. Added to the complete emptiness of the rooms (except for the robot), this makes our test environment close to a worst case scenario for sonar usage.

To mitigate the impact of those reflections, two options were open to us, we could either modify our rooms, or filter out those reverberations in software. After a bit of trial and error, we decided to do a bit of both. The argumentation behind these choices are explained in Chapter 5.1.

Concerning the rooms modification, we decided to add acoustic foam to the walls of our rooms, therefore reducing reflectiveness as much as possible. This single modification had a major impact on the precision of the distance measurements taken by the sonars. This impact is studied in Chapter 5.1.1. We detail the other software solutions enforced to tackle these issues in the discussion about the implementations of the room agents in Chapter 4.2.

Construction

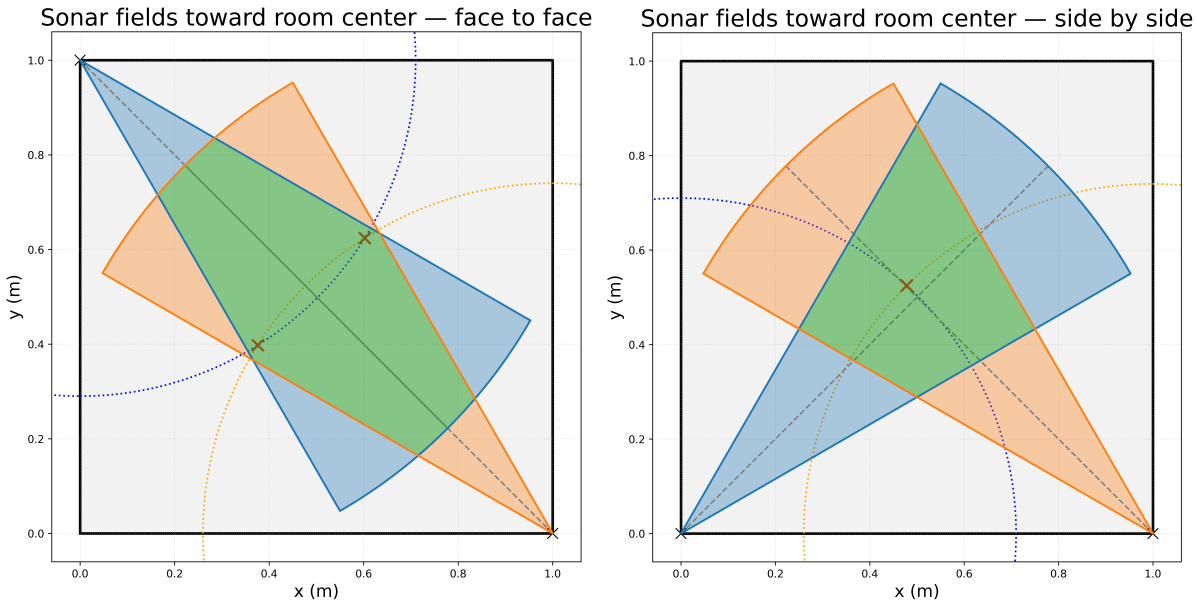
The modular rooms were designed to be as simple to build as possible and to take up as little space as possible. As can be seen in Appendix A.4a and A.4c, we simply put three supports in the four corners of the rooms to be able to slide the wall panels into place. To consolidate the rooms, and create a more precise square floor, we added some small 3D printed supports in each corner.

3.1.2 Sonars

Impact of the position of sonars

Deciding where to position the pair of sonars in the rooms was an important decision. The aim was to maximize the area covered by both sensors. Based on this criterion, we considered several solutions and we calculated the surface area covered by each one according to their positions to find the best possible configuration.

Figure 3.4 was built to illustrate our finding, and thus support our decision:



(a) Intersection Area in the Room = $0.218526[m^2]$

(b) Intersection Area in the room = $0.157338[m^2]$

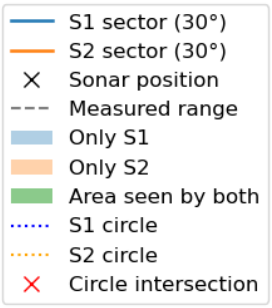


Figure 3.4: Area Comparison Depending on the Sonar Position

As can be seen, the surface area covered by both sonars is greater when they are placed facing each other than when placed on the same wall. On the other hand, when the sensors are placed on opposite corners, two intersections are possible inside the room. This creates an ambiguity in the determination of the right position of the tracked object. Choosing to put the sensors on opposite walls would thus add complexity to the calculation of the object location in the room.

Furthermore, the increase in area is not significant enough to justify this increase in complexity. We thus decided to place our sonars side by side in each room of our test environment.

Sonar hardware

We wanted our sonar sensor to be iterable and easy to build. To achieve this, we decided to split its casing into three different part, all 3D printed. Those parts are presented in Appendix A.4

The first part is a stand to hang the GRiSP to. It also contains a slot for a small lithium battery and a buck converter to power it. The second part is a corner adapter, allowing the GRiSP stand to be placed in the corners of the rooms. The previously mentioned two parts remained the same throughout the tests. The last part was a angle adapter that allowed us to choose the inclination of the sonars relative to our test area. This last part was changed two to three times to try to find the best angle to maximize the ability of the sonars to see in the rooms.

3.2 A House as a Multi-Agent System

The system we created is composed of several connected components, also named agents. Each of them has its unique role and communicates with the others in different ways and with different goals. Before diving deeper inside the implementation of each agent, we explain the structure of our system, the main objective of every agent and the protocols used throughout the system.

Our system can be divided into three separate agents :

1. **The user controller**: executed on a computer, the controller allows the end user to control the robot, define the configuration of the house and monitor the position of the robot in real time.

2. **The robot**: the robot is the central target of the system. It gathers information coming from the rooms in order to deduce its position and broadcast it to the whole system.

3. **The rooms** : The rooms are a more conceptual type of agent. We can subdivise the room in two parts : the physical conception and the of GRiSP sensors that gather real time data in that concerned space. As explained in Section 3.1, the sensors are mounted in the corners of the rooms. Grouped in pairs, they provide the robot with distance measurements coming from their sonars. Each pair contains one master sensor and one slave sensor. The master sensor is in charge of the clock.

The sole multi-agent system's role is to feed the robot with the data it needs to do its sensor fusion. Calling back to the architecture defined at Figure 3.1, it gives all the environment data feeding in the robots position sensor-fusion layer.

3.2.1 Protocol organisation

Multiple protocols have been implemented to provide the robot with the correct amount of data in the most reliable way possible. We can divide those protocols into three categories by zone of influence. :

1. The **general protocols** that involve the whole system. This category comprises the *discovery* and the *configuration* protocols. Both protocols are currently only used once at startup. However, nothing prevents them from being subsequently integrated into the system's operations. This would allow agents to enter and exit the system without issue.

2. The **neighbouring protocols** deal, as their name suggest with the data transfer in the neighbourhood of each room. This category of protocol is composed of the *handover*, *routing* and *propagation protocols*. Note that the routing and propagation protocols are extensions and are not required for the collection of sensor-fusion data.

3. **Local protocols** work inside each room to gather the required data. There are two protocols in this category, namely the *room assembly protocol* and the *sensor-fusion data collection protocol*. The latter is, of course, the main one.

3.2.2 General protocols

Discovery/Configuration protocol

The discovery protocol handles the initial booting of the system. It involves the rooms and the robot and is handled the exact same way by both types of agents. When powered up, the robot agent also spawns the WiFi access point used by the whole system. Once connected to that WiFi, the user can launch the controller. That controller will spawn a server that broadcasts a "*ping*" message on the network every 3 seconds.

When a device manages to connect to the WiFi and receives one of those "*ping*" messages, it will send a "*hello, DeviceName*" to the server. The device name is either "*robot*" or "*sensor_X*" with X being an ID set by the GRiSP jumpers. The ID is the binary addition of the jumpers value, allowing a maximum of 32 sensors in the current state of the system. We decided to use real names instead of the GRiSP serial number given by the *node()* function to be able to quickly and easily identify a GRiSP from the others. Upon reception of this *Hello, DeviceName* message, the server sends a "*ack, hello*" acknowledgment message to the corresponding device.

Then, the configuration phase begins. In this phase, the user can define the simulated house configuration, the position of all sensors, and the initial position of the robot. This allows the system to then send the relevant information to the chosen devices. As we use a small mockup, it enables us to force the creation of an incomplete connection graph between all devicse, which is much more similar to a real-life situation in a home. The predefined configuration data are thus sent to the specified devices. To prevent packet loss from destroying the entire system, each configuration message is resent until it has been acknowledged.

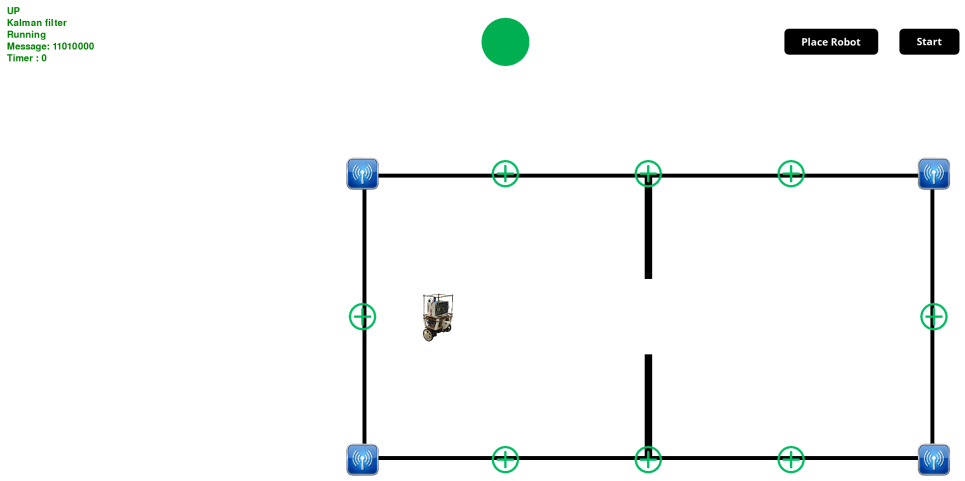


Figure 3.5: Example of a Two Rooms Configuration in the Controller

The configuration, schematized on Figure 3.6 comprises different messages, all answered with an acknowledgment message coming from the receiving devices. The server waits for all known devices to acknowledge a packet before advancing to the next configuration message.

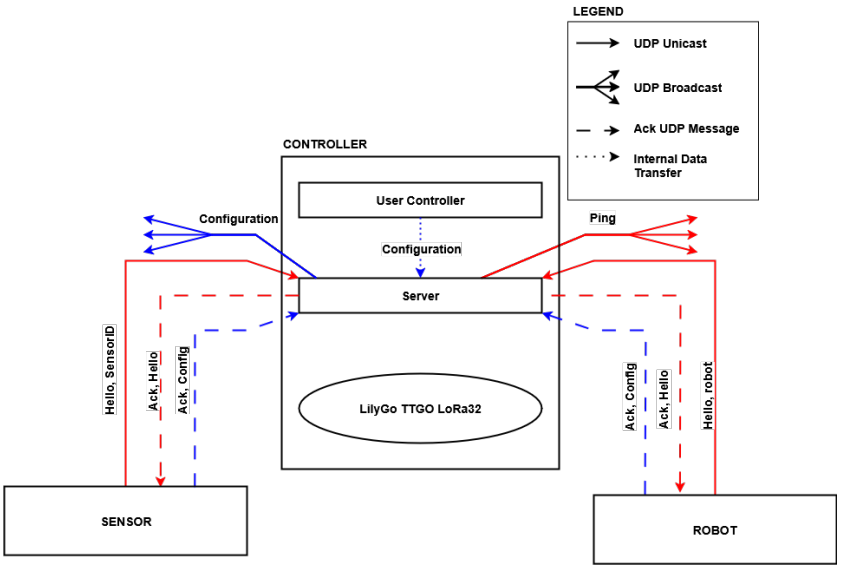


Figure 3.6: Configuration Communication

The configuration is composed of a succession of messages of different types :

- **{Add_device, Name, Ip, Port}**: Adds a device (sensor or robot) to the list of devices reachable. We implemented this message to be able to simulate a discontinued system, even in our small scale simulation environment.
- **{Pos, Name, X, Y, Height, Angle, Room}**: Defines the 3D position of each sensor in the system (X, Y, Height). This is one of the most important pieces of information in the entire configuration, as it is used for multiple applications such as determining the position of the robot, computing the ground distance or pairing the sensors in a same room.
- **{Add_Link, Name, Ip, Port}**: Adds a device to the list of devices used for information propagation between rooms. (see Section 3.2.5 for more information).
- **{Room_info, BLx, BLy, TRx, TRy}**: Defines the limits of a room. As we assume that rooms are rectangles, we only define them using the x and y coordinates of two of their corners (bottom left corner (BL) and top right corner (TR)).
- **{Init_pos, X, Y, Angle, Room}**: Defines the initial position of the robot.

Once the configuration is complete, the server sends a "*Start*" message. This message is critical because it marks the start of measurements. Since "*Start*" is a command and not a data packet, losing that message can cause unexpected behaviour in the system. To prevent those packet losses from occurring, we send the start message four times, adding some redundancy.

3.2.3 Local protocols

Room assembly protocol

During the first seconds after the "*Start*" message is received, the room agents will assemble. The room assembly protocol enables paired sensors to identify each other, elect Master and Slave roles via a handshake with random numbers, allowing them to then organize their sonar measurement to avoid interferences. This process is illustrated on Figure 3.7.

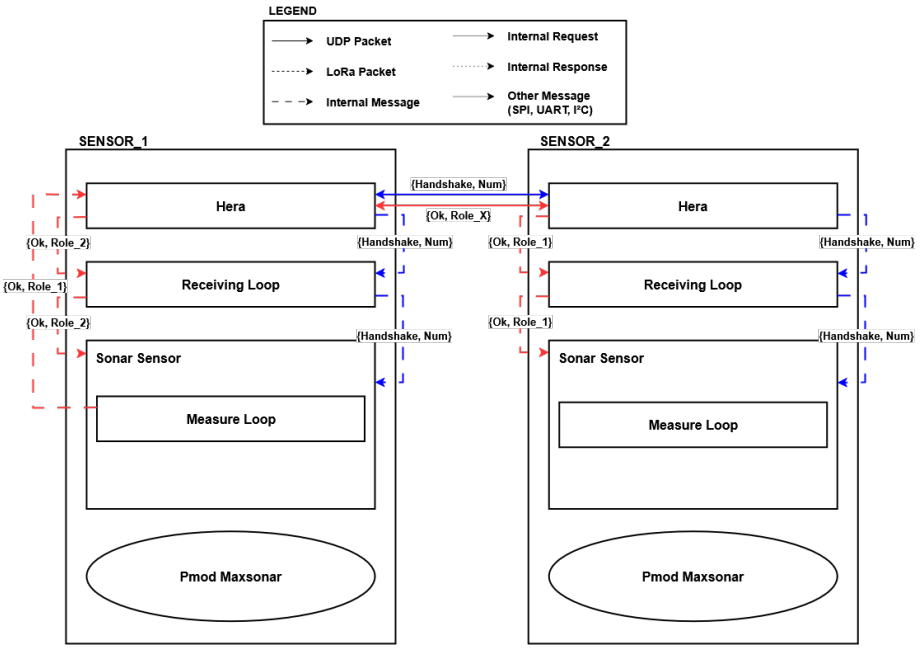


Figure 3.7: Sonar Handshake Protocol

First, using the information gathered by the configuration, each sensor deduces the name of the other sensor in the room. Once found, they will send a "*handshake*" message to their partner together with a random number. The sensor with the highest number will become the Master, the other one becoming the Slave. In the unlikely but possible case where both sensors found the same random number, the process restarts. Once their roles are determined, both sensors send a "*ok, Role*" message to the other. The Master then becomes responsible for the synchronization of the measurements, thus reducing the amount of interferences between the two sonars. See Appendix C for full configuration/room assembly logs instances.

Sensor-fusion data collection protocol

Now that everything is setup, we enter the real operational phase, where sonars take measures and where the robot determines its position. First, we explain what happens in a room while the robot is in its jurisdiction.

The Clock process, spawned only in the master sensor of each room agent, only allows its room sensors to measure if the robot is inside the room or within a predefined margin of the room. Once in that area, the clock uses a logic similar to a time multiplexing protocol to give the right to the sensors to measure.

In time multiplexing protocols, the time is divided in "timeslots" (time frames of a fixed duration) and devices can only transmit data during their assigned timeslot. In our case, and as represented in Figure 3.8, the clock allows the master sonar to measure the distance during the first 50ms timeslot of each 300ms frames, and the slave to measure in the third timeslot, thus between the 150 and 200ms mark.

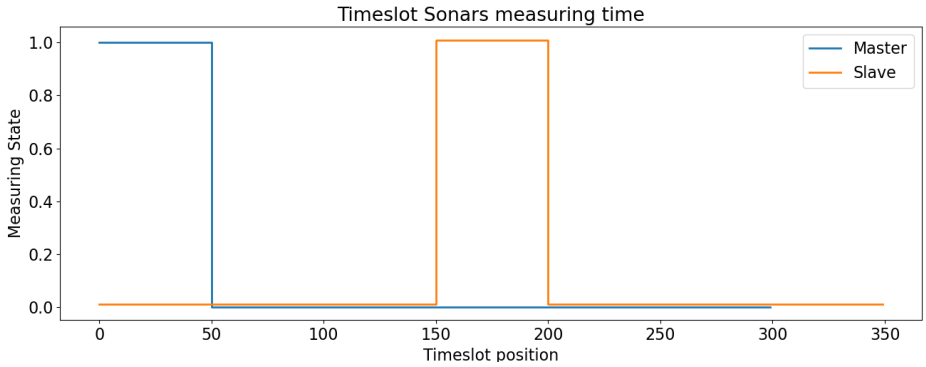


Figure 3.8: Timeslot Measure Times

As shown in Figure 3.9, the clock process, periodically sends a "*Tick*" message, either to its own measuring loop, or to the other sensor via a UDP message. When a sonar loop receives that message, it can get one measure from its sonar. Once the measure is retrieved and filtered, Hera propagates it to all the known devices in the system.

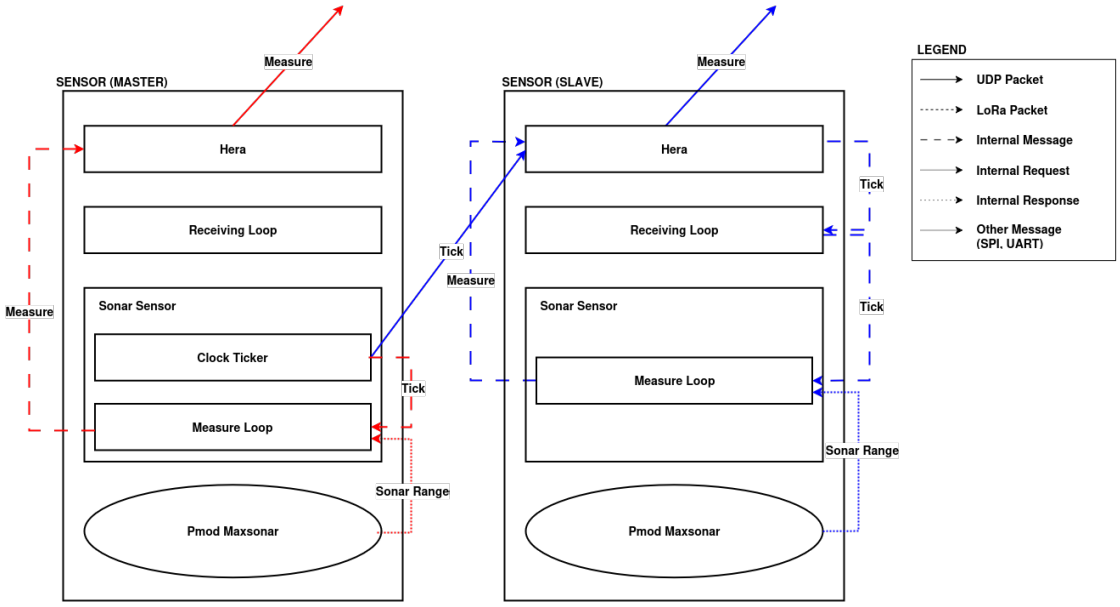


Figure 3.9: Sonar Measure Logic

The internal process of the robot is a bit more complex. Two Kalman filters are at work simultaneously. Figure 3.10 illustrates these processes.

The first filter is used for the stability loop. It uses the data coming from the Pmod Nav's accelerometer and the commands received by the LilyGo to determine the pitch angle of the robot. Although we refactored the code, all the logic used in that Kalman filter is inherited from Cédric Ponsard and François Goens's work. [6]

The second filter is the one we develop for this master's thesis. It is in charge of estimating the position and yaw angle of the robot based on the data coming from the sonars and the data retrieved from the Pmod Nav accelerometer and gyroscope. We can divide the actions of the Kalman filter in three different parts:

1. Sensor choice: based on its last position known, the robot can determine which sensor data to use.

2. Kalman resolution: Based on the pair of sensors deduced in the previous step, the robot can now gather the data and resolve the needed computation. It is Important to note that the robot uses the data from the sonars only if new data is available. If no new measurement is available, the Kalman will use its inertial data to predict its new position, thus using its prediction as estimate.

3. Room inference: Now that the robot has a new estimate of its position in the building, it can deduce in which room it stands.

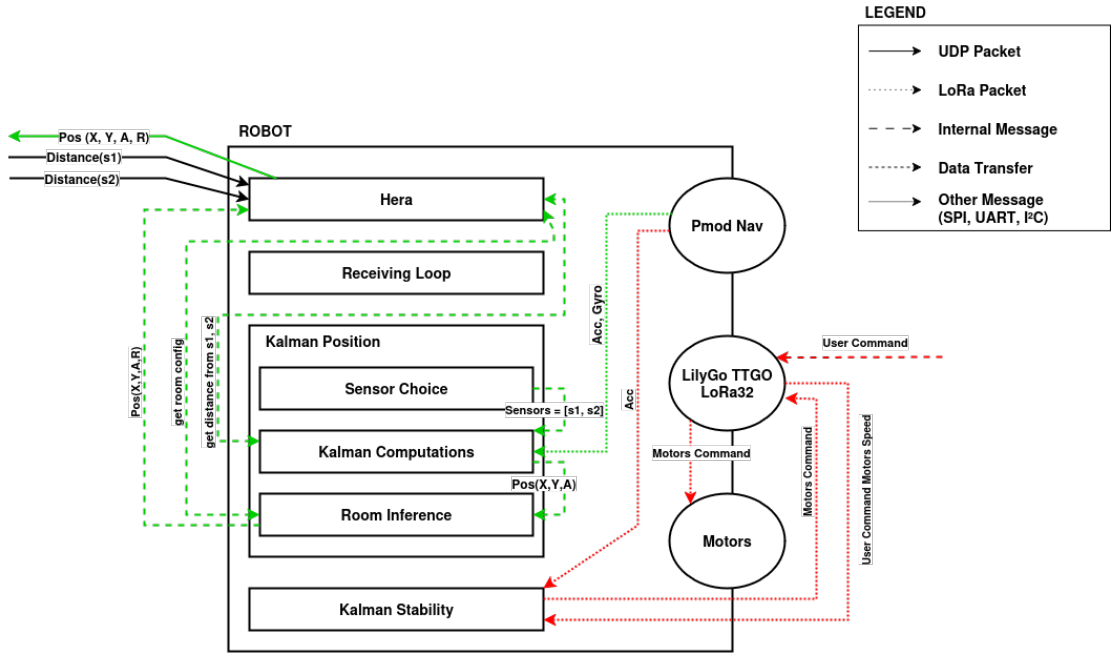


Figure 3.10: Inner Robot Kalman Resolutions

3.2.4 Neighbouring rooms protocols

Handover protocol

The handover protocol handles the case where the robot changes from one room to another. The robot needs to know who to listen to and the room needs to know that the robot is under its responsibility.

To determine which room to listen to, the robot retrieves, before each iteration of its position Kalman filter, the sensors that are in the same room as it. Thanks to the configuration information received at startup and the estimated position found in the last iteration, the robot knows in which room it is.

In order to prevent big interferences in our small mockup, and to prevent the congestion of message at the small ESP32 WiFi access point, we decided to allow the rooms to take measurements only if the robot is within a predefined range of them. We imposed this range to prevent a delay between the entry of the robot in a room and the first reception of sonar data by the robot.

In this view, the clock always checks the robot's position before sending the "*Tick*" message to its room's sensors. The implementation of this logic is explained in Section 4.2.4.

Routing protocol

In a real house, it is very unlikely that all rooms always have access to all the information available in the system. To get as close as possible to a real-world situation, we decided to hypothesize that every room only knows the information coming from the neighbouring ones. Moreover, the robot only communicates with the current room and the neighbouring ones.

To implement this routing, the controller will send each room the configuration information of the rooms adjacent to it.

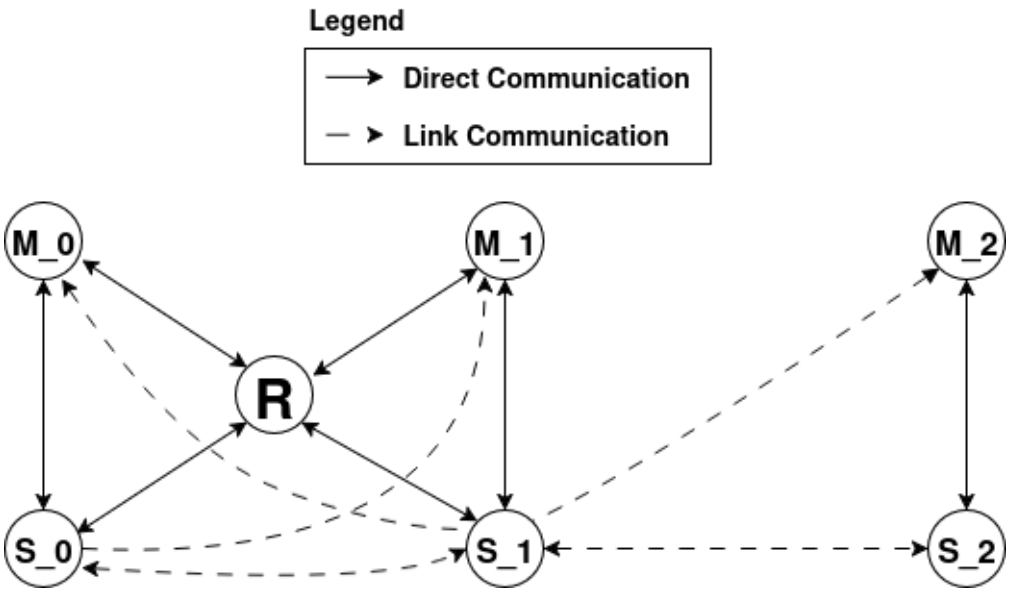


Figure 3.11: Routing Graph in a Three Room System

In the current implementation, this routing graph is artificially simulated in software. Since our system uses UDP over WiFi, all devices communicate through a WiFi access point, which then routes packets to the correct device based on its IP address. In that kind of network, any device can reach any other one within the network, so simply extending the WiFi network using extenders would allow all room agents to communicate with every other one directly.

The purpose of this propagation protocol is therefore to show that it can be implemented and to prepare the system for alternative communication means such as peer-to-peer protocols (e.g LoRa or Bluetooth).

3.2.5 Propagation protocol

The propagation protocol we implemented is quite simple. It uses two different groups of communication. The first one is the group communication provided by the hera_com module. It allows sensors in the same room to communicate with each other and with the robot. We called it direct communication on Figure 3.11. The second group is the "link" communication. We chose to use the slave sensor in each room to serve as link with its adjacent rooms. When this device receives a message that must be forwarded, it sends it to all the links it knows (the adjacent room agents).

We acknowledge the fact that this propagation protocol is pretty simplistic and isn't a real contribution, the goal is more to show that it can be done on this system.

Chapter 4

Implementation

4.1 Hera Extensions

Before going in depth in the implementation of each of the system's component, we explain what we added to the existing *Hera* functionalities. This will ease the understanding of the following chapters of this part.

Getting on board with new unknown technologies can be a really hard task. When first trying to learn about the GRiSP technology, something that really helped us getting up to speed was the GitHub GRiSP wikis. Those simple tutorials helped us learn about the basic features of GRiSP and how to develop and test for it. This motivated us to do the same with Hera. We thus created a wiki on the Hera repository. The goal was to show future users how to get started, what are the available features and what API to use for each of the existing Hera module.

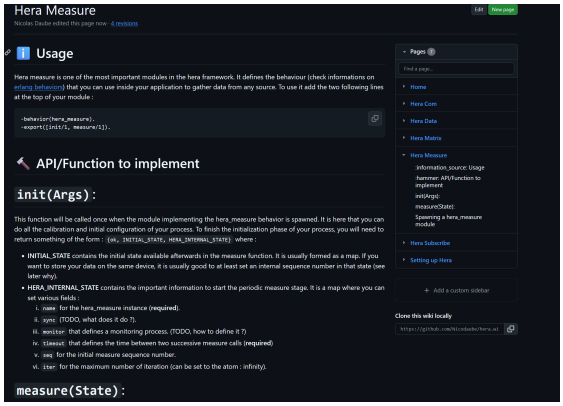


Figure 4.1: Screeshot of the hera_measure Wiki Page

GitHub wikis are a great and easy way to write documentation. Attached to the repository, and written in markdown, they are an easy and understanble way to replace a list of README files.

All the code implementing the functionalities in this Chapter is available via the link below [1] or in Appendix D of this document.

4.1.1 Hera_subscribe

One of our main concern when upgrading Hera was to keep it as modular as possible to accommodate to the widest range of IoT application possible. Hera should thus be able to delegate some application specific messages to the high level GRiSP application. An other thing that Hera didn't yet allow was the notification of event triggering from the inside of Hera to the concrete application modules. By triggers we mean some important events undetectable by the application. As example, a successful connection to the network triggers an event "*connected*".

This is why we implemented the *Hera_subscribe* module. This module allows user crafted processes to "subscribe" to events triggered by Hera or other user implemented processes. For now the triggers are few, but this simple module opens a lot of possibilities for future usages or improvements. Its API is comprised of:

- **hera_subscribe:subscribe/1**: adds the Pid passed as argument to the list of subscribers.
- **hera_subscribe:notify/1**: Sends the message passed as argument to all subscribers.

4.1.2 Handling the loss of connectivity

We also noticed that Hera was capable of determining when the system was successfully connected to the WiFi Access Point, but not when the connection is lost. We thus implemented a periodic check of connectivity. Every 5 seconds, Hera analyses the information available in the address list returned by the function *inet:getifaddrs()*. When no IP address is available in the *wlan* field, this means that the connection is lost. Hera then restart the connection attempts and triggers a *"disconnected"* event.

[1] https://github.com/Nicodaube/hera

4.1.3 Unlink the sending of information

While implementing our system, we noticed that the main bottleneck of every instance of the *hera_measure* behaviour was the broadcast of the measured data at each iteration of the loop. To go around that issue, we decided to decouple the measure from the information sharing process. In order to achieve that goal, each *hera_measure* instance spawns a second process, called *hera_measure_sender*. This second process just receives the information from the *hera_measure* process and shares it through the system (using the *hera_com* module).

4.1.4 Logging

As it embeds a lot of important common IoT processes, it can be very useful for developers to log what happens inside the Hera processes. But when debugging the top level application, this excess of logs can cause some headaches. Since Hera is addded as a rebar dependence, pulling Hera directly from its GitHub repository, it can be very annoying to have to, commit, push, update and redeploy the whole Hera application each time logs are needed. In addition, adding commits changing logging lines simply did not make a lot of sense. We thus made the decision to incorporate a debug mode in Hera that activates or deactivates all logging lines.

Since Hera is started at boot time, before the app is even initialized, we had to find a way to get the argument in an other manner than with a simple function call. The solution we found was to set this value as a application environment variable, set in the *sys.config* file of the GRiSP project. This then spawns Hera with or without debug mode. Instead of the classical *io:format/1/2* method of logging, we implemented the *hera:logg/2* method, checking if debug Mode is activated before logging. This method works much like the traditional method, taking a message and a list of variables as argument.

4.1.5 Separation of the Kalman filter steps

While creating the Kalman filter responsible for the position of our robot, we encountered a problem. For reasons explained in the next chapter, we weren't able to get new distance measurements faster than three to four times per seconds. That frequency being too slow for the precise determination of the position by the Kalman Filter, we had to find a solution.

As explained in Chapter 2.1 of this master thesis, the Kalman filter can be separated in two steps, the estimation and the prediction. Even if those two steps are essential for the best determination of the state of the system, it is not essential that each estimation step is followed by a prediction step.

This is the reason why we decided to separate the prediction and estimation steps in two distinct methods in the *hera_kalman* module. This allows us to only execute the prediction steps when new distance measures are available.

4.1.6 Explicit naming and saving of nodes

Hera originally refers to a GRiSP node by the serial number of the board its running on, available in software via the *node()* function. This simple fact can quickly become a burden. That serial number is only available behind the boards, meaning it is not easy to access it when the boards are mounted. Furthermore, it becomes increasing harder to identify boards when their number increase. This is why we allowed the explicit naming of nodes in Hera. It allowed us to also implement a list of known devices in Hera, allowing unicast communication within the system.

4.2 The Room Agent

The room agents are very important components in this system's architecture. They are composed of a physical room mockup (as explained in Chapter 3.1) and a pair of GRiSP2 equipped with Pmod Maxsonars. Those components, that we simply call sensors, are responsible for the measurement of the distance between them and the robot, if he is in the room.

The simple fact of recording distance seemed, at the beginning of our work, like a formality. The GRiSP ecosystems embeddeds a simple *pmod_maxsonar:get()* function call that allows to take a measure from the sonar. It however soon emerged as a more complex task, with more difficult aspects that we had not foreseen. This section will show you how we organized our sensor code, what problems we encountered and what design decisions we made in order to solve them. All the code used by the sensor is contained in Appendix E and on our open-source GitHub repository[2].

[2]https://github.com/Nicodaube/Grisp_robot/tree/main/sensor

4.2.1 Code structure and organization

The sensor architecture is composed of 3 modules:

- **The main** module (called sensor) is in charge of the initial configuration protocol and of the handling of messages coming from Hera and thus by extension, the outside system.
- **The sonar_measure** module is in charge of taking measurement and filtering them to avoid excessive noise and outliers. It is built as an hera_measure instance.
- **The clock_ticker** module, which is only used in master sensors, sends message to their sonar_measure module and to the slave sensor to determine when they have to make measurements.

As our system grew, it became harder and harder to monitor the state of each GRiSP2 board in the system. To ease that process, we decided to use the LEDs available on the boards to show in which state a board is. Here is a list of the color code used in the sensor:

- **Flashing yellow**: Means that the sensor is booting. Most of the time, this means that it is waiting for Hera to notify that it has reached the WiFi access point and that the network functionalities are now available.
- **Flashing red**: Means that Hera lost the connection to the WiFi access point, and now has restarted the search for the access point.
- **Flashing white**: The sensor has condistancenected to the WiFi access point and now waits for the "*ping*" message coming from the server.
- **Flashing green**: The sensor found the server and waits for the configuration to be sent by the controller.
- **Stay green**: The sensor has started its measurement.
- **Stay blue**: The sensor is waiting for the robot to reach its room before starting the measurements.

4.2.2 The main module

Initialisation

The main module is the first module to spawn in the sensor. It is the one called by the GRiSP2 system after booting. The first thing the sensor does is configuring itself. It sets himself as an Hera subscriber and notifies the GRiSP2 that a Pmod Maxsonar is plugged in its UART port. The initialisation logic can be found at Appendix E.1.1. After that, the sensor will compute its ID.

Discovery/Configuration protocol

Once the initial internal setup is done, the sensor starts its Discovery/Configuration protocol.

The implementation of this protocol is done by using a set of *receive* expressions waiting for messages coming from the hera_subscribe module. They are all built to work one after the others. A first one waits for Hera to achieve connection to the WiFi access point, then one waits for the ping coming from the server, then one waits for the acknowledgment message coming from the server after the hello message was sent.

Once this initial discovery phase is done, the sensor will wait for the configuration to arrive. This is done using a first loop, called *loop_config*. That loop allows the sensor to handle any configuration sent by the server and to save it in memory. We store the data in two different places, depending on whether the data may change over time or not. The persistent_term storage [19] contains all constant data. It allows us to get all fixed values in constant time. The hera_data storage, on the other hand, contains all the data that may change over time.

Room assembly protocol

All configurations are followed by a "*Start*" message coming from the server, signifying the end of the configuration phase. Upon reception of this message, all sensors will start their assembly into pairs to form the room agents, following the configuration just received.

Each sensor will start by deducing who is the other sensor in its room, based on the sensor position determined during the configuration. It is also at this moment that the sensor module will spawn the sonar_measure module, keeping its Pid in persistent memory to be able to contact it. The rest of the room assembly protocol will be explained in Section 4.2.3.

Regular Operation

Once the room assembly protocol has started, the sensor module calls the loop_run receive expression. This is where this module will loop until the system is exited. This loop contains all the operational triggers used during the operational phase of the system. From this time on, the only responsibility of the sensor process is to react to message coming from other agents.

Exit protocol

In the idea of easing the testing phase of our master's thesis, we incorporated an exit function to the system. When the user controller is exited gracefully (shutting down the window), it broadcasts a "*exit*" message, which is used as trigger for all sensors to kill their sonar_measure modules, erase all the data gathered and to go back to the discovery/configuration protocol.

4.2.3 Hera_measure instance : sonar_measure

The sonar_measure module uses the hera_measure behaviour. As a reminder, this behaviour calls the measure function at a predetermined frequency set by the developer, and sends that data through the system to all known devices. This behaviour only needs two functions to start its looping measures : init/1 and measure/1.

The initialisation phase

Upon startup, the behaviour will call the init function. This function is only called once upon process creation. First, the function will want to assemble the room, to know what is the role of this sonar module.

For that, it will call the get_sensor_role function, which checks if the sensor's role has already been defined, in which case it does not restart the handshake protocol with the other sensor. This can happen when the sonar_module crashes for some reason. In that case the Hera supervisor will create a new sonar_measure process, and the process can just extract its role from memory.

If no role has been assigned yet, the sensor will choose a random number and send it packaged in an handshake message. When the equivalent message is received coming from the other sensor, the process will acknowledge the reception and compare the two numbers, the sensor with the highest number getting the role of master. This random number method has been chosen for test purpose, but in the real application, it could be replaced with a more meaningful number, such as a

measure of the signal strength, like the RSSI (Received Signal Strength Indicator). The master will then spawn the clock_ticker process and record its Pid.

The room assembly protocol has now done its job, we now have a fully functional room agent.

The init function of this sonar_measure process will now create an internal state containing a sequence number, the last_measure done (to use in the low pass), a buffer used for the Hampel filter and a buffer for the final smoothing of the measures. As far as the hera_measure parameters are concerned, we tell Hera to use this loop infinitly and gave our loop the lowest timeout possible so we could trigger the measure ourselves using the clock.

The measuring phase

Once this init function is done, we enter the regular measurement phase, where Hera calls the measure function each time the previous one finished.

The measure function always starts with a receive expression, where the function waits for the clock to send it a *Tick* message, granting it the possibility to take a measure. Upon reception the sonar_measure will gather the data coming from the Pmod Maxsonar and make the conversion from inches to centimeters.

The measure is then passed through three different filters in chain. We use the three filters to be able to use softer parameters. First, the value is passed through a low pass filter with an alpha of 0.2, which means that the new value is taken as 20% of the last measure + 80% of the new one. Second, Hampel filter is applied, with a buffer of size 7 and a n_σ of 2.0. The role of this filter is solely to exclude absurd spikes forming in the measures. It has almost no effect on the small measures. Lastly, that value is passed through a soft smoothing filter, which takes the median value of its buffer. The impact of each of these filters is studied in Chapter 5.1.2.

After that data is filtered, we use the result to infer the distance between the corner where the sonar is located and the robot. Figure 4.2 visually shows what is computed. For that we simply use the Pythagorean equation for right triangle.

On the following graph and equation, the following notations are applied :

- H_S is the height at which the sensor sits.
- H_R is the size of the robot.
- D_S is the distance measured by the sonar.
- D_G is the distance on the ground.

$$D_G = \sqrt{D_S^2 - (H_S - H_R)^2}$$

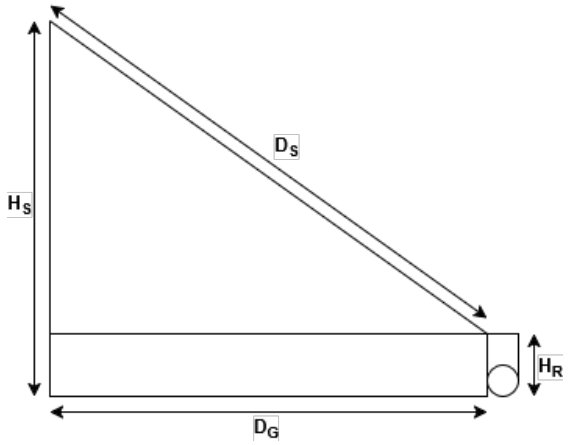


Figure 4.2: Ground Distance Schema

4.2.4 The clock_ticker process

The clock process is only spawned in the master sensor of each room. This is the process responsible for the timing of the measurement. When spawned, the process fetches all the necessary data needed for its operational process. The process can then start its loop function with all the data as argument.

The clock we implemented only ticks if the robot is close to its room. We inserted a macro (*STARTUP_MARGIN*) that determines at which range of the room the sonars will start to measure. This prevents a zone and time where the robot is not in any room, or where it must wait for the new room to start its measure. We chose a margin of 12cm, which represents approximatively 10% of the size of the room.

The second macro that we have set is the *TIMESLOT_SIZE*. This one represents the time between two sonar measures. For a good compromize between frequency of update and errors (that seem to increase with the frequency), we chose a frequency of 1 measure every 300ms (frequency of 3.33Hz).

Figure 4.3 shows the logic with which the clock sends its message. For that, it first computes when the next measure should happen based on a count number and the initial time clock. Then, it determines if the robot is in the current room, or within the margin determined by the STARTUP_MARGIN macro. If the robot is in the room, the robot determines which sonar must measure (still based on that count variable). At last, the robot waits for the next measure mark to happen and sends the message to the right sonar. As you can see on the diagram, we wait at different times for each flow path. We do it so it is done at the very last time. This allows for the clock to account for the computing time of the previous logic and to skip the measure if the mark is already passed.

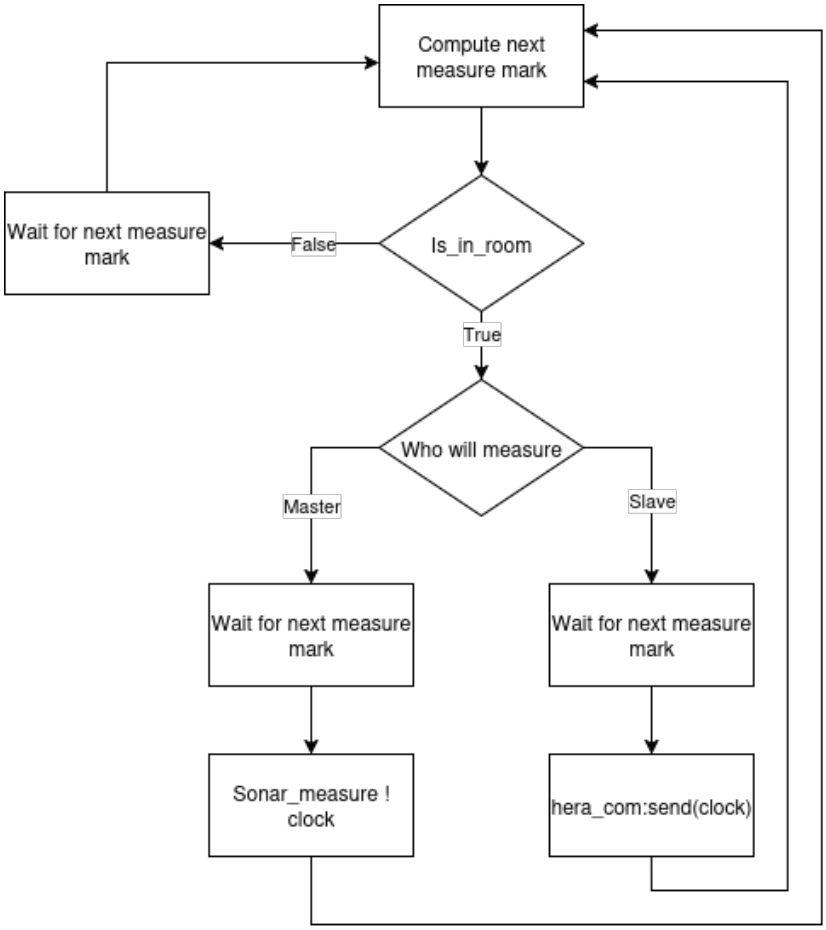


Figure 4.3: Clock Flow Diagram

4.3 The Robot Agent

The robot agent was originally developed by Cédric Ponsard and François Goens [6]. For the purpose of this thesis, we have adapted their foundational design to better suit the objectives of our system.

This mobile robot is equipped with a GRiSP board, a LilyGo microcontroller, DC motors, and various electronic components that allow it to maintain balance and navigate within a test environment. Among these components, we will focus on those most relevant to our implementation and analysis.

The GRiSP ecosystem plays a central role in our setup. It enables the retrieval of position data computed by the Room agent and performs real-time stability control using embedded computations. Additionally, it collects the necessary sensor data to estimate both the position and orientation of the robot.

This chapter presents the components and subsystems that are essential for understanding the functioning of the robot, as well as the modifications and improvements made throughout the project.

4.3.1 Code structure and organization

The sensor architecture is composed of 3 modules:

- **The main** module (called balancing_robot) is in charge of the initial configuration protocol and the launch of the various calculations necessary for the stability and position of the robot. As it is very similar to the sensor module studied in Section 4.2.2, we won't explain it a second time.
- **The stability_layer_module** module is in charge of calculating and controlling the robot's stability and to allow movement.
- **The position_layer_module** module is responsible for calculating the robot's location.

4.3.2 Stability loop and control

This chapter is a simplified explanation of the work of Cédric Ponsard and François Goens [6].

The stability loop implemented in this project uses an Extended Kalman Filter (EKF) based on a partial digital twin of the robot. The predictive model comes directly from the physical equations of the system, which allows to take into account several effects at the same time: linear acceleration from the control input, gravitational acceleration, centripetal acceleration and angular acceleration effects. This approach provides a more accurate estimation of the states compared to the simple model, especially for the angle θ and its rate $\dot{\theta}$, while also using the accelerometer measurements directly in the update step.

The state vector is defined as:

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

and the prediction step of the physical model is:

$$f = \begin{bmatrix} \theta_k + \dot{\theta}_k \Delta t \\ \dot{\theta}_k + \left(\frac{g}{h + J/mh} \sin(\theta) - \frac{u}{h + J/mh} \cos(\theta) \right) \Delta t \end{bmatrix}$$

The measurement model $h(x)$ combines the lateral, angular, centripetal and gravitational acceleration components to match the accelerometer readings. This allows the EKF to estimate the angle and angular velocity while rejecting noise from the sensors.

Once the state is estimated by the EKF, the control is applied using a combination of:

- A PD controller that stabilises the robot by correcting the error between the estimated angle θ and the equilibrium angle θ_{eq}.
- A PI controller that adjusts θ_{eq} dynamically to reach the desired speed, adapting to variations in payload or terrain.

The general control law of the PID/PD controller is:

$$u(t) = K_p\, e(t) + K_i \int e(t)\, dt + K_d \frac{de(t)}{dt}$$

where $e(t)$ is the difference between the reference and the measured value. To prevent sudden changes, a trapezoidal speed profile is used, limiting accelerations and decelerations and improving stability.

In summary, this loop combines the precision of a physics-based EKF with the flexibility of PID control, ensuring that the robot remains stable even in changing conditions.

4.3.3 Kalman filter Implementation

To ensure a good estimate robot's position in our test environment, we implemented two kalman instances, one for the robot's orientation and the other for its position.

Kalman filter for orientation

The explanation in this chapter is based on the work of Hadrien Sonnet [16] and the theses of Victor Verpoten and Sébastien Kalbusch [17].

The first step in solving the robot motion tracking problem is to create an **Attitude and Heading Reference System** (AHRS). This system enables us to transform sensor frame measurements into the global reference frame, making it possible to interpret data on gravitational acceleration, rotational acceleration, and the strength of the magnetic field in a global context.

The Kalman filter for attitude estimation will be used to express the data and states of each sensor in the $\{\text{North}, \text{Top}, \text{East}\}$ reference frame. This guarantees a consistent representation of orientation in our test environment.

Quaternions

Quaternions play an important role in representing the orientation of our robot. Here, we introduce the concept of quaternions [5] and the various notations used.

Quaternions are four-dimensional numbers consisting of a real part and three imaginary parts, representing the orientation of an object. A 3D rotation can be described as a rotation about a certain axis by a certain angle.

A quaternion is expressed as:

$$q = w + x\,i + y\,j + z\,k \tag{4.1}$$

where w, x, y, z are real numbers, and i, j, k are the imaginary units.

Quaternions can be added and multiplied, but multiplication is not commutative. The multiplication rules for the imaginary units are:

$\times$	i	j	k
i	-1	k	$-j$
j	$-k$	-1	i
k	j	$-i$	-1

Table 4.1: Multiplication rules for quaternion base elements

Unit quaternions. Unit quaternions are particularly well suited to represent rotations: they are unambiguous and computationally more efficient than rotation matrices. Rotation matrices often require re-orthogonalization and renormalization due to accumulated numerical errors in iterative computations. Quaternions, once expressed as vectors, can be normalized just like any vector:

$$q = (w \quad x \quad y \quad z)^T \tag{4.2}$$

From quaternion to rotation matrix. A quaternion can be converted to a rotation matrix:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.3}$$

From rotation matrix to quaternion. Given $\mathbf{R}$, the corresponding quaternion is:

$$\mathbf{q} = \frac{1}{2} \begin{pmatrix} \sqrt{1 + r_{11} + r_{22} + r_{33}} \\ \frac{r_{32} - r_{23}}{\sqrt{1 + r_{11} + r_{22} + r_{33}}} \\ \frac{r_{13} - r_{31}}{\sqrt{1 + r_{11} + r_{22} + r_{33}}} \\ \frac{r_{21} - r_{12}}{\sqrt{1 + r_{11} + r_{22} + r_{33}}} \end{pmatrix} \tag{4.4}$$

Rotation of a vector using a quaternion. From a quaternion, the rotation of a vector $\mathbf{p}$ can be expressed as:

$$\mathbf{F}(\mathbf{p}) = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^{-1} \tag{4.5}$$

which expands to:

$$\mathbf{F}(\mathbf{p}) = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{4.6}$$

Prediction model (Gyroscope-based)

To determine the robot's rotation at any time, we continuously update the quaternion using gyroscope data. The angular velocity vector from the gyroscope allows us to approximate the quaternion derivative and predict its next state:

$$\boldsymbol{\omega}(t) \triangleq \frac{d\boldsymbol{\phi}_{\mathcal{L}}(t)}{dt} \approx \frac{\Delta\boldsymbol{\phi}_{\mathcal{L}}}{\Delta t} \tag{4.7}$$

If the perturbation angle is small:

$$\Delta\mathbf{q}_{\mathcal{L}} \approx \begin{bmatrix} 1 \\ \Delta\boldsymbol{\Phi}_{\mathcal{L}}/2 \end{bmatrix} \tag{4.8}$$

The discrete update rule is:

$$\mathbf{q}_{n+1} = \mathbf{q}_n \otimes \left(1 + \frac{1}{2}\boldsymbol{\omega}_c \Delta t\right) \tag{4.9}$$

or equivalently:

$$\mathbf{q}_{n+1} = \left(\mathbb{I} + \frac{1}{2}\Omega(\boldsymbol{\omega})\Delta t\right) \mathbf{q}_n \tag{4.10}$$

where:

$$\Omega(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{bmatrix} \tag{4.11}$$

The quaternion is normalized after each prediction step.

Update model (Accelerometer + Magnetometer)

The measurement update combines accelerometer and magnetometer data to obtain an absolute orientation estimate.

Gravity and magnetic field in the local frame. Let:

$$\boldsymbol{g}_{\text{mes}} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}, \quad \boldsymbol{m}_{\text{mes}} = \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix}$$

After normalization:

$$\boldsymbol{m}_{\perp} = \boldsymbol{m}_{\text{mes}} - (\boldsymbol{m}_{\text{mes}}^{\top} \boldsymbol{g}_{\text{mes}})\, \boldsymbol{g}_{\text{mes}}, \quad \boldsymbol{m}_{\text{hor}} = \frac{\boldsymbol{m}_{\perp}}{\|\boldsymbol{m}_{\perp}\|}$$

Constructing the measured attitude. From $\boldsymbol{g}_{\text{mes}}$ and $\boldsymbol{m}_{\text{hor}}$:

$$\boldsymbol{T} = -\boldsymbol{g}_{\text{mes}}, \quad \boldsymbol{N} = \frac{\boldsymbol{m}_{\text{hor}}}{\|\boldsymbol{m}_{\text{hor}}\|}, \quad \boldsymbol{E} = \boldsymbol{T} \times \boldsymbol{N}$$

The rotation matrix is:

$$\mathbf{R}_{\text{mes}} = \begin{bmatrix} \boldsymbol{N} & \boldsymbol{T} & \boldsymbol{E} \end{bmatrix}$$

Converted to quaternion via:

$$t = r_{11} + r_{22} + r_{33}$$

If $t > 0$:

$$\begin{cases} w = \frac{1}{2}\sqrt{1+t} \\ x = \frac{r_{32}-r_{23}}{4w} \\ y = \frac{r_{13}-r_{31}}{4w} \\ z = \frac{r_{21}-r_{12}}{4w} \end{cases}$$

Finally, normalize $\boldsymbol{q}_{\text{mes}}$.

Measurement model for the Kalman filter.

$$\boldsymbol{z}_k = \boldsymbol{q}_{\text{mes}} = h(\boldsymbol{x}_k) + \boldsymbol{v}_k, \quad \boldsymbol{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$$

The gyroscope prediction is corrected using the accelerometer+magnetometer measurement to cancel drift.
In the Appendix B, the figure *a* that helps to better visualize the situation.

Kalman filter model

Now that we have completed the mathematical descriptions of the various equations used to construct our orientation Kalman filter, here are the different matrices used in our filter. In this formulation, the quaternion state is $\mathbf{X} = (q_0, q_1, q_2, q_3)^\top$, where $q_0 = w$, $q_1 = x$, $q_2 = y$, and $q_3 = z$. The angular velocity components $(\omega_x, \omega_y, \omega_z)$ are obtained from the gyroscope and expressed in rad/s.

$$\mathbf{F} = \begin{bmatrix} 1 & -\frac{\omega_x \Delta t}{2} & -\frac{\omega_y \Delta t}{2} & -\frac{\omega_z \Delta t}{2} \\ \frac{\omega_x \Delta t}{2} & 1 & \frac{\omega_z \Delta t}{2} & -\frac{\omega_y \Delta t}{2} \\ \frac{\omega_y \Delta t}{2} & -\frac{\omega_z \Delta t}{2} & 1 & \frac{\omega_x \Delta t}{2} \\ \frac{\omega_z \Delta t}{2} & \frac{\omega_y \Delta t}{2} & -\frac{\omega_x \Delta t}{2} & 1 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{Q} = \begin{bmatrix} 0.0002 & 0 & 0 & 0 \\ 0 & 0.0002 & 0 & 0 \\ 0 & 0 & 0.0002 & 0 \\ 0 & 0 & 0 & 0.0002 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 0.15 & 0 & 0 & 0 \\ 0 & 0.15 & 0 & 0 \\ 0 & 0 & 0.15 & 0 \\ 0 & 0 & 0 & 0.15 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{7.13}$$

The process noise covariance $\mathbf{Q}$ and the measurement noise covariance $\mathbf{R}$ are tuned according to the estimated noise characteristics of the gyroscope, accelerometer, and magnetometer. After each prediction and update step, the quaternion state $\mathbf{X}$ is normalized to maintain unit norm.

Kalman filter for position

The other important part of localizing our robot in the environment is determining its exact position.
For this, we implemented an Extended Kalman Filter (EKF), as our system is nonlinear and based on noisy measurements.

Construction of the model (Prediction)

We chose a simple constant-velocity model for the position EKF:

$$X_{k+1} = X_k + V_{\text{mes},k+1} \cdot \cos(\Theta_{\text{mes},k})\, \Delta t \tag{4.12}$$

$$Y_{k+1} = Y_k + V_{\text{mes},k+1} \cdot \sin(\Theta_{\text{mes},k})\, \Delta t \tag{4.13}$$

Here, V_{mes} is obtained from wheel odometry, and Θ_{mes} from the orientation Kalman filter.

Linear acceleration is not explicitly estimated as a separate state, but modeled through the process noise Q. This approximation is valid because:

- We have no direct linear acceleration measurement (it would require multiplying accelerometer data by the orientation, introducing additional error).
- The sampling rate is relatively low, limiting velocity variations between two iterations.
- Unmodeled acceleration is captured by zero-mean Gaussian noise $\mathcal{N}(0, \sigma_a^2)$, calibrated experimentally to balance precision and stability.

The state vector is:

$$\mathbf{x}_k = \begin{pmatrix} X \\ Y \end{pmatrix}$$

The state transition function and Jacobian are:

$$f(\mathbf{x}_k) = \begin{pmatrix} X_k + V_{\text{mes}} \cos(\Theta_{\text{mes},k})\, \Delta t \\ Y_k + V_{\text{mes}} \sin(\Theta_{\text{mes},k})\, \Delta t \end{pmatrix}, \quad F_k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4.14}$$

Measurement model (Update)

The position is determined using data collected by the sonar, which measures the shortest distance between itself and an obstacle. From these distance measurements, we can perform a straightforward calculation to determine the object's X and Y coordinates within the room. The sonar is positioned in such a way that, when calculating the intersection of circles, only one unique pair of coordinates can exist

in our environment. Figure 3.434 provides a visual overview. After this calculation, we can therefore determine the coordinates measured by the sonars:

$$\mathbf{z}_k = \begin{pmatrix} X_{\text{mes},k} \\ Y_{\text{mes},k} \end{pmatrix}$$

The observation model is:

$$h(\mathbf{x}_k) = \begin{pmatrix} X \\ Y \end{pmatrix}$$

with Jacobian:

$$H_k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Kalman filter model

The process noise covariance $\mathbf{Q}$ and measurement noise covariance $\mathbf{R}$ are set based on the sensor specifications and fine-tuned experimentally:

$$R = \begin{pmatrix} 0.0075 & 0 \\ 0 & 0.0075 \end{pmatrix}, \quad Q = \begin{pmatrix} 0.0002 & 0 \\ 0 & 0.0002 \end{pmatrix}$$

In Appendix B, Figure *b* helps to better visualize the situation.

4.4 The User Controller

The user controller is an agent in charge of several key features:

- The creation of the connection graph between the different agents.
- The creation of the configuration describing the assembly of room agents, with their child sensors, their size and location, and the initial position of the robot.
- The monitoring of the real time position of the robot.

It is composed of a controller and a server module. The controller is responsible for the UI and the interpretation of user inputs while the server is responsible for all the communication between the controller and the system. All the code we will talk about in this chapter is available at Appendix G

4.4.1 The controller module

The controller is built as a Pygame [25] / Pygame GUI [23] application. It also uses the serial library for communication with the LilyGo. The original controller was built during the previous balancing robot master's thesis [6]. Even though Pygame may not be the technology of choice to build this type of software, we decided to extend the original controller to accelerate development, as it was not the main concern of our project.

Launching the controller

To start the controller, we decided to use a bash script. This script fetches the IP address of the computer, thus preventing the user from having to change its IP in the code manually everytime the DHCP changes it.

General usage

When the controller is started, the user is presented with the UI that is used for the whole controller. Figure 4.4 shows all the actions initially available on the controller.

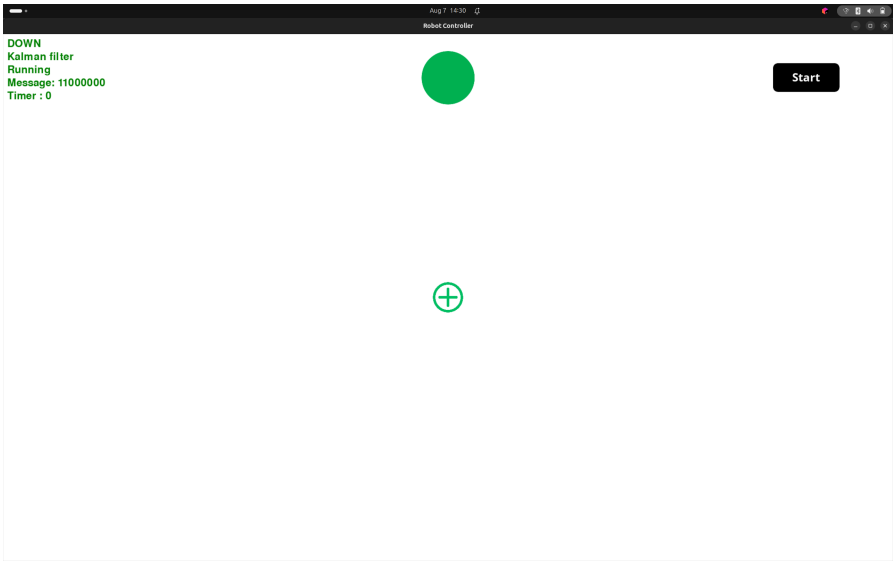


Figure 4.4: Initial Controller

On Figure 4.4, we can see:

- A text box containing important data concerning the current state of the system. This was already part of the original controller developed for the previous master's thesis [6].

- The green dot shows the current control sent to the robot; It can turn into a left, right, forward or backward arrow, idle with a green dot or stop with a stop sign. This is also inherited from the last master's thesis.
- The plus sign allows you to create a first room. When clicked, a pop-up appears asking you to fill in the size of the room (in two dimensions).
- The start button notifies the controller that the configuration is ready and that it can transmit it to the system. When clicked, all the data is transmitted through the server to all the agents in the system.

When a first room is created, two new options appear (see Figure 4.5). The new plus signs allow the user to add rooms to the sides or sensors to the corners. The *Place Robot* button shows a pop-up asking the initial X, Y and angle position of the robot. Once the system is started, the image of the robot (see Figure 3.5) moves and turns following the estimated state given by the Kalman Filter.

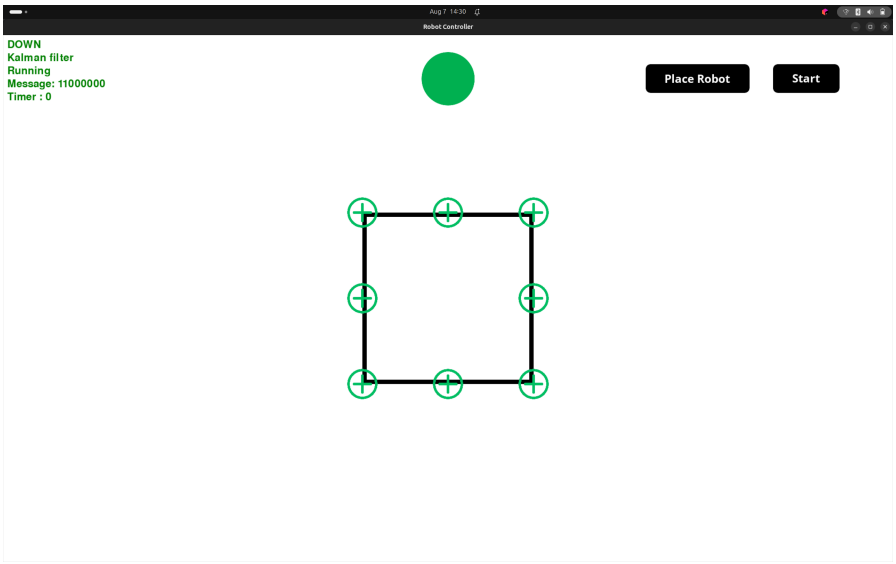


Figure 4.5: Controller Containing One Room

Chapter 5

Evaluation

5.1 Sonar protocol tests

When we started measuring the distance in our rooms, we quickly noticed a drop in measurement accuracy. Moreover, we also noticed that theses errors did not occur when the sonars were used outside the wooden rooms.

The fact is that these errors caused huge drops in performance in our Kalman filter, as the deduced position of the robot was completely off when encountering a spike. To reduce the impact of these errors, we implemented two different counter-measures. First, we paneled the test environment with absorbing material, based on the hypothesis that the errors resulted from echoing sonar impulses. Second, we implemented some filters on the measurements given by the sonars. We decided to rely more on the sonar filters, because if they could perform well in a difficult environment, they should perform even better in a more forgiving one.

It was not possible to rely solely on the filters though. To only use filters, we had to give them very strict parameters in order for them to filter out the outliers, thus also filtering out some valuable data. This is why we also implemented some modifications on the rooms structure.

5.1.1 The influence of the absorbing material

For those experiments, we initially decided to use a timeslot of 200ms, meaning that each sensor takes a measurement every 100ms. Following the specifications given by the Pmod Maxsonar datasheet [21], this creates a no-measure time zone of 50 ms between the end of the measure by a sensor and the start of the other measure by the other sensor. Note that, in the tests done hereafter, the measure

are taken without any filtering of the sonars data.

On the following graphs, we present in different colours the measures coming from the two sensors in a same room. The red vertical lines present the *tick* (beginning of the time slot) sent by the master clock.

To start measuring the impact of the hypothetic multi-path echo in our house mockup, we decided to first use the most basic setup possible, monitoring the measures retrieved by the sonars. Figure A.3 in Appendix A.2 shows that initial setup.

This first experiment allowed us to have a good comparison point, and check wether our hypothesis was correct and that there was indeed an echo forming inside the rooms. Figure 5.1 presents the results retrieved from the sonar without any absorption in the room.

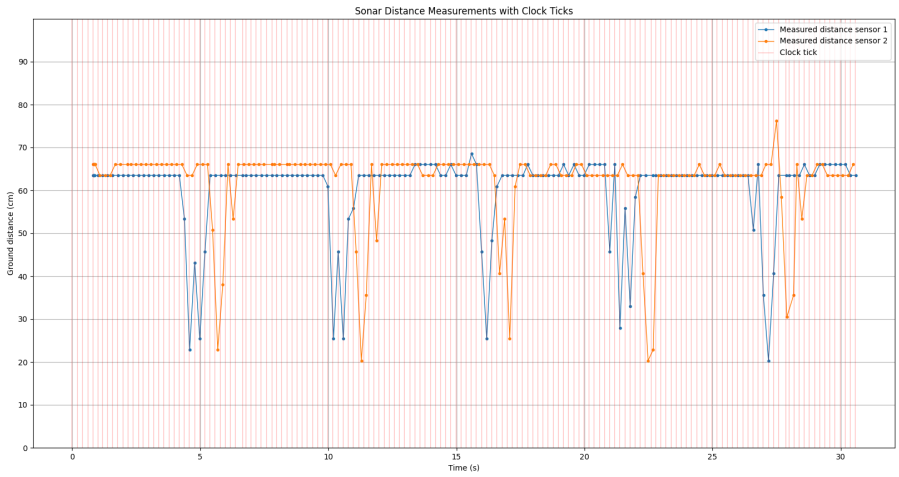


Figure 5.1: Sonar Measure Without Acoustic Foam

When analyzing this graph, we identified two main issues:

1. The most obvious issue is the appearance of these huge peaks approximately every 6 or 7 seconds.
2. The second identified issue are those small errors of around 2 to 3cm that happen randomly. They are less important but can lead to unprecise measurements if occurring too often.

When first encountered these spikes, we blamed the clock, thinking that the measurement frequency must have been too high, causing a cross-talk (sound waves from one sonar read by the other one) between the two sonars. We thus tried to increase the timeslot to 1000ms. Figure 5.2 shows the result of this test.

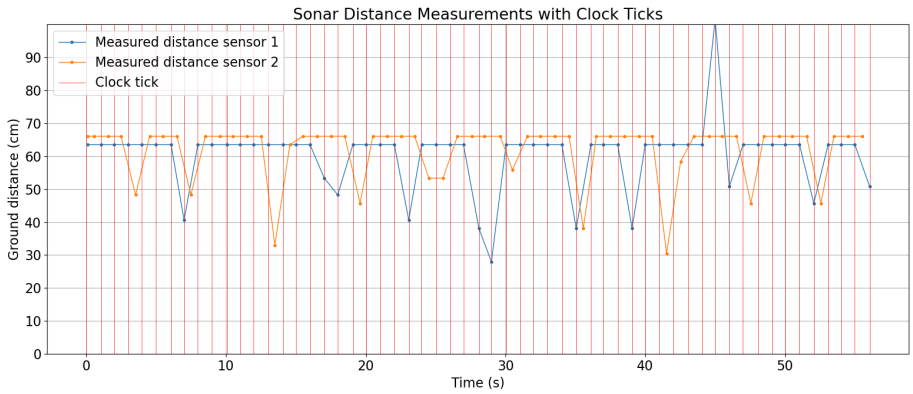


Figure 5.2: Sonar Measure with a Timeslot Size of 1000ms

The clock was clearly not to blame here. We can clearly see that spikes are still present, even with a timeslot of 1000ms.

Then, before gluing the foam to the wall panels, we decided to simply place the acoustic foam at the opposite end of the room, relative to the sonars. We did not put anything else in the room than the robot. Figure A.2 in Appendix A.2 shows a picture of the experiment setup.

Analyzing the results shown on Figure 5.3 retrieved with the setup of Figure A.2 in Appendix A.2, we can observe that the absorbing material modifies the measures in two significant ways :

1. The number of small errors is greatly decreased, bringing them almost to a full stop. This is a great improvement as it allows our filters to give a more precise estimate of the real distance.

2. The impact of the large outlier spikes is highly reduced in most cases. On Figure 5.1, we can see that the spikes have an error of between 40 to 45 cm. After the foam is installed, most errors get compressed to about 25cm, although some of them still reach the 40cm range. In addition, the length of the peaks is also reduced. This last observation is very important as the filters implemented afterwards use a fixed-sized set of previous measures to

filter the faulty ones. Having shorter errors thus reduces the impact of the outliers on the final result.

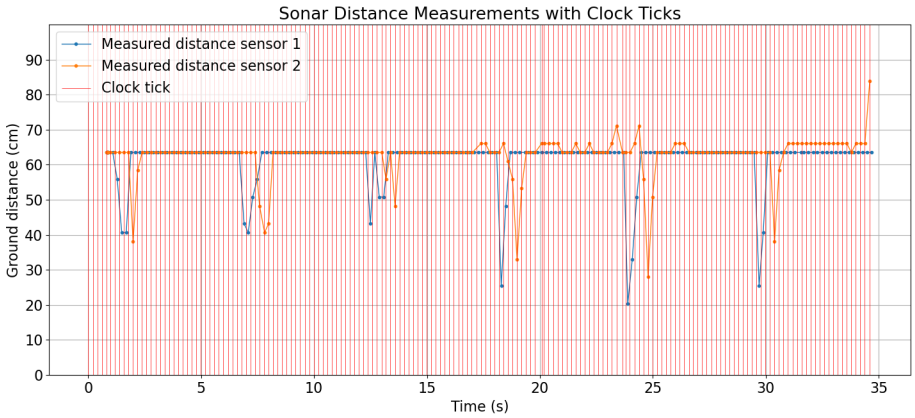


Figure 5.3: Sonar Measure with Acoustic Foam

Based on the previous observations, we can safely say that the absorbing acoustic foam in the room has a positive impact on the precision of our measures.

On the other hand, the big spikes are still present, and even though their impact is less pronounced, an error of 25 cm in rooms of 1.14m X 1.14m is still a 22% error rate, which is huge. We were not able to pinpoint the precise cause of this error, and the investigation necessary for the finding of the cause of this error goes beyond the scope of this master thesis, but we still have some candidates that could explain these remaining spikes :

- The error may come from reflections outside the wooden test environment. The ceiling or the walls of the labs where we tested our system may be responsible.
- The error may also come from an other external noise. It may be caused by an electronic device polluting the environment around the sonars.
- Although not likely, it is possible that the foam used isn't very effective.

Nevertheless, as absorption had a positive impact, we decided to include some acoustic foam to the setup, cutting the foam to try to get a more even distribution of the absorption, without having to cover all the walls with it.

5.1.2 The influence of the implemented filters

As seen in the previous point, some big error spikes still exist, even after the absorption was installed. To mitigate these errors, we decided to introduce some filtering on the measure. We tested three different filters, and then built upon our findings. We used a scenario where the robot was moving to see the full impact of the filters on our measures.

Low Pass Filter (LPF)

As explained in Chapter 2.1, the low pass filter uses a weighted portion of the last measure (defined by the α parameter) to smooth the quick variations in the measurement. In order to find the optimal α value, we tried to understand the impact of this parameter on the measures. Figure 5.4 shows the impact of α on the measurement.

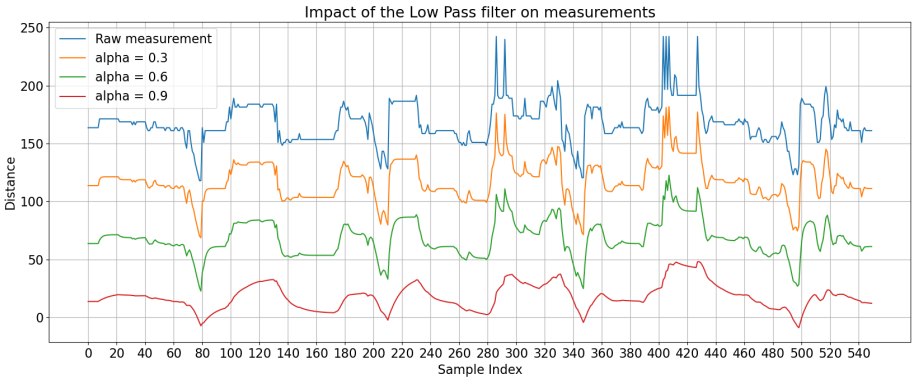


Figure 5.4: LPF Impact Depending on the Value of α

Two phenomenons can be deduced from this graph. While, as expected, the higher the α parameter, the smoother the measurement gets, we can see that the increase of α also induces a loss of information for the small variations.

Hampel Filter

Then, we tried the Hampel filter. Here, two parameters are available for tuning. First, the Hampel window size is the length of the set of measures used for filtering. The second parameter is the N_σ. This value determines the tolerancy of the filter to variation. To test the impact of the Hampel window size, we fixed N_σ at 1. Figure 5.5 shows our findings.

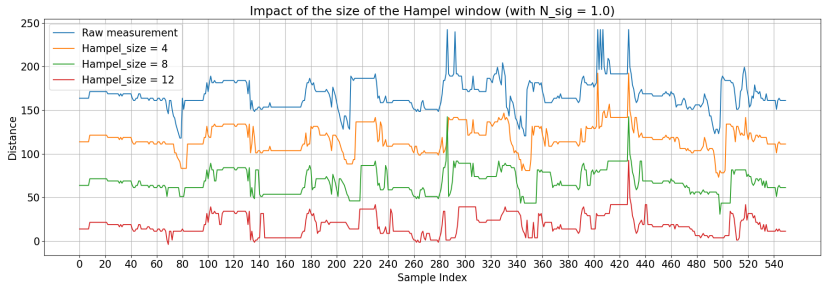


Figure 5.5: Hampel Filter Impact Depending on the Window Size

Analyzing the graph, we can see that the Hampel filter is good at filtering the outliers. We also notice that the filter introduces a delay between the movement and the acceptance of the new measurement data. It appears that this parameter seems to have an optimal point. On the green plot, the filter succeeds at rejecting most of the spikes, but successive spikes seems to create a tolerance for errors. On the other hand, the orange plot does not filter every outlier but seems to create less hallucinations. The optimal value of the Hampel window size must thus be between 4 and 8.

On Figure 5.6, we fixed the Hampel window size at 7 and changed the value of the N_σ parameter. What we can see is that this parameter seems to have less impact than expected. The sole impact that we were able to determine is that N_σ seems to have an impact on the duration of the error caused by the outliers when they are accepted.

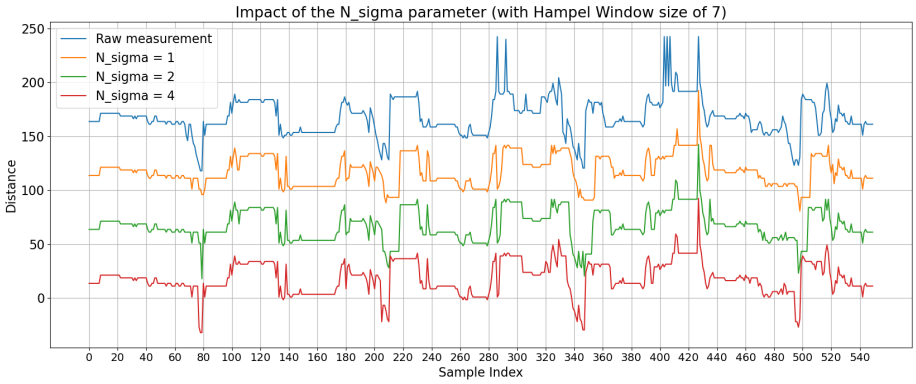


Figure 5.6: Hampel Filter Impact Depending on the N_σ Parameter

Median Smoothing Filter

The third filter we tried was a Median Smoothing Filter. This filter simply works by taking the median value of a neighbouring set of values. The only parameter available is the size of this set. As for the other filters, we tried different values to determine the impact of this parameter on the final result. Figure 5.7 shows the results given by this filter.

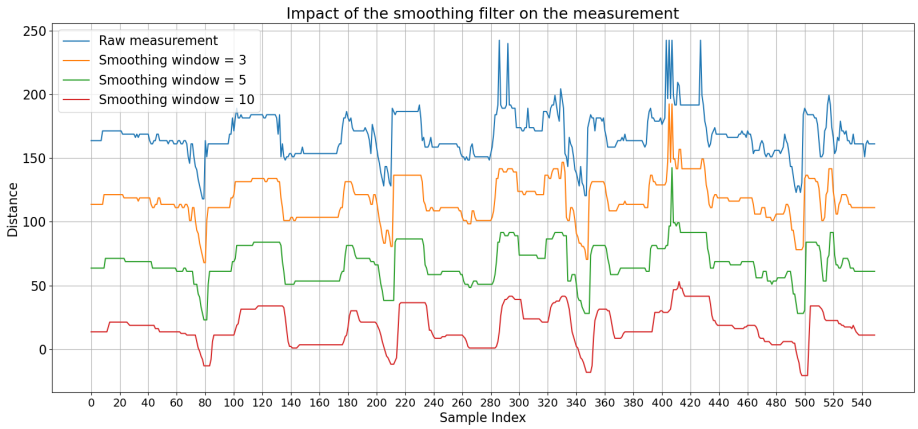


Figure 5.7: Median Smoothing Filter Impact Depending on the Window Size

This simple filter seems to give great results. It greatly improves the stability of the measures at the cost of a bit of inaccuracy. From the graph, we can deduce that the stability increases with the size of the smoothing window, but so does the inaccuracy of the measure.

After testing those three filters, we wondered which of them was the best fit in our case. All three present some advantages and imperfections. The LPF is not strong enough to completely smooth the measurements when big spikes arise. The Hampel filter is good at filtering the big spikes, but did not convince us much as it creates new spikes from time to time. The Median Smoothing Filter showed good stability, but as those data are used as corrector for the position of the robot, the inaccuracy that it introduces can be problematic.

We thus tried to combine the filters, which allowed us to use softer parameters and limit the impact of each of the filter's downsides. We found out that the Hampel Filter worked better when paired with a soft Low Pass. Trying multiple combination of filters, we found a satisfactory filter by using the three filters one

after the other, with the following parameters value :

- Low Pass Filter : $\alpha = 0.2$
- Hampel Filter : Window size = 7, $N_\sigma = 2$
- Median Smoothing Filter : Window size = 3

Figure 5.8 shows the impact of each filter on the resulting measures.

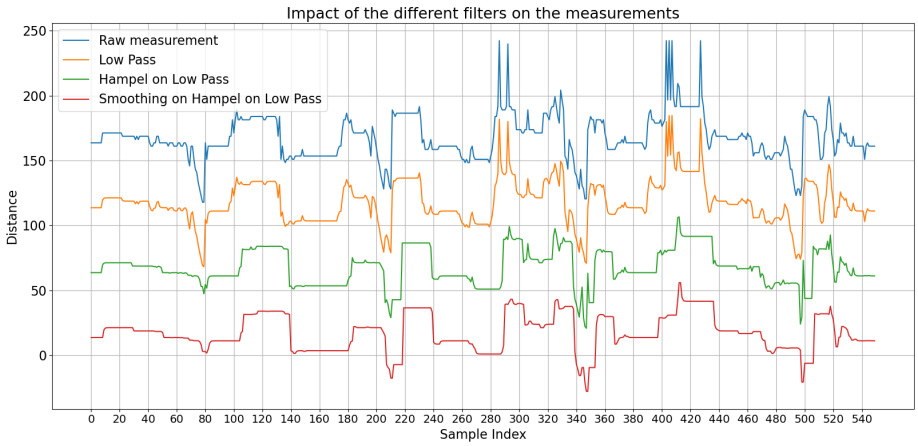


Figure 5.8: Three Filters Impact on the Measures

5.2 Sensor Fusion/Kalman Filter Tests

In order to successfully track our robot agent in our environment, we need to know precisely where it is located in our room. That is why, in this section, we will detail, analyse and validate the various experiments we carried out with our Kalman filters.

First, we conducted static tests to identify the optimal values for our covariance matrices, ensuring that the Kalman filters could respond as accurately and robustly as possible to their environment.

To build a reliable position Kalman filter, we needed a robot orientation angle that was both very stable and accurate. Therefore, we first tested the dynamics of our orientation Kalman filter in order to validate and refine our covariance matrices.

Finally, we performed dynamic tests on our position Kalman filter that also incorporates orientation estimates.

For these various tests, the values of the different covariance matrices were those presented in Chapter 4.3.

5.2.1 Determination of the static position of the robot

For this experiment, we studied the Kalman filter for position and the Kalman filter for orientation independently, as they both serve different purposes when the robot is stationary and rotating without changing places.

Position Kalman filter

When the robot is stationary, the position Kalman filter aims at maintaining an accurate estimate of the robot's position using the available data, such as wheel speed and sonar measurements. The filter must rely on the state transition model and sensor inputs to remain as close as possible to the true position, despite the presence of noise.

To evaluate whether our Kalman filter correctly integrates the data it receives, we conducted a simple experiment. We placed our robot at a distance where the sonar sensors could reliably detect it, and we analyzed, using a graph, whether the estimated position remained stable even when the sonar data was occasionally noisy or returned erroneous measurements.

This initial experiment provided valuable insights into the tuning of the $\mathbf{Q}$ and $\mathbf{R}$ covariance matrices. Recall that $\mathbf{Q}$ represents the process noise covariance, which models uncertainties in the prediction step (e.g., imprecise motion model or wheel slip), while $\mathbf{R}$ represents the measurement noise covariance, which accounts for uncertainties in the sensors (e.g., sonar noise). Since the purpose of this test was to achieve a stable position estimate, we experimentally adjusted the different entries of $\mathbf{Q}$ and $\mathbf{R}$ to obtain the most stable response possible, ensuring that the Kalman filter neither diverged nor became overly sensitive to measurement disturbances.

We conducted one test. The test was performed when the robot was completely stationary and did not move from its starting point.

Test results:

Figure 5.10 shows that our results are in line with our expectations. Indeed, we can see that the robot's position is static over time. Some measurements deviate from the initial point, but this is negligible compared to the size of the robot's environment. The maximum deviation from the initial point is approximately 0.1 cm. Details of the measurements of position x as a function of time and y as a function of time can be found in Figure 5.10 *a* and *b*.

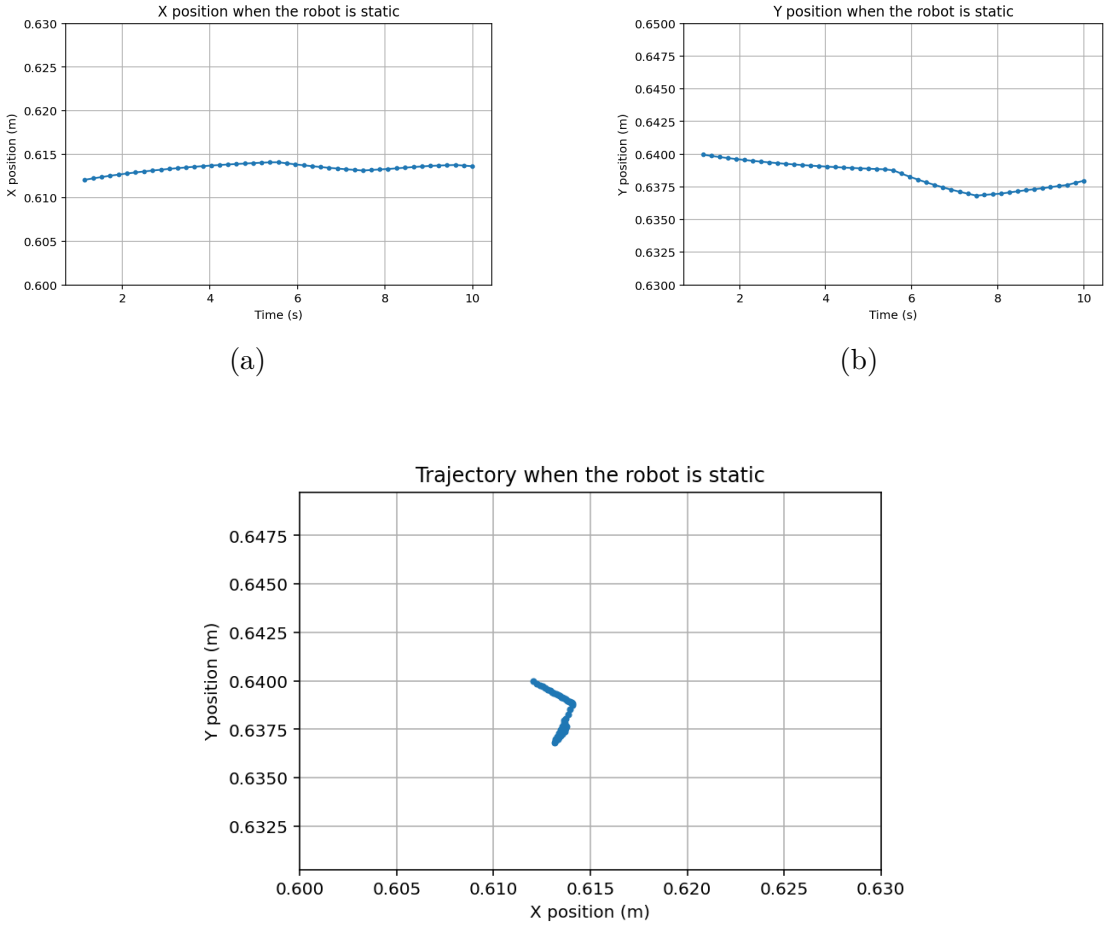


Figure 5.10: Static Kalman Position Test Results

Orientation Kalman filter

For the Kalman filter of orientation, the principle is similar to that of the Kalman filter of position. The main difference is that we analyse the estimated orientation angle produced by the filter. Thanks to this experiment, we were also able to validate specific values for the covariance matrices **Q** and **R** in order to ensure a balance between modelling process noise and taking measurement noise into account.

Results

In this test, we can see in the Figure 5.11 that the angle remains static over time. A slight drift remains but it is corrected over time thanks to our orientation Kalman filter measurements.

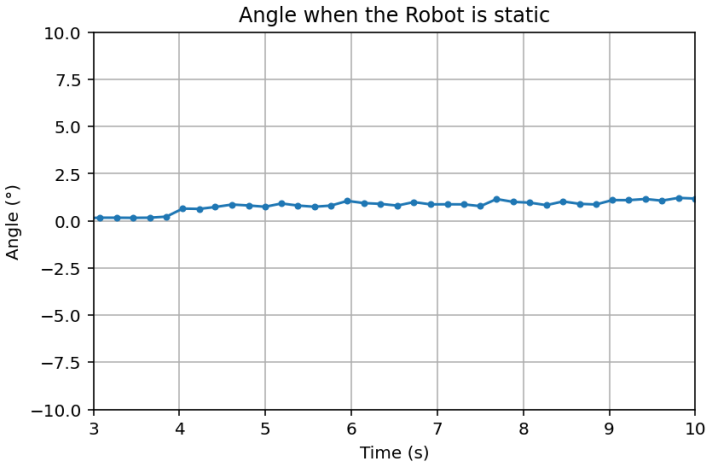


Figure 5.11: Kalman Orientation Results for a Static Position

5.2.2 Determination of the dynamic angle of the robot

The most critical factor in validating our Kalman filters was the robot's orientation, as its position estimation depends directly on it. We had already validated the orientation under static conditions, and now we wanted to test the filter's response when the robot moves in its environment. It was essential to ensure very accurate orientation in order to avoid the propagation of errors in the position calculation. To do this, we conducted a series of tests designed to cover as many realistic scenarios as possible.

We carried out a simple test. We started with the robot at zero degrees, then turned it to -90 degrees, then turned it directly to 90 degrees, and then returned it to 0 degrees.

We can see from the Figure 5.12 that the test is successful and the Kalman orientation completely follows the robot's movement.

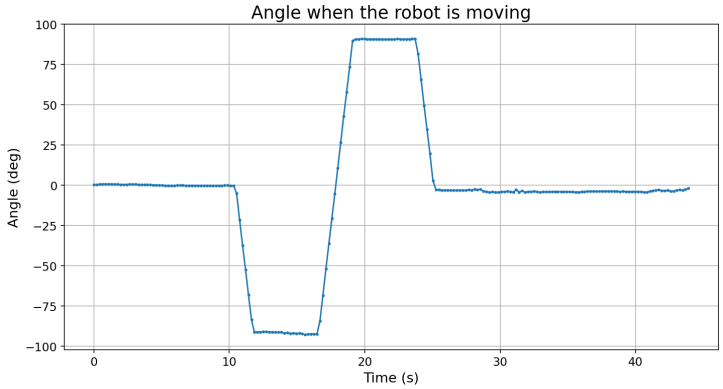


Figure 5.12: Kalman orientation results for a dynamic position

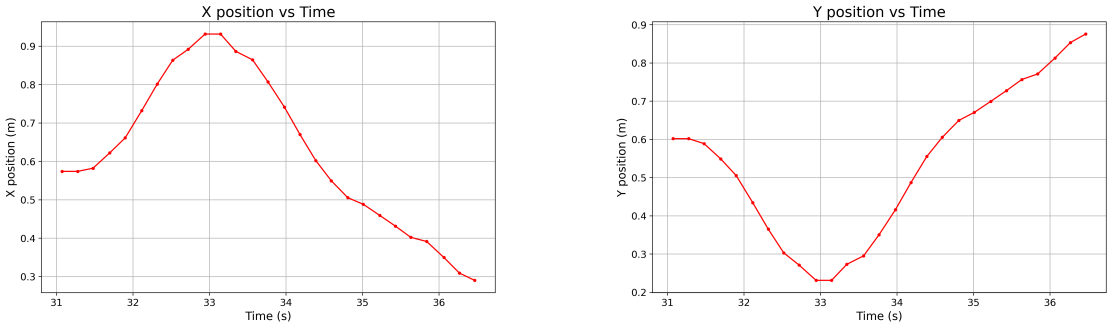
5.2.3 Determination of the dynamic position of the robot

In order to validate our covariance values, we performed two tests to ensure that our position Kalman filter worked correctly.

To validate all directions, we decided to conduct a test at a constant angle of 45 degrees to see if our robot could reach all areas of our environment.

Test results:

For this test, we decided to start at the beginning of our environment. Then we moved from one corner of the environment to another. We can see in Figures 5.14 that our Kalman filter is fully capable of covering the entire environment.



(a) (b)

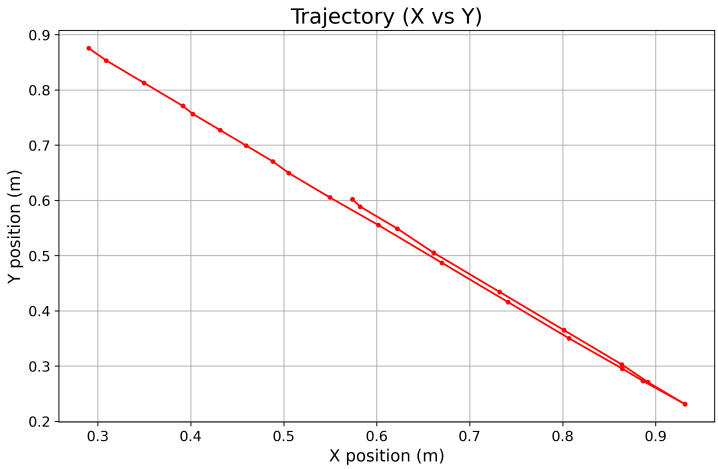


Figure 5.14: Dynamic Kalman Position Test Results with an Angle of 45°

Last test results

After validating all previous tests, we conducted a comprehensive evaluation covering all possible scenarios. This final test enabled us to validate all of our filters and guarantee the accuracy of the robot's movements in its environment.

To do this, we created a square in our environment to validate the combination of our Kalman filter. The results obtained are shown in Figure 5.15.

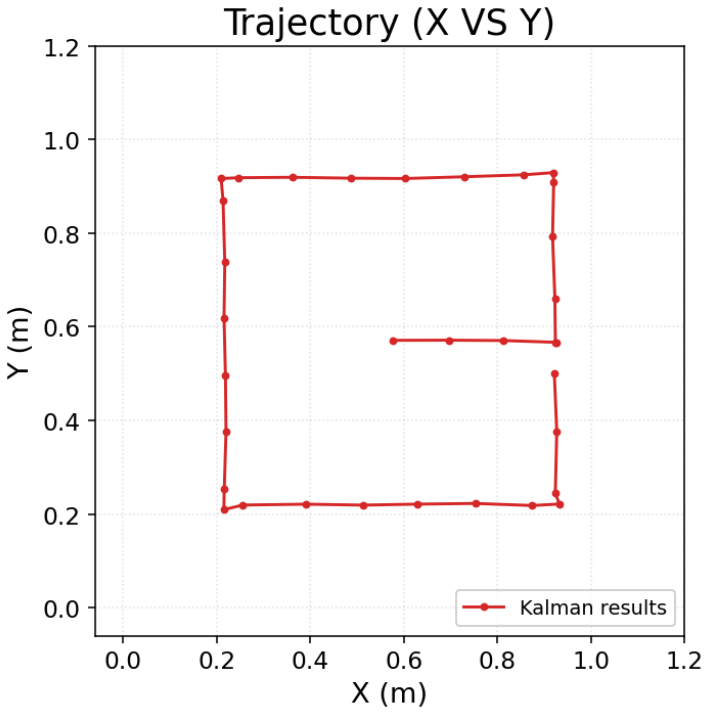


Figure 5.15: Kalman Position Results

We can see that it is feasible to make a square with our Kalman measurements. However, it was quite difficult to make it correctly. We had to repeat the experiment many times to achieve the expected result. This shows that our Kalman filters are functional but not necessarily robust in all circumstances.

5.3 Handover and Propagation Protocols Tests

5.3.1 Handover protocol test

For this test, we created a two room setup in a straight line (on the X axis). Then by simply controlling the robot for it to move from room 0 to room 1, we were able to visually see that the handover was working (through the sensors LED going from blue to green in room 1, and from green to blue in room 0). This first assumptions was then confirmed by the data we were able to collect. Figure 5.16 shows those results.

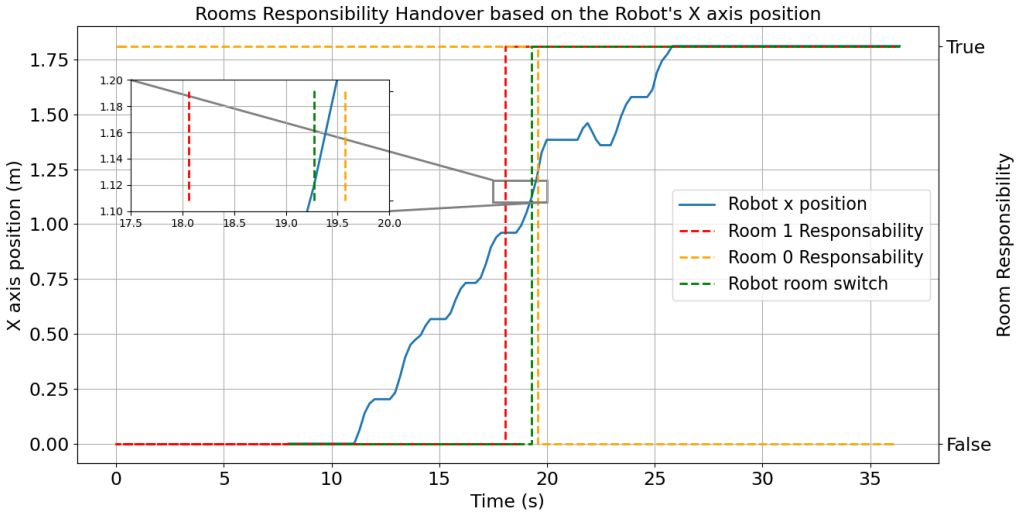


Figure 5.16: Graph of the Handover of Responsibility Between Rooms

This graph illustrates the inner works of our handover protocol. The blue curve shows the position of the robot on the x axis and the dashed lines show the trigger of events. In red and orange, we get a boolean True value if the room is currently measuring. In green we get a True if the robot listens to room 1, listening to room 0 otherwise.

We can clearly observe that room 1 starts measuring a bit before the robot enters its room, as explained in Chapter 4.2.4. Then we can see that when the robot enters room 1, it switches its focus to the sensors in room number 1. Lastly, we can notice that when the robot is sufficiently far away from room 0, its sensors stop measuring.

Additionally, to avoid oscillations between rooms when the robot is near the boundary, we implemented a form of hysteresis. This hysteresis is achieved through a buffer zone where both room's sensors are activated when the robot is in the transition area. However, the robot itself determines when to change its active room based on its calculated prediction. This approach ensures that even in a rapid oscillation between rooms, the robot will keep its position correct. The system's decision to switch rooms is therefore not based solely on sensor input but mainly influenced by the robot's internal calculations, providing a stable transition without unnecessary fluctuations.

Furthermore, since the change in coordinates is strongly linked to the speed of the robot's wheels, and given the robot's stability during operation, we can conclude that the change in direction is not abrupt. Instead, the transition from one room to the other is precise, At one point in space and determined solely by the robot.

5.3.2 Propagation protocol test

Routing graph test

In order to test that our artificial routing was done correctly, we monitored all the configuration messages received by the sensors during the configuration phase. See Appendix C to see those logs.

The mockup was put in a L shape with *sensor_1* and *sensor_2* in room zero, *sensor_3* and *sensor_4* in room one and *sensor_5* and *sensor_6* in room two.

By analyzing the logs, we can check that *sensor_1* received :

- server, sensor_1, robot and sensor_2 with a *Add_Device* message (Direct communication).
- sensor_3 and sensor_4 with a *Add_Link* message (propagation communication).

The sensor_3 logs show that it received :

- server, sensor_3, robot and sensor_4 with a *Add_Device* message (Direct communication).
- sensor_1, sensor_2, sensor_5 and sensor_6 with a *Add_Link* message (propagation communication).

The sensor_5 logs show that it received :

- server, sensor_5, robot and sensor_6 with a *Add_Device* message (Direct communication).
- sensor_3, sensor_4 with a *Add_Link* message (propagation communication).

This is what was expected by our routing protocol (see Figure 3.11)

Propagation test

To test whether our propagation protocol worked as intended, we controlled the robot moving through the rooms to see if all the rooms received all the informations. Figure 5.17 shows our results.

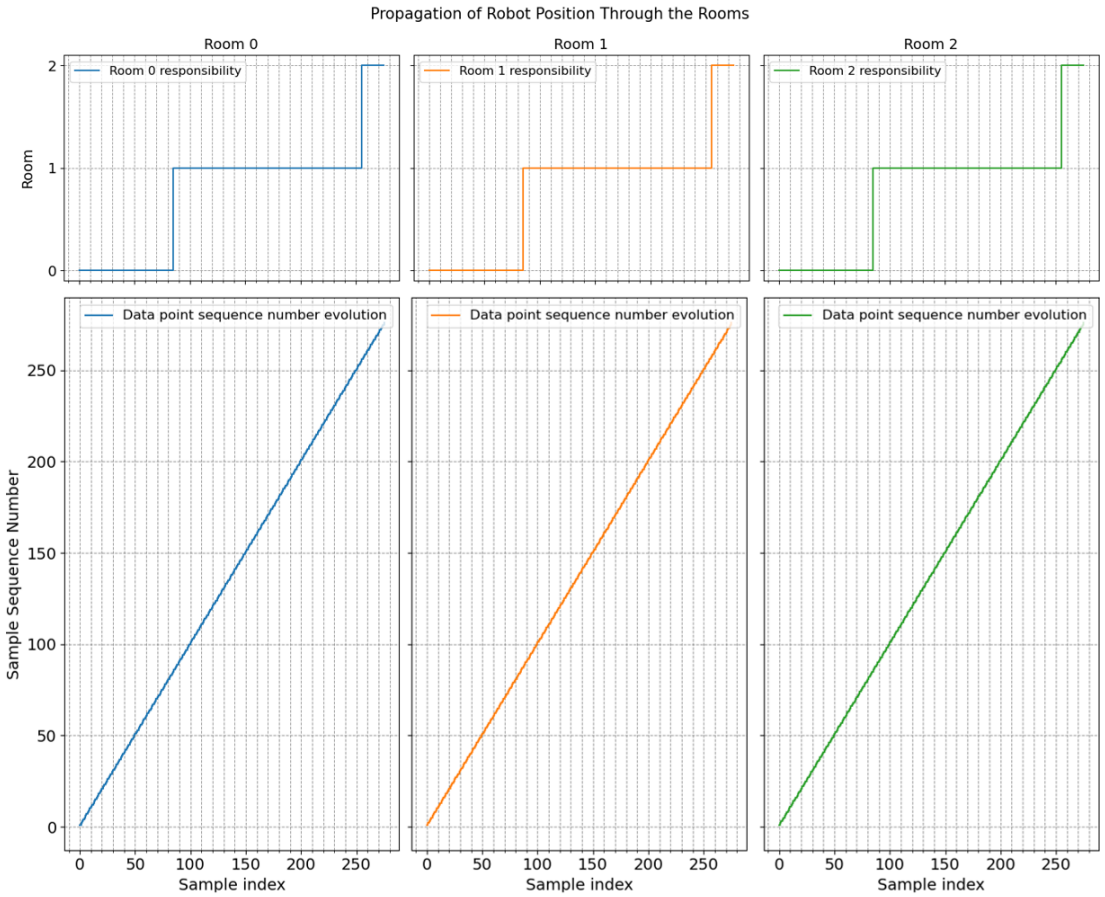


Figure 5.17: Room Position and Sequence Number Evolution

In the first row, each graph represents the knowledge of a specific room, showing the room in which the robot is located according to that room's data. The handover from room 0 to room 1 occurs around index 85, and the handover from room 1 to room 2 occurs around index 255.

In the second row, we plot the evolution of the sequence number associated with each data point. This sequence number is attached to every hera_data message and corresponds to a measurement.

Based on the routing that has been checked by the logs, we can conclude that when the robot is in room 0, room 2 should not be able to know where the robot is, unless the information is propagated through room 1. Figure 5.18 shows what the knowledge would look like without propagation. The fact that that room 1 has all knowledge is due to the fact that it is central in this small scale mockup. Its information would become partial in a larger building.

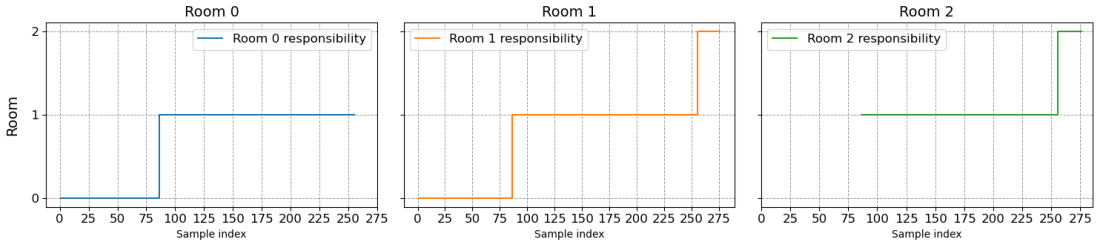


Figure 5.18: Room Knowledge Without the Propagation Protocol

Since that knowledge is shared and since all three rooms received the same hera_data sequence number, we can conclude that, at least in this three rooms setup, the simple propagation protocol we implemented works as intended.

Chapter 6

Conclusion

6.1 Discussion

As a conclusion, we reached to achieve all the objectives initially fixed for our master's thesis. Our list of contributions is thus as follows:

- We refactored and extended the software of a robot built for a previous master's thesis and integrated it in a larger system.
- We created a multi-agent distributed system using GRiSP2 prototyping boards and sensor-fusion to allow a robot to determine its position. we indeed added a new layer of awareness (position awareness) in the process of the robot.
- We demonstrated the feasibility for a robot to dynamically hand over its focus to different groups of sensors.
- We demonstrated how useful the Hera framework is for the creation of an IoT distributed system.
- We added new functionalities to the Hera framework.
- We clarified the organization and document the Hera framework's API and inner works.
- We implemented a time multiplexing like protocol for the prevention of cross talk interferences in the sonar sensors.
- We created a propagation protocol that allows communication in a discontinued environment.

- We implemented an executive function above a timeclock in order to check whether a measurement is needed.
- We implemented a Kalman filter using Hera to determine the position, angle and room in which an object is moving.
- We created a data representation of a mutli-room building.

6.2 Limitations

As great as it looks, our system still shows lots of limitations.

First, the major limitation of our system is the Pmod maxsonar. The opening angle of the sonar (30°) does not allow a sufficient area of coverage. The measures from the sonars are used as corrector for the estimated state of the system. It would thus greatly benefit from an increase in the area covered. Note that when scalled up, this area increases.

The usage of range finding sonars also prohibits the upscalling of the system. Indeed, those sensors only see the closest object in their sight, not all of them. In a room filled with furniture, these sonars wouldn't even be able to find the robot.

In this small scale mockup, the width of the robot produces a significant relative error on its position estimate. Thankfully, the relative size of the robot decreases as the size of the rooms increases. This error should thus reduce, to be negligible in a full sized room.

Even though those limitations are significant, we expect their influence to reduce when the environment used grows.

To stay on the sensors, the electric consumption of the GRiSP2 measuring at full speed and communicating by WiFi with the other agents burns quickly through the small lithium battery pack that we integrated. In order to create a product out of this system, it would be necessary to find a better suited battery pack, or to find ways to reduce the energy consumption of the system.

We achieved a total number of 8 devices, which is the maximum that can connect to the LilyGo ESP32 access point (six sensors, one robot and one controller). This can be a limitation if the number of room in a building is greater that three.

The precision of the clock protocol that we implemented is bound by the Round Trip Time (RTT) necessary for a "*tick*" message to reach the other sensor. If any congestion occurs on the access point, then the clock can drift, causing new errors on the sonar measures.

We also found the SD ports in the GRiSP2 boards to be capricious. The card were sometimes not recognized. When using multiple cards, it can quickly become unpleasant.

In addition to these limitations, the ones found by Cédric Ponsard and François Goens [6] in their master's thesis are still applicable. To name a few, the I2C communication between the GRiSP and the ESP32 sometimes can't be opened, the Pmod Nav is sometime unreachable by the GRiSP and the robot sometimes fall unexpectedly due to a drop in performance.

6.3 Future Work

Our work considerably contributes to the project of a finished GRiSP/Hera robot product. However the road ahead remains long. In this section, we discuss the possible future developments that would be interesting towards this goal.

First, and as discussed in the previous chapter, the current performance of the sensors is a big bottleneck for this project. The arrival of new, more performant sensors such as an **Ultrawide Band sensor** [20] or a **Time of Flight (ToF)** [1] sensors could allow the sensors to get a more accurate estimated position of the robot, thus allowing a better precision of the position deduced by the Kalman filter. It would then become possible to test and extend the system at **full scale** in a real building.

Another limitation of the sensors that we discussed was their energy consumption. It is true that all the sensors keep drawing energy from their battery even when idle in a room far away from the robot. Latest advances in the **GRiSP Nano** prototyping boards allow us to hope that longer lasting energy-aware sensors can be developped, maybe integrating a sleep mode when the robot is far away from the concerned room.

In order to further reduce the energy consumption of the sensors, **new communication media** could replace the current WiFi communication between the different agents in the system. Since the robot and the User Controller already use LoRa to communicate the movement commands, it would make sense to use the same communication media for all communication, further reducing the energy consumption in the process.

Second, the current system should work in a 3D building with different floors, as their must be a link (e.g a staircase) between floors that allow to flatten the building as a 2D representation. But in the current setup, the rooms are considered static, what happens if the **rooms start moving** ? In an hotel where the link between floors is an elevator. In this case, the room representing the elevator moves through the building's graph, requiring the graph to be dynamically modified each time the elevator arrives at a different floor.

Third, on another topic, the user controller is, for now, more a testing controller than a finished product. A new controller could be implemented using more suited technologies such as React for example. It could be implemented as a **mobile app**, improving the end-user experience and the portability of the system.

Last but not least, it would be nice to give the system a **practical purpose**. In their ongoing master's thesis, Arthur Dandoy and Sam Raymackers are transforming the robot into a mobile table and implementing an obstacle avoidance layer. Merged with our project, this opens an exciting possibility. It would then be possible to implement even another layer : **autonomous movements**.

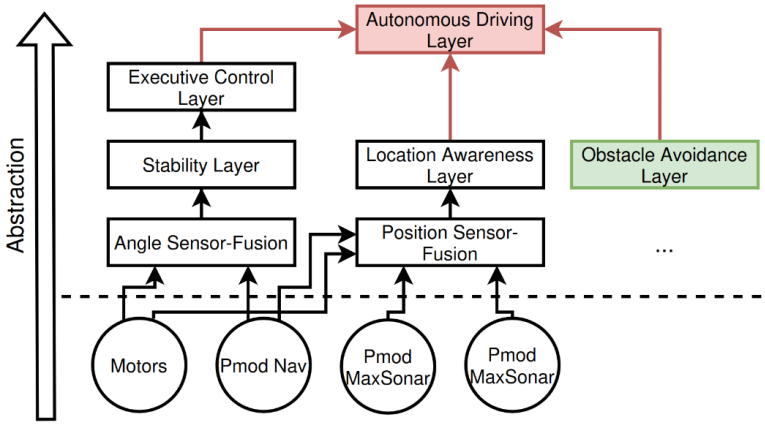


Figure 6.1: Possible Autonomous Robot Architecture

Once empowered by autonomous movement, numerous applications become possible. The robot could, for example, be implemented as a house assistant for elderly or handicaped people, helping them stay independent, as a nurse assistant in a hospital carrying tools and medications... or a butler-robot carrying glasses of wine around in a pub.

The GRiSP/Hera robotic system could then evolve from a proof of concept to a versatile, reliable system, ready to tackle real-word issues and improve lives in numerous meaningful ways.

Bibliography

[1] Time-of-flight camera - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Time-of-flight_camera. [Accessed 09-08-2025].

[2] Alexander. Median Absolute Deviation — statisticshowto.com. https://www.statisticshowto.com/median-absolute-deviation/. [Accessed 11-07-2025].

[3] Rayko Agramonte Claudio Urrea. Kalman Filter: Historical Overview and Review of Its Use in Robotics 60 Years after Its Creation. https://onlinelibrary.wiley.com/doi/10.1155/2021/9674015. [Accessed 31-07-2025].

[4] The Erlang community. Erlang System Documentation v28.0.1. https://www.erlang.org/doc/system/design_principles.html. [Accessed 13-07-2025].

[5] Wikipedia contributors. Quaternion — wikipedia, free encuclopedia. Quaternion, 2025. [Accessed 21-07-2025].

[6] François Goens Cédric Ponsard. Dynamic balancing in the real world with Grisp. https://thesis.dial.uclouvain.be/entities/masterthesis/c11f8a20-c183-4f9e-b3ec-b14248faa61a, 2024. [Accessed 13-07-2025].

[7] Shilpa Devalal and A. Karthikeyan. Lora technology - an overview. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 284–290, 2018.

[8] DigilentTeam. Digilent Pmod Maxsonar. https://digilent.com/reference/pmod/pmodmaxsonar/start. [Accessed 03-07-2025].

[9] Peer Stritzinger GmbH. GRiSP Ecosystem by Stritzinger - Erlang & Elixir — grisp.org. https://www.grisp.org/hardware. [Accessed 03-07-2025].

[10] Peer Stritzinger GmbH. Peer Stritzinger GmbH — stritzinger.com. https://stritzinger.com/. [Accessed 03-07-2025].

[11] Stanley F.Schmidt Leonard A.McGee. Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry. https://ntrs.nasa.gov/api/citations/19860003843/downloads/19860003843.pdf, November 1985. [Accessed 01-08-2025].

[12] Satya Nadella. Microsoft build 2016 keynote. https://www.youtube.com/watch?v=b5ZuP4vvDco, 2016. [Accessed 10-08-2025].

[13] Lucas Nélis. Low-cost high-speed sensor fusion with Grisp and Hera. https://thesis.dial.uclouvain.be/entities/masterthesis/afe439ce-8cc4-4148-8acc-38b667979fc2, 2023. [Accessed 13-07-2025].

[14] Miguel Otero Pedrido. Outlier detection using Hampel — migueloteropedrido. https://medium.com/@migueloteropedrido/hampel-filter-with-python-17db1d265375. [Accessed 11-07-2025].

[15] RTEMS Project. The RTEMS Project home — rtems.org. https://www.rtems.org/. [Accessed 03-07-2025].

[16] Hadrien Sonnet. Sensor fusion for three-dimensional movement of human beings on an internet of things network. https://thesis.dial.uclouvain.be/entities/masterthesis/9b5f4424-e137-4f1e-bd4b-3fd74f10bfec, 2024. [Accessed 03-08-2025].

[17] Vincent Verpoten Sébastien Kalbusch. The Hera framework for fault-tolerant sensor fusion on an Internet of Things network with application to inertial navigation and tracking. https://thesis.dial.uclouvain.be/entities/masterthesis/50fe2775-30f5-4f1f-bb41-a7620d5bcf4c, 2021. [Accessed 13-07-2025].

[18] Digilent Team. Digilent Pmod Nav. https://digilent.com/reference/pmod/pmodnav/start. [Accessed 07-07-2025].

[19] Erlang Team. persistent_term &x2014; erts v16.0.2 — erlang.org. https://www.erlang.org/doc/apps/erts/persistent_term.html. [Accessed 28-07-2025].

[20] Mapsted Team. Ultra Wideband Indoor Positioning- How It Works — mapsted.com. https://mapsted.com/blog/uwb-positioning-explained. [Accessed 08-08-2025].

[21] MaxBotix team. Ultrasonic Distance Sensor Datasheet: LV-MaxSonar-EZ — maxbotix.com. https://maxbotix.com/pages/lv-maxsonar-ez-datasheet. [Accessed 26-07-2025].

[22] Peer Stritzinger GmbH | GRiSP Team. GRiSP Ecosystem by Stritzinger - Erlang & Elixir — grisp.org. https://www.grisp.org/blog/posts/2025-06-11-grisp-nano-codebeam-sto. [Accessed 08-08-2025].

[23] Pygame_GUI team. Documentation - Home Page &x2014; Pygame GUI 0.6.14 documentation — pygame-gui.readthedocs.io. https://pygame-gui.readthedocs.io/en/latest/. [Accessed 05-08-2025].

[24] The MatLab Team. Hampel Filter Algorithm explaination. https://www.mathworks.com/help/dsp/ref/hampelfilter.html. [Accessed 11-07-2025].

[25] The Pygame Team. About the pygame library. https://www.pygame.org/wiki/about. [Accessed 05-08-2025].

[26] TinytronicsTeam. LilyGO TTGO T3 LoRa32 433MHz V1.6.1 ESP32 — tinytronics.nl. https://www.tinytronics.nl/nl/development-boards/microcontroller-boards/met-lora/lilygo-ttgo-t3-lora32-433mhz-v1.6.1-esp32. [Accessed 31-07-2025].

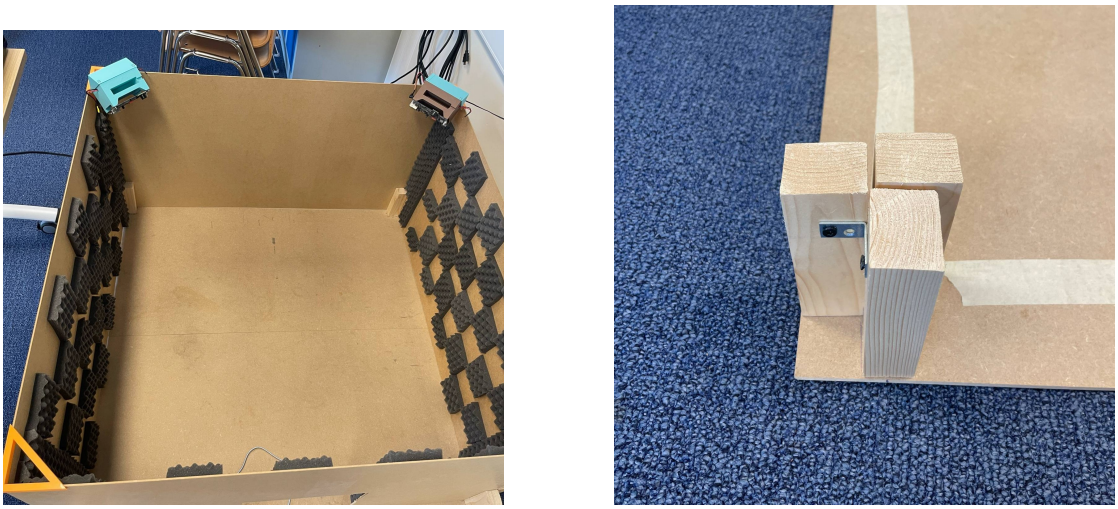
Part II

Appendix

Appendix A

Environment pictures

A.1 Mockup presentation



(a) Side wall panel (b) Side wall support

Figure A.1: Building of a room

A.2 Environment testing

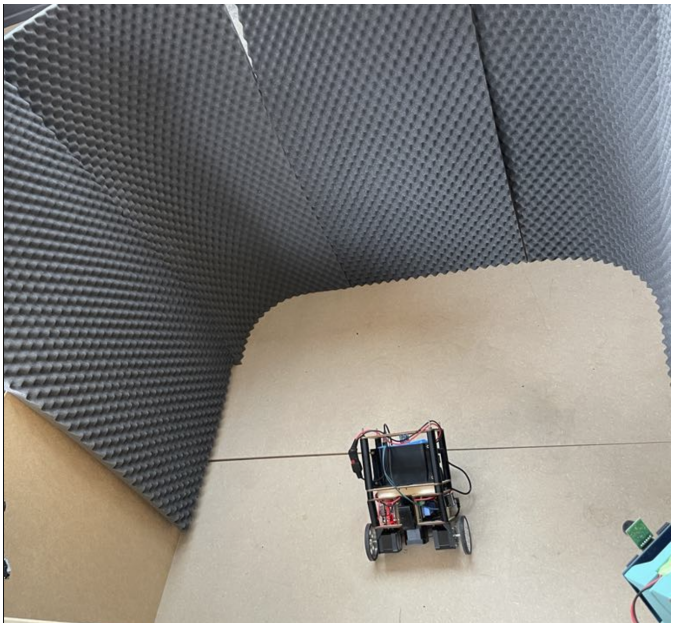
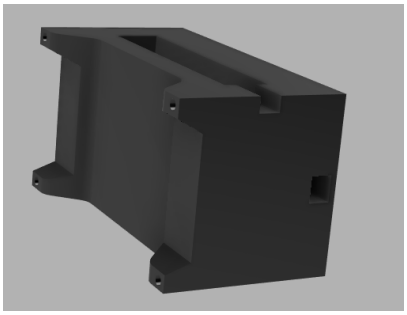


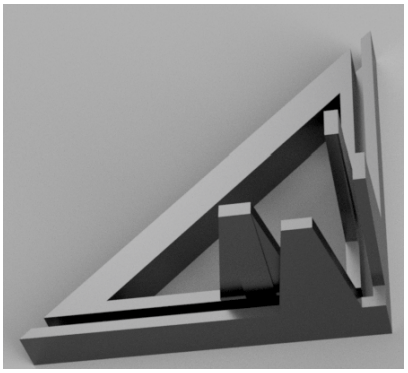
Figure A.2: Room adapted with foam

Figure A.3: Room setup with no absorption

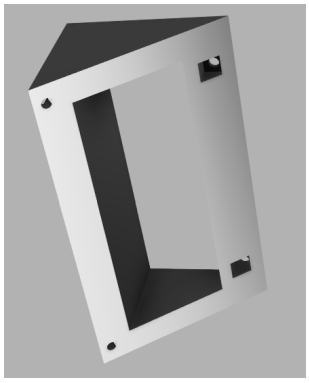
A.3 Sensor Construction



(a) 3D Printed GRiSP2 Stand



(b) 3D Printed Corner Adapter

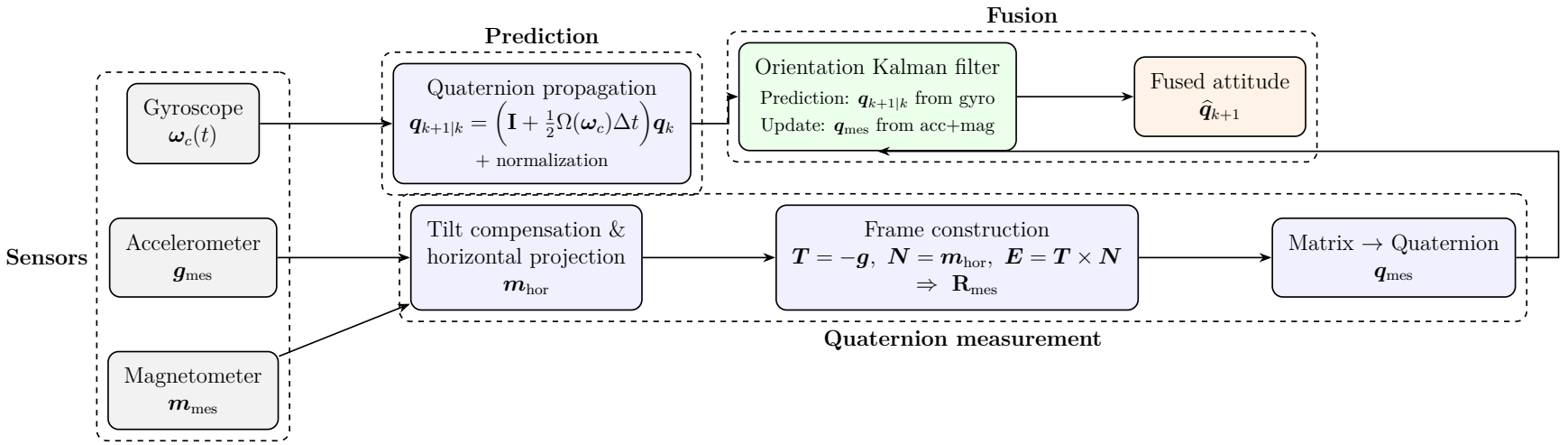


(c) 3D Printed Angle Adapter

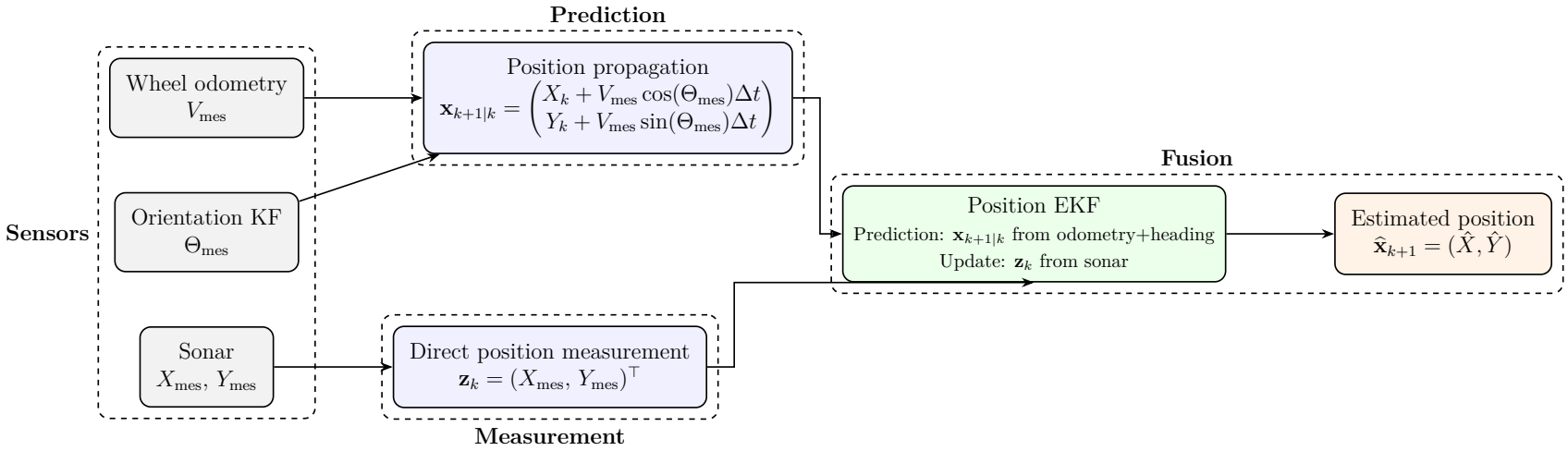
Figure A.4: Schemes of the Three 3D Printed Sonar Sensor Parts

Appendix B

Kalman filter visualisation diagram



(a) Measurement–prediction–fusion pipeline for quaternion-based orientation (AHRS).



(b) Measurement–prediction–fusion pipeline for position estimation.

Appendix C

Configuration logs

C.1 Sensor_1 (in room 0) logs

```
[HERA] Startup
[HERA] DebugMode on
[HERA_COM] Could not open socket: badarg
[SENSOR] start initialization sequence
[HERA_COM] Retrying in 1 [s], attempt number 1
[HERA_SUBSCRIBE] New subscriber <0.346.0>
[SENSOR] sensor id :1
[SENSOR] WiFi setup starting...
wlan0: WPA: Key negotiation completed with 64:b7:08:ad:3a:55 [PTK=
    CCMP GTK=CCMP]
wlan0: CTRL-EVENT-CONNECTED - Connection to 64:b7:08:ad:3a:55
    completed [id=0 id_str=]
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 2 [s], attempt number 2
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 4 [s], attempt number 3
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 8 [s], attempt number 4
err: wlan0: ipv4_addroute: File exists
err: wlan0: ipv4_addroute: File exists
[HERA_COM] Connected to private network with IP: {192,168,4,6}
[HERA_SUBSCRIBE] Notifying "connected"
[SENSOR] WiFi setup done

[SENSOR] Waiting for ping from server
[HERA_SUBSCRIBE] Notifying ["ping","server","192.168.4.4","5000"]
[HERA_COM] Discovered new device : server
[HERA_COM] Sending "Hello,1" to server
[HERA_SUBSCRIBE] Notifying ["Ack","server"]
[SENSOR] Received ACK from server
```

```
[SENSOR] Waiting for start signal ...
[HERA_SUBSCRIBE] Notifying ["Add_Device","sensor_1","192.168.4.6",
    "9000"]
[HERA_COM] Sending "Ack,Add_device,sensor_1,1" to server
[HERA_SUBSCRIBE] Notifying ["Pos","1","0.12","0.98","0.6","45","0"
    ]
[HERA_COM] Sending "Ack,Pos,sensor_1,1" to server
[HERA_DATA] Storing room, sensor_1, 1, [0]
[HERA_DATA] Storing pos, sensor_1, 1, [0.12,0.98,0.6,45]
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_3","192.168.4.7","
    9000"]
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_4","192.168.4.10","
    9000"]
[HERA_COM] Sending "Ack,Add_Link,sensor_3,1" to server
[SENSOR] Discovered new link : "sensor_3"
[HERA_COM] Sending "Ack,Add_Link,sensor_4,1" to server
[SENSOR] Discovered new link : "sensor_4"
[HERA_SUBSCRIBE] Notifying ["ping","server","192.168.4.4","5000"]
[HERA_SUBSCRIBE] Notifying ["Add_Device","sensor_2","192.168.4.8",
    "9000"]
[HERA_COM] Sending "Ack,Add_device,sensor_2,1" to server
[HERA_COM] Discovered new device : sensor_2
[HERA_SUBSCRIBE] Notifying ["Pos","2","0.16","0.12","0.6","315","0
    "]
[HERA_COM] Sending "Ack,Pos,sensor_2,1" to server
[HERA_DATA] Storing room, sensor_2, 1, [0]
[HERA_DATA] Storing pos, sensor_2, 1, [0.16,0.12,0.6,315]
[HERA_SUBSCRIBE] Notifying ["Room_info","0","0.0","0.0","1.14","
    1.14"]
[HERA_DATA] Storing room_info, 0, 1, [0.0,0.0,1.14,1.14]
[HERA_COM] Sending "Ack,Room_info,0,1" to server
[HERA_SUBSCRIBE] Notifying ["Room_info","1","1.14","0.0","2.28","
    1.14"]
[HERA_DATA] Storing room_info, 1, 1, [1.14,0.0,2.28,1.14]
[HERA_COM] Sending "Ack,Room_info,1,1" to server
[HERA_SUBSCRIBE] Notifying ["Room_info","2","1.14","1.14","2.28","
    2.28"]
[HERA_DATA] Storing room_info, 2, 1, [1.14,1.14,2.28,2.28]
[HERA_COM] Sending "Ack,Room_info,2,1" to server
[HERA_SUBSCRIBE] Notifying ["Add_Device","robot","192.168.4.2","
    9000"]
[HERA_COM] Sending "Ack,Add_device,robot,1" to server
[HERA_COM] Discovered new device : robot
[HERA_SUBSCRIBE] Notifying ["Init_pos","0.57","0.57","0.0","0"]
[HERA_SUBSCRIBE] Notifying ["Add_Device","robot","192.168.4.2","
    9000"]
[HERA_COM] Sending "Ack,Pos,robot,1" to server
[HERA_DATA] Storing robot_pos, robot, 1, [0.57,0.57,0.0,0]
[HERA_COM] Sending "Ack,Add_device,robot,1" to server
```

```
[HERA_SUBSCRIBE] Notifying ["Init_pos","0.57","0.57","0.0","0"]
[HERA_COM] Sending "Ack,Pos,robot,1" to server
[HERA_SUBSCRIBE] Notifying ["Start","192.168.4.4"]
[SENSOR] Start received, starting the computing phase
[SENSOR] Other sensor is : sensor_2
[HERA_COM] Sending "Handshake,1657,163533" to sensor_2
[HERA_SUBSCRIBE] Notifying ["Handshake","1677","163713"]
[SONAR_MEASURE] External priority higher, sensor role : SLAVE
[HERA_COM] Sending "Ok,role" to sensor_2
[HERA_SUBSCRIBE] Notifying ["Ok","role"]

[SONAR_MEASURE] Starting measurements
```

C.2 Sensor_3 (in room 1) logs

```
[HERA] Startup
[HERA] DebugMode on
[HERA_COM] Could not open socket: badarg
[SENSOR] start initialization sequence
[HERA_COM] Retrying in 1 [s], attempt number 1
[HERA_SUBSCRIBE] New subscriber <0.346.0>
[SENSOR] sensor id :3
[SENSOR] WiFi setup starting...
wlan0: Trying to associate with 64:b7:08:ad:3a:55 (SSID='RobotNet'
    freq=2412 MHz)
Failed to add supported operating classes IE
info: wlan0: link state changed to UP
wlan0: Associated with 64:b7:08:ad:3a:55
err: wlan0: ipv4_sendrawpacket: No buffer space available
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 2 [s], attempt number 2
wlan0: WPA: Key negotiation completed with 64:b7:08:ad:3a:55 [PTK=
   CCMP GTK=CCMP]
wlan0: CTRL-EVENT-CONNECTED - Connection to 64:b7:08:ad:3a:55
   completed [id=0 id_str=]
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 4 [s], attempt number 3
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 8 [s], attempt number 4
err: wlan0: ipv4_addroute: File exists
err: wlan0: ipv4_addroute: File exists
[HERA_COM] Connected to private network with IP: {192,168,4,7}
[HERA_SUBSCRIBE] Notifying "connected"
[SENSOR] WiFi setup done

[SENSOR] Waiting for ping from server
```

```
[HERA_SUBSCRIBE] Notifying ["ping","server","192.168.4.4","5000"]
[HERA_COM] Discovered new device : server
[HERA_COM] Sending "Hello,3" to server
[HERA_COM] Sending "Hello,3" to server
[HERA_SUBSCRIBE] Notifying ["Ack","server"]
[SENSOR] Received ACK from server
[SENSOR] Waiting for start signal ...

[HERA_SUBSCRIBE] Notifying ["Add_Device","sensor_3","192.168.4.7",
    "9000"]
[HERA_COM] Sending "Ack,Add_device,sensor_3,3" to server
[HERA_SUBSCRIBE] Notifying ["Pos","3","2.12","1.02","0.6","135","1
    "]
[HERA_COM] Sending "Ack,Pos,sensor_3,3" to server
[HERA_DATA] Storing room, sensor_3, 1, [1]
[HERA_DATA] Storing pos, sensor_3, 1, [2.12,1.02,0.6,135]
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_1","192.168.4.6","
    9000"]
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_2","192.168.4.8","
    9000"]
[HERA_COM] Sending "Ack,Add_Link,sensor_1,3" to server
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_6","192.168.4.9","
    9000"]
[SENSOR] Discovered new link : "sensor_1"
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_5","192.168.4.5","
    9000"]
[HERA_COM] Sending "Ack,Add_Link,sensor_2,3" to server
[SENSOR] Discovered new link : "sensor_2"
[HERA_COM] Sending "Ack,Add_Link,sensor_6,3" to server
[SENSOR] Discovered new link : "sensor_6"
[HERA_COM] Sending "Ack,Add_Link,sensor_5,3" to server
[SENSOR] Discovered new link : "sensor_5"
[HERA_SUBSCRIBE] Notifying ["ping","server","192.168.4.4","5000"]
[HERA_SUBSCRIBE] Notifying ["Add_Device","sensor_4","192.168.4.10"
    ,"9000"]
[HERA_COM] Sending "Ack,Add_device,sensor_4,3" to server
[HERA_COM] Discovered new device : sensor_4
[HERA_SUBSCRIBE] Notifying ["Pos","4","2.12","0.12","0.6","225","1
    "]
[HERA_COM] Sending "Ack,Pos,sensor_4,3" to server
[HERA_DATA] Storing room, sensor_4, 1, [1]
[HERA_DATA] Storing pos, sensor_4, 1, [2.12,0.12,0.6,225]
[HERA_SUBSCRIBE] Notifying ["Room_info","0","0.0","0.0","1.14","
    1.14"]
[HERA_DATA] Storing room_info, 0, 1, [0.0,0.0,1.14,1.14]
[HERA_COM] Sending "Ack,Room_info,0,3" to server
[HERA_SUBSCRIBE] Notifying ["Room_info","1","1.14","0.0","2.28","
    1.14"]
[HERA_DATA] Storing room_info, 1, 1, [1.14,0.0,2.28,1.14]
```

```
[HERA_COM] Sending "Ack,Room_info,1,3" to server
[HERA_SUBSCRIBE] Notifying ["Room_info","2","1.14","1.14","2.28","
    2.28"]
[HERA_DATA] Storing room_info, 2, 1, [1.14,1.14,2.28,2.28]
[HERA_COM] Sending "Ack,Room_info,2,3" to server
[HERA_SUBSCRIBE] Notifying ["Add_Device","robot","192.168.4.2","
    9000"]
[HERA_COM] Sending "Ack,Add_device,robot,3" to server
[HERA_COM] Discovered new device : robot
[HERA_SUBSCRIBE] Notifying ["Init_pos","0.57","0.57","0.0","0"]
[HERA_SUBSCRIBE] Notifying ["Add_Device","robot","192.168.4.2","
    9000"]
[HERA_COM] Sending "Ack,Pos,robot,3" to server
[HERA_DATA] Storing robot_pos, robot, 1, [0.57,0.57,0.0,0]
[HERA_COM] Sending "Ack,Add_device,robot,3" to server
[HERA_SUBSCRIBE] Notifying ["Init_pos","0.57","0.57","0.0","0"]
[HERA_COM] Sending "Ack,Pos,robot,3" to server
[HERA_SUBSCRIBE] Notifying ["Start","192.168.4.4"]
[SENSOR] Start received, starting the computing phase
[SENSOR] Other sensor is : sensor_4
[HERA_COM] Sending "Handshake,344,163738" to sensor_4
[HERA_SUBSCRIBE] Notifying ["Handshake","2900","121972"]
[SONAR_MEASURE] External priority higher, sensor role : SLAVE
[HERA_COM] Sending "Ok,role" to sensor_4
[HERA_SUBSCRIBE] Notifying ["Ok","role"]
[SONAR_MEASURE] Starting measurements
```

C.3 Sensor_5 (in room 2) logs

```
[HERA] Startup
[HERA] DebugMode on
[HERA_COM] Could not open socket: badarg
[SENSOR] start initialization sequence
[HERA_COM] Retrying in 1 [s], attempt number 1
[HERA_SUBSCRIBE] New subscriber <0.346.0>
[SENSOR] sensor id :5
[SENSOR] WiFi setup starting...
wlan0: Trying to associate with 64:b7:08:ad:3a:55 (SSID='RobotNet'
    freq=2412 MHz)
Failed to add supported operating classes IE
info: wlan0: link state changed to UP
wlan0: Associated with 64:b7:08:ad:3a:55
err: wlan0: ipv4_sendrawpacket: No buffer space available
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 2 [s], attempt number 2
wlan0: WPA: Key negotiation completed with 64:b7:08:ad:3a:55 [PTK=
    CCMP GTK=CCMP]
```

```
wlan0: CTRL-EVENT-CONNECTED - Connection to 64:b7:08:ad:3a:55
    completed [id=0 id_str=]
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 4 [s], attempt number 3
[HERA_COM] Could not open socket: badarg
[HERA_COM] Retrying in 8 [s], attempt number 4
err: wlan0: ipv4_addroute: File exists
err: wlan0: ipv4_addroute: File exists
[HERA_COM] Connected to private network with IP: {192,168,4,5}
[HERA_SUBSCRIBE] Notifying "connected"
[SENSOR] WiFi setup done

[SENSOR] Waiting for ping from server
[HERA_SUBSCRIBE] Notifying ["ping","server","192.168.4.4","5000"]
[HERA_COM] Discovered new device : server
[HERA_COM] Sending "Hello,5" to server
[HERA_SUBSCRIBE] Notifying ["ping","server","192.168.4.4","5000"]
[HERA_COM] Sending "Hello,5" to server
[HERA_SUBSCRIBE] Notifying ["Ack","server"]
[SENSOR] Received ACK from server
[SENSOR] Waiting for start signal ...

[HERA_SUBSCRIBE] Notifying ["Add_Device","sensor_5","192.168.4.5",
    "9000"]
[HERA_COM] Sending "Ack,Add_device,sensor_5,5" to server
[HERA_SUBSCRIBE] Notifying ["Pos","5","1.26","2.12","0.6","45","2"
    ]
[HERA_COM] Sending "Ack,Pos,sensor_5,5" to server
[HERA_DATA] Storing room, sensor_5, 1, [2]
[HERA_DATA] Storing pos, sensor_5, 1, [1.26,2.12,0.6,45]
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_4","192.168.4.10","
    9000"]
[HERA_SUBSCRIBE] Notifying ["Add_Link","sensor_3","192.168.4.7","
    9000"]
[HERA_COM] Sending "Ack,Add_Link,sensor_4,5" to server
[SENSOR] Discovered new link : "sensor_4"
[HERA_COM] Sending "Ack,Add_Link,sensor_3,5" to server
[SENSOR] Discovered new link : "sensor_3"
[HERA_SUBSCRIBE] Notifying ["ping","server","192.168.4.4","5000"]
[HERA_SUBSCRIBE] Notifying ["Add_Device","sensor_6","192.168.4.9",
    "9000"]
[HERA_COM] Sending "Ack,Add_device,sensor_6,5" to server
[HERA_COM] Discovered new device : sensor_6
[HERA_SUBSCRIBE] Notifying ["Pos","6","2.12","2.16","0.6","135","2
    "]
[HERA_COM] Sending "Ack,Pos,sensor_6,5" to server
[HERA_DATA] Storing room, sensor_6, 1, [2]
[HERA_DATA] Storing pos, sensor_6, 1, [2.12,2.16,0.6,135]
```

```
[HERA_SUBSCRIBE] Notifying ["Room_info","0","0.0","0.0","1.14","
    1.14"]
[HERA_DATA] Storing room_info, 0, 1, [0.0,0.0,1.14,1.14]
[HERA_COM] Sending "Ack,Room_info,0,5" to server
[HERA_SUBSCRIBE] Notifying ["Room_info","1","1.14","0.0","2.28","
    1.14"]
[HERA_DATA] Storing room_info, 1, 1, [1.14,0.0,2.28,1.14]
[HERA_COM] Sending "Ack,Room_info,1,5" to server
[HERA_SUBSCRIBE] Notifying ["Room_info","2","1.14","1.14","2.28","
    2.28"]
[HERA_DATA] Storing room_info, 2, 1, [1.14,1.14,2.28,2.28]
[HERA_COM] Sending "Ack,Room_info,2,5" to server
[HERA_SUBSCRIBE] Notifying ["Add_Device","robot","192.168.4.2","
    9000"]
[HERA_COM] Sending "Ack,Add_device,robot,5" to server
[HERA_COM] Discovered new device : robot
[HERA_SUBSCRIBE] Notifying ["Init_pos","0.57","0.57","0.0","0"]
[HERA_SUBSCRIBE] Notifying ["Add_Device","robot","192.168.4.2","
    9000"]
[HERA_COM] Sending "Ack,Pos,robot,5" to server
[HERA_DATA] Storing robot_pos, robot, 1, [0.57,0.57,0.0,0]
[HERA_COM] Sending "Ack,Add_device,robot,5" to server
[HERA_SUBSCRIBE] Notifying ["Init_pos","0.57","0.57","0.0","0"]
[HERA_COM] Sending "Ack,Pos,robot,5" to server
[HERA_SUBSCRIBE] Notifying ["Start","192.168.4.4"]
[SENSOR] Start received, starting the computing phase
[SENSOR] Other sensor is : sensor_6
[HERA_COM] Sending "Handshake,149,163817" to sensor_6
[HERA_SUBSCRIBE] Notifying ["Handshake","2589","128036"]
[HERA_SUBSCRIBE] Notifying ["Ok","role"]
[SONAR_MEASURE] External priority higher, sensor role : SLAVE
[HERA_COM] Sending "Ok,role" to sensor_6

[SONAR_MEASURE] Starting measurements
```

Appendix D

Hera Framework

D.1 Hera main module

```
1  -module(hera).
2
3  -behaviour(application).
4
5  -export([start_measure/2, timestamp/0, logg/2]).
6  -export([start/2, stop/1]).
7
8  -type timestamp() :: integer() | undefined.
9
10 -export_type([timestamp/0]).
11
12 %% starts and supervise the measure defined in the callback module
13 %% using Module:init(Args)
14 -spec start_measure(Module, Args) -> {ok, pid()} | {error, term()}
       when
15     Module :: module(),
16     Args :: term().
17
18 start_measure(Module, Args) ->
19     hera_measure_sup:start_child(Module, Args).
20
21
22 -spec timestamp() -> timestamp().
23
24 timestamp() ->
25   erlang:monotonic_time(millisecond).
26
27 logg(Message, Args) ->
28     DebugMode = persistent_term:get(debugMode),
29     if
```

```
30          DebugMode ->
31              io:format(Message, Args);
32          true ->
33              ok
34      end.
35
36  start(_StartType, _StartArgs) ->
37      io:format("[HERA] Startup~n"),
38      DebugMode = application:get_env(hera, debugmode, false),
39      case DebugMode of
40          true ->
41              persistent_term:put(debugMode, true),
42              hera:logg("[HERA] DebugMode on~n", []);
43          _ ->
44              persistent_term:put(debugMode, false),
45              ok
46      end,
47      hera_sup:start_link().
48
49  stop(_State) ->
50      ok.
```

D.2 hera_com module

```
-module(hera_com).

-export([start_link/0]).
-export([send/3, send/4, send_unicast/3, send_link/5, add_device/3, reset_devices/0]).
-export([encode_half_float/1,decode_half_float/1]).
-export([get_bits/1]).

-define(MULTICAST_ADDR, {239,255,0,1}).
-define(PORT, 9000).

start_link() ->
    persistent_term:put(devices, []),
    Pid = spawn_link(fun init/0),
    register(?MODULE, Pid),
    {ok, Pid}.


-spec send(Name, Seq, Values) -> ok when
    Name :: atom(),
    Seq :: pos_integer(),
    Values :: [number(), ...].

send(Name, Seq, Values) ->
    Message = {hera_data, Name, node(), Seq, Values},
    try ?MODULE ! {send_packet, term_to_binary(Message)}
    catch
        error:_ -> ok
    end,
    ok.

send(Name, Seq, From, Values) ->  % To use when wanting to use personalized naming
    Message = {hera_data, Name, From, Seq, Values},
    try ?MODULE ! {send_packet, term_to_binary(Message)}
    catch
        error:_ -> ok
    end,
    ok.

send_unicast(Name, Message, Type) -> % Allows to send String messages (Those messages will be notified by hera_subscribe)
    hera:logg("[HERA_COM] Sending ~p to ~p~n", [Message, Name]),
    NewMessage = case Type of
        "UTF8" -> Message;
        "Binary" -> term_to_binary(Message);
```

```
44          _ -> Message
45      end,
46      ?MODULE ! {send_packet_unicast, Name, NewMessage},
47      ok.
48
49  send_link(Name, Ip, Port, Message, Type) -> % allows to send a
        message to an unregistered Ip/Port address
50      hera:logg("[HERA_COM] Sending link ~p to ~p~n", [Message, Name
            ]),
51      NewMessage = case Type of
52          "UTF8" -> Message;
53          "Binary" -> term_to_binary(Message);
54          _ -> Message
55      end,
56      ?MODULE ! {send_packet_unicast, Ip, Port, NewMessage},
57      ok.
58
59  %
60  %Returns a list of each bit in the byte given by Byte
61  %e.g. Byte = 163 gives [true, false, true, false, false, false,
        true, true]
62  %
63  get_bits(Byte) ->
64      if
65          Byte =/= 255 ->
66              _ = [ (Byte band round(math:pow(2,X))) =/= 0 || X <-
                    [7,6,5,4,3,2,1,0]];
67          true ->
68              [false, false, false, false, false, false, false,
                    false]
69      end.
70
71  %
72  %Encodes a list of values from double (8 bytes) to half-float (2
        bytes)
73  %Values = [Double1,Double2,...]
74  %
75  encode_half_float(Values) ->
76          lists:map(fun(X) -> enc_hf(X) end, Values).
77
78  %
79  %Decodes a list of values from half-float (2 bytes) to double (8
        bytes)
80  %Values = [<<Hf1A, Hf1B>>, <<Hf2A, Hf2B>>, ...]
81  %e.g. Values = [<<16#43,16#0A>>, <<16#4B,16#0C>>] gives [3.52,
        14.1]
82  %
83  decode_half_float(Values) when is_list(Values) ->
        decode_half_float(Values, []).
```

```
84 decode_half_float([<<A:8, B:8>> | Rest], Acc) -> decode_half_float
       (Rest, Acc ++ [dec_hf(<<A, B>>)]);
85 decode_half_float([], Acc) -> Acc.
86
87 add_device(Name, Ip, Port) ->
88     Devices = persistent_term:get(devices),
89     case lists:member({Name, Ip, Port}, Devices) of
90         false ->
91             hera:logg("[HERA_COM] Discovered new device : ~p~n", [
                   Name]),
92             NewDevices = [{Name, Ip, Port} | Devices],
93             persistent_term:put(devices, NewDevices);
94         _ ->
95             ok
96     end.
97
98 reset_devices() ->
99     persistent_term:put(devices, []).
100
101 init() ->
102     Socket = open_socket(1, 1),
103     loop(Socket).
104
105 open_socket(Delay, Attempts) ->
106     try open_socket()
107     catch
108         error:Reason ->
109             hera:logg("[HERA_COM] Could not open socket: ~p~n", [
                   Reason]),
110             hera:logg("[HERA_COM] Retrying in ~p [s], attempt
                   number ~p~n", [Delay, Attempts]),
111             timer:sleep(Delay*1000),
112             open_socket(min(2*Delay, 8), Attempts+1)
113     end.
114
115
116 open_socket() ->
117     {ok, Addrs} = inet:getifaddrs(),
118     Ipaddr = hd([
119         Addr || {_, Opts} <- Addrs, {addr, Addr} <- Opts,
120         size(Addr) == 4, Addr =/= {127,0,0,1}
121     ]),
122     {ok, Socket} = gen_udp:open(?PORT, [binary, {active, true}, {
           reuseaddr, true}]),
123
124     case Ipaddr of
125         {192, _, _, _} ->
126             hera:logg("[HERA_COM] Connected to private network
                   with IP: ~p~n", [Ipaddr]),
```

```
127             persistent_term:put(multicast, false);
128         {10, _, _, _} ->
129             hera:logg("[HERA_COM] Connected to private network
                    with IP: ~p~n", [Ipaddr]),
130             persistent_term:put(multicast, false);
131         {172, _, _, _} ->
132             hera:logg("[HERA_COM] Connected to mobile hotspot (
                    unicast only) IP: ~p~n", [Ipaddr]),
133             persistent_term:put(multicast, false);
134         _ ->
135             hera:logg("[HERA_COM] Unknown network IP: ~p~n", [
                    Ipaddr]),
136             persistent_term:put(multicast, false)
137     end,
138     hera_subscribe:notify("connected"),
139     Socket.
140
141 loop(Socket) ->
142     receive
143         {udp, _Sock, _IP, _InPortNo, Packet} ->
144             case catch binary_to_term(Packet) of
145                 {'EXIT', _} ->
146                     handle_string_packet(binary_to_list(Packet));
147                 {hera_data, Name, From, Seq, Values} ->
148                     hera_data:store(Name, From, Seq, Values)
149             end,
150             loop(Socket);
151         {send_packet, Packet} ->
152             Multicast_enabled = persistent_term:get(multicast),
153             if
154                 Multicast_enabled ->
155                     gen_udp:send(Socket, ?MULTICAST_ADDR, ?PORT,
                            Packet);
156                 true ->
157                     [gen_udp:send(Socket, IP, Port, Packet) || {_,
                            IP, Port} <- persistent_term:get(devices)]
158             end,
159             loop(Socket);
160         {send_packet_unicast, Name, Packet} ->
161             Devices = persistent_term:get(devices),
162             SelectedDevice = lists:keyfind(Name, 1, Devices),
163             case SelectedDevice of
164                 false -> hera:logg("[HERA_COM] Unregistered Device
                        : ~p~n", [Name]);
165                 {_, IP, Port} -> gen_udp:send(Socket, IP, Port,
                        Packet)
166             end,
167             loop(Socket);
168         {send_packet_unicast, Ip, Port, Packet} ->
```

```
169             gen_udp:send(Socket, Ip, Port, Packet),
170             loop(Socket);
171         _ ->
172             loop(Socket)
173     after 5000 ->
174         case test_connection() of
175             ok ->
176                 loop(Socket);
177             error ->
178                 New_Socket = restart(Socket),
179                 loop(New_Socket)
180         end
181     end.
182
183 test_connection() ->
184     {ok, Addrs} = inet:getifaddrs(),
185     case lists:keyfind("wlan0", 1, Addrs) of
186         {"wlan0", Fields} ->
187             case proplists:get_value(broadaddr, Fields, none) of
188                 none ->
189                     error;
190                 _BroadAddr ->
191                     ok
192             end;
193         _ ->
194             error
195     end.
196
197 restart(Socket) ->
198     hera:logg("[HERA_COM] Lost connection~n", []),
199     gen_udp:close(Socket),
200     hera_subscribe:notify("disconnected"),
201     open_socket(1, 1).
202
203 handle_string_packet(String) ->
204     Tokens = string:tokens(String, ": ,"),
205     hera_subscribe:notify(Tokens).
206
207 %Encodes one value from a double to a half-float
208 enc_hf(Double) ->
209         <<_,_,A,B,C,_,_,_,_,_,_>>=term_to_binary(Double),
210         A2 = (A band 192) bor ((B bsr 2) band 63),
211         B2 = ((B bsl 6) band 192) bor ((C bsr 2) band 63),
212         <<A2,B2>>.
213
214 %Decodes one value from a half-float to a double
215 dec_hf(<<0,0>>) -> 0.0;
216 dec_hf(Half_Float) ->
217         <<X,Y>> = Half_Float,
```

```
218         if
219                 (X band 64) == 0 ->
220                         A = (X band 192) bor 63;
221                 true ->
222                         A = (X band 192)
223         end,
224         B = ((X bsl 2) band 252) bor ((Y bsr 6) band 3),
225         C = ((Y bsl 2) band 252),
226         binary_to_term(<<131,70,A,B,C,0,0,0,0,0>>).
```

D.3 hera_data

```
1  -module(hera_data).
2
3  -behaviour(gen_server).
4
5  -export([start_link/0]).
6  -export([get/1, get/2]).
7  -export([store/4, reset/0]).
8  -export([init/1, handle_call/3, handle_cast/2]).
9
10 -type measure() :: {node(), pos_integer(), hera:timestamp(), [
       number(), ...]}.
11
12 -export_type([measure/0]).
13
14 -record(data, {
15     seq = 0 :: non_neg_integer(),
16     values :: [number(), ...] | undefined,
17     timestamp :: hera:timestamp(),
18     file :: string() | undefined
19 }).
20
21 start_link() ->
22     gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).
23
24
25 -spec get(Name) -> Measures when
26     Name :: atom(),
27     Measures :: [measure()].
28
29 get(Name) ->
30     gen_server:call(?MODULE, {get, Name}).
31
32
33 -spec get(Name, Node) -> [Measure] when
```

```
34      Name :: atom(),
35      Node :: node(),
36      Measure :: measure().
37
38  get(Name, Node) ->
39      gen_server:call(?MODULE, {get, Name, Node}).
40
41
42  -spec store(Name, Node, Seq, Values) -> ok when
43      Name :: atom(),
44      Node :: node(),
45      Seq :: pos_integer(),
46      Values :: [number(), ...].
47
48  store(Name, Node, Seq, Values) ->
49      hera:logg("[HERA_DATA] Storing ~p, ~p, ~p, ~p~n",[Name, Node,
            Seq, Values]),
50      gen_server:cast(?MODULE, {store, Name, Node, Seq, Values}).
51
52  reset() ->
53      gen_server:cast(?MODULE, reset).
54
55  init([]) ->
56      {ok, #{}}.
57
58
59  handle_call({get, Name}, _From, MapData) ->
60      MapMeasure = maps:get(Name, MapData, #{}),
61      L = maps:to_list(MapMeasure),
62      Res = [{Node,S,T,V} || {Node, #data{seq=S,values=V,timestamp=T
            }} <- L],
63      {reply, Res, MapData};
64
65  handle_call({get, Name, Node}, _From, MapData) ->
66      MapMeasure = maps:get(Name, MapData, #{}),
67      Res = if
68          is_map_key(Node, MapMeasure) ->
69              #data{seq=S,values=V,timestamp=T} = maps:get(Node,
                    MapMeasure),
70              [{Node,S,T,V}];
71          true ->
72              []
73      end,
74      {reply, Res, MapData};
75
76  handle_call(_Request, _From, State) ->
77      {reply, ok, State}.
78
79
```

```
80 handle_cast({store, Name, Node, Seq1, L}, MapData) ->
81     MapNode0 = maps:get(Name, MapData, #{}),
82     IsLogger = application:get_env(hera, log_data, false),
83     MapNode1 = if
84         is_map_key(Node, MapNode0) ->
85             MapNode0;
86         IsLogger ->
87             File = file_name(Name, Node),
88             MapNode0#{Node => #data{file=File}};
89         true ->
90             MapNode0#{Node => #data{}}
91     end,
92     Data = maps:get(Node, MapNode1),
93     MapNode2 = case Data of
94         #data{seq=Seq0} when Seq0 < Seq1 ->
95             T = hera:timestamp(),
96             log_data(Data#data.file, {Seq1, T, L}, IsLogger),
97             NewData = Data#data{seq=Seq1,values=L,timestamp=T},
98             maps:put(Node, NewData, MapNode1);
99         _ ->
100            MapNode1
101    end,
102    {noreply, maps:put(Name, MapNode2, MapData)};
103
104 handle_cast(reset, _) ->
105    NewState = #{},
106    {noreply, NewState};
107
108 handle_cast(_Request, State) ->
109    {noreply, State}.
110
111 file_name(Name, Node) ->
112    lists:append(
113        ["measures/", atom_to_list(Name), "_", atom_to_list(Node),
            ".csv"]).
114
115
116 log_data(_, _, false) ->
117    ok;
118
119 log_data(File, {Seq, T, Ms}, true) ->
120    Vals = lists:map(fun(V) -> lists:flatten(io_lib:format("~p", [
       V])) end, Ms),
121    S = string:join(Vals, ","),
122    Bytes = io_lib:format("~p,~p,~s~n", [Seq, T, S]),
123    ok = filelib:ensure_dir("measures/"),
124    ok = file:write_file(File, Bytes, [append]).
```

D.4 hera_kalman module

```
-module(hera_kalman).

-export([predict/3, update/4, extended_predict/3, extended_update/4]).
-export([filter/6, extended_filter/6, extended_control/7]).

%% see https://en.wikipedia.org/wiki/Kalman_filter


%% A kalman filter without control input
filter({X0, P0}, F, H, Q, R, Z) ->
    {Xp, Pp} = predict({X0, P0}, F, Q),
    update({Xp, Pp}, H, R, Z).


predict({X0, P0}, F, Q) ->
    Xp = mat:'*'(F, X0),
    Pp = mat:eval([F, '*', P0, '*  ', F, '+', Q]),
    {Xp, Pp}.


update({Xp, Pp}, H, R, Z) ->
    S = mat:eval([H, '*', Pp, '*  ', H, '+', R]),
    Sinv = mat:inv(S),
    K = mat:eval([Pp, '*  ', H, '*', Sinv]),
    Y = mat:'-'(Z, mat:'*'(H, Xp)),
    X1 = mat:eval([K, '*', Y, '+', Xp]),
    P1 = mat:'-'(Pp, mat:eval([K, '*', H, '*', Pp])),
    {X1, P1}.


%% An extended kalman filter without control input, [X0, P0, Q, R, Z] must be mat matrices, [F, Jf, H, Jh] must be functions
extended_filter({X0, P0}, {F, Jf}, {H, Jh}, Q, R, Z) ->
    % Prediction
    {Xp, Pp} = extended_predict({X0, P0}, {F, Jf}, Q),

    % Update
    extended_update({Xp, Pp}, {H, Jh}, R, Z).

extended_predict({X0, P0}, {F, Jf}, Q) ->
    Xp = F(X0),
    Jfx = Jf(X0),
    Pp = mat:eval([Jfx, '*', P0, '*  ', Jfx, '+', Q]),
    {Xp, Pp}.

```

```
45 extended_update({Xp, Pp}, {H, Jh}, R, Z) ->
46     Jhx = Jh(Xp),
47     S = mat:eval([Jhx, '*', Pp, '*  ', Jhx, '+', R]),
48     Sinv = mat:inv(S),
49     K = mat:eval([Pp, '*  ', Jhx, '*', Sinv]),
50     Y = mat:'-'(Z, H(Xp)),
51     X1 = mat:eval([K, '*', Y, '+', Xp]),
52     P1 = mat:'-'(Pp, mat:eval([K, '*', Jhx, '*', Pp])),
53     {X1, P1}.
54
55 %% Same function as ekf/ with command input
56 extended_control({X0, P0}, {F, Jf}, {H, Jh}, Q, R, Z, U) ->
57     % Prediction
58     Xp = F(X0,U),
59     Jfx = Jf(X0),
60     Pp = mat:eval([Jfx, '*', P0, '*  ', Jfx, '+', Q]),
61
62     % Update
63     Jhx = Jh(Xp),
64     S = mat:eval([Jhx, '*', Pp, '*  ', Jhx, '+', R]),
65     Sinv = mat:inv(S),
66     K = mat:eval([Pp, '*  ', Jhx, '*', Sinv]),
67     Y = mat:'-'(Z, H(Xp)),
68     X1 = mat:eval([K, '*', Y, '+', Xp]),
69     P1 = mat:'-'(Pp, mat:eval([K, '*', Jhx, '*', Pp])),
70     {X1, P1}.
```

D.5 hera_measure module

```
1 -module(hera_measure).
2
3 -export([start_link/2]).
4
5 -type measure_spec() :: #{
6     name := atom(), % measure id
7     iter := pos_integer() | infinity, % number of measures to
          perform
8     sync => boolean(), % must the measure must be synchronized? (
          default: false)
9     timeout => timeout() % min delay between two measures (default
          : 1)
10 }.
11
12 -export_type([measure_spec/0]).
13
14 -callback init(Args :: term()) ->
```

```
15      {ok, State :: term(), Spec :: measure_spec()}.
16
17  -callback measure(State :: term()) ->
18      {ok, Values, NewState} | {undefined, NewState} when
19      Values :: [number(), ...],
20      NewState :: term().
21
22  -record(state, {
23      name :: atom(),
24      sync = false :: boolean(),
25      monitor :: {pid(), reference()} | undefined,
26      timeout = 1 :: timeout(),
27      seq = 1 :: pos_integer(),
28      iter = 1 :: non_neg_integer() | infinity,
29      mod :: module(),
30      mod_state :: term(),
31      sender = undefined :: pid()
32  }).
33
34  -define(record_to_tuplelist(Name, Rec),
35      lists:zip(record_info(fields, Name), tl(tuple_to_list(Rec)))).
36
37  start_link(Module, Args) ->
38      Pid = spawn_link(fun() -> init({Module, Args}) end),
39      {ok, Pid}.
40
41  init({Mod, Args}) ->
42      case Mod:init(Args) of
43          {ok, ModState, Spec} ->
44              L0 = ?record_to_tuplelist(state, #state{}),
45              L1 = lists:map(fun({Key, Val}) -> maps:get(Key, Spec,
                    Val) end, L0),
46              State = list_to_tuple([state|L1]),
47              Seq = init_seq(State#state.name),
48              Sender_Pid = spawn_link(hera_measure_sender, init, [])
                    ,
49              case State#state.sync of
50                  true ->
51                      PidRef = subscribe(State#state.name),
52                      NewState =
53                          State#state{seq=Seq,mod=Mod,mod_state=
                                ModState,monitor=PidRef,sender=
                                Sender_Pid},
54                      loop(NewState, true);
55                  false ->
56                      NewState = State#state{seq=Seq,mod=Mod,
                            mod_state=ModState,sender=Sender_Pid},
57                      loop(NewState, false)
58              end;
```

```
59          {stop, Reason} ->
60              hera:logg("[HERA_MEASURE] Measure not initialized
                    because: ~p~n", [Reason])
61      end.
62
63
64  loop(State, false) ->
65      continue(measure(State));
66  loop(State=#state{monitor={From,Ref}}, true) ->
67      receive
68          {authorized, From} ->
69              NewState = measure(State),
70              From ! {ok, self()},
71              continue(NewState);
72          {'DOWN', Ref, _, _, _} ->
73              PidRef = subscribe(State#state.name),
74              continue(State#state{monitor=PidRef})
75      end.
76
77
78  continue(#state{iter=0}) ->
79      {stop, normal};
80  continue(State) ->
81      timer:sleep(State#state.timeout),
82      loop(State, State#state.sync).
83
84
85  subscribe(Name) ->
86      {ok, Pid} = hera_subscribe:subscribe(Name),
87      Ref = monitor(process, Pid),
88      {Pid, Ref}.
89
90
91  %% return 1 or 1 + the last seq number known among all nodes
92  init_seq(Name) ->
93      {ResL, _} = rpc:multicall(hera_data, get, [Name, node()]),
94      L = lists:filtermap(fun(Res) ->
95          case Res of
96              [{_,Seq,_,_}] -> {true, Seq};
97              _ -> false
98          end
99      end, ResL),
100     lists:max([0|L]) + 1.
101
102
103 measure(State=#state{name=N, mod=M, mod_state=MS, seq=Seq, iter=
        Iter, sender=Sender_Pid}) ->
104     case M:measure(MS) of
105         {undefined, NewMS} ->
```

```
106             State#state{mod_state=NewMS};
107         {ok, Vals=[_|_], NewMS} ->
108             Sender_Pid ! {N, Seq, Vals},
109             NewIter = case Iter of
110                 infinity -> Iter;
111                 _ -> Iter-1
112             end,
113             State#state{seq=Seq+1, iter=NewIter, mod_state=NewMS};
114         {ok, Vals=[_|_], Name, From, NewMS} ->
115             Sender_Pid ! {Name, Seq, From, Vals},
116             NewIter = case Iter of
117                 infinity -> Iter;
118                 _ -> Iter-1
119             end,
120             State#state{seq=Seq+1, iter=NewIter, mod_state=NewMS};
121         {no_share, NewMS} ->
122             NewIter = case Iter of
123                 infinity -> Iter;
124                 _ -> Iter-1
125             end,
126             State#state{seq=Seq+1, iter=NewIter, mod_state=NewMS};
127         {stop, Reason} ->
128             hera:logg("[HERA_MEASURE] Stoping because : ~p~n",[
                Reason]),
129             State#state{seq=Seq+1, iter=0, mod_state=MS}
130     end.
```

D.6 hera_measure_sender module

```
1  -module(hera_measure_sender).
2  -export([init/0]).
3
4  init() ->
5      loop().
6
7  loop() ->
8      receive
9          {Name, Seq, From, Values}->
10             hera_com:send(Name, Seq, From, Values);
11         {Name, Seq, Values}->
12             hera_com:send(Name, Seq, Values);
13         Msg ->
14             hera:logg("[HERA_MEASURE_SENDER] received strange
                   message ~p~n", [Msg])
15     end,
16     loop().
```

D.7 hera_pid_controller module

```
1  -module(hera_pid_controller).
2
3  -export([pid_init/4, pid_init/6]).
4  -export([saturation/2, sign/1]).
5
6  %Controller initialisation without limits on the command and on
       the integral error
7  pid_init(Kp, Ki, Kd, Set_Point) ->
8
9    process_flag(priority, max),
10
11   T0 = erlang:system_time() * 1.0e-9,
12   pid_interface({Kp, Ki, Kd, -1, -1}, {Set_Point, Set_Point}, {0,
        T0, 0}).
13
14 %Complete controller initialisation
15 pid_init(Kp, Ki, Kd, Limit, Int_limit, Set_Point) ->
16   T0 = erlang:system_time() * 1.0e-9,
17   pid_interface({Kp, Ki, Kd, Limit, Int_limit}, {Set_Point,
        Set_Point}, {0, T0, 0}).
18
19 %General case of the controller
20 pid_interface({Kp, Ki, Kd, Limit, Int_limit}, {Set_point,
       Current_input}, {Prev_error, T0, Integral_error}) ->
21         receive
22                 %Exit process
23                 {_, {exit}} ->
24       hera:logg("[HERA_PID] Exiting~n", []);
25
26                 %Parameter modification
27                 {_, {kp, New_Kp}} ->
28                         pid_interface({New_Kp, Ki, Kd, Limit,
                               Int_limit}, {Set_point, Current_input},
                                {Prev_error, T0, Integral_error});
29                 {_, {ki, New_Ki}} ->
30                         pid_interface({Kp, New_Ki, Kd, Limit,
                               Int_limit}, {Set_point, Current_input},
                                {Prev_error, T0, Integral_error});
31                 {_, {kd, New_Kd}} ->
32                         pid_interface({Kp, Ki, New_Kd, Limit,
                               Int_limit}, {Set_point, Current_input},
                                {Prev_error, T0, Integral_error});
33                 {_, {limit, New_Limit}} ->
34                         pid_interface({Kp, Ki, Kd, New_Limit,
                               Int_limit}, {Set_point, Current_input},
                                {Prev_error, T0, Integral_error});
```

```
35                  {_, {int_limit, New_Int_limit}} ->
36                          pid_interface({Kp, Ki, Kd, Limit,
                                New_Int_limit}, {Set_point,
                                Current_input}, {Prev_error, T0,
                                Integral_error});
37
38                  %Setpoint modification
39                  {_, {set_point, New_Set_point}} ->
40                          pid_interface({Kp, Ki, Kd, Limit,
                                Int_limit}, {New_Set_point,
                                Current_input}, {Prev_error, T0,
                                Integral_error});
41
42                  %Get next value
43                  {PID, {input, New_input}} ->
44                          pid_controller_iteration({Kp, Ki, Kd,
                                Limit, Int_limit}, {Set_point,
                                New_input}, {Prev_error, T0,
                                Integral_error}, PID);
45
46                  %Reset integral error
47                  {_, {reset, _}} ->
48                          pid_interface({Kp, Ki, Kd, Limit,
                                Int_limit}, {Set_point, Current_input},
                                 {Prev_error, T0, 0})
49    end.
50
51  %Sends the next value for the command to the process with Pid =
        Output_PID
52  pid_controller_iteration({Kp, Ki, Kd, Limit, Int_limit}, {
        Set_point, Input}, {Prev_error, T0, Integral_error}, Output_PID
        ) ->
53    T1 = erlang:system_time() * 1.0e-9,
54    Dt = T1 - T0,
55
56    Error = Set_point - Input,
57    New_Integral_error = saturation(Integral_error + Error * Dt,
          Int_limit),
58    Derivative_error = (Error - Prev_error) / Dt,
59
60    Command = saturation(Kp * Error + Ki * New_Integral_error + Kd *
          Derivative_error, Limit),
61
62    %Send control to the process with Pid = Output_PID
63    Output_PID ! {self(), {control, Command}},
64
65    pid_interface({Kp, Ki, Kd, Limit, Int_limit}, {Set_point, Input
          }, {Error, T1, New_Integral_error}).
66
```

```
67
68 saturation(Value, Limit) ->
69   if
70     Limit =< 0 -> Value;
71     Value > Limit -> Limit;
72     Value < -Limit -> -Limit;
73     true -> Value
74   end.
75
76 sign(Value) ->
77     if
78         Value < 0 ->
79             -1;
80         true ->
81             1
82     end.
```

D.8 hera_subscribe module

```
1  -module(hera_subscribe).
2
3  -behaviour(gen_server).
4
5  -export([start_link/0, subscribe/1, notify/1]).
6  -export([init/1, handle_call/3, handle_cast/2]).
7
8  start_link() ->
9      gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).
10
11 subscribe(Pid) ->
12     hera:logg("[HERA_SUBSCRIBE] New subscriber ~p~n",[Pid]),
13     gen_server:call(?MODULE, {subscribe, Pid}).
14
15 notify(Msg) ->
16     hera:logg("[HERA_SUBSCRIBE] Notifying ~p~n",[Msg]),
17     gen_server:cast(?MODULE, {notify, Msg}).
18
19 init([]) ->
20     {ok, []}.
21
22 handle_call({subscribe, Pid}, _From, Subscribers) ->
23     {reply, ok, [Pid|Subscribers]}.
24
25 handle_cast({notify, Msg}, Subscribers) ->
26     [Pid ! {hera_notify, Msg} || Pid <- Subscribers],
27     {noreply, Subscribers}.
```

D.9 hera_sup module

```
-module(hera_sup).

-behaviour(supervisor).

-export([start_link/0]).
-export([init/1]).

start_link() ->
    supervisor:start_link(?MODULE, []).

init([]) ->
    SupFlags = #{
        strategy => one_for_one,
        intensity => 6,
        period => 3600
    },
    HeraData = #{
        id => hera_data,
        start => {hera_data, start_link, []}
    },
    HeraCom = #{
        id => hera_com,
        start => {hera_com, start_link, []}
    },
    HeraMeasureSup = #{
        id => hera_measure_sup,
        start => {hera_measure_sup, start_link, []},
        type => supervisor
    },
    HeraSub = #{
    id => hera_subscribe,
    start => {hera_subscribe, start_link, []}
    },
    ChildSpecs = [HeraData, HeraSub, HeraCom, HeraMeasureSup],
    {ok, {SupFlags, ChildSpecs}}.
```

D.10 hera_measure_sup module

```
-module(hera_measure_sup).

-behaviour(supervisor).

-export([start_link/0]).
-export([start_child/2]).
```

```
7
8  -export([init/1]).
9
10 start_link() ->
11     supervisor:start_link({local, ?MODULE}, ?MODULE, []).
12
13 start_child(Module, Args) ->
14     supervisor:start_child(?MODULE, [Module, Args]).
15
16 init([]) ->
17     SupFlags = #{
18         strategy => simple_one_for_one,
19         intensity => 10,
20         period => 60
21     },
22     HeraMeasure = #{
23         id => hera_measure,
24         start => {hera_measure, start_link, []},
25         restart => transient
26     },
27     ChildSpecs = [HeraMeasure],
28     {ok, {SupFlags, ChildSpecs}}.
```

Appendix E

Sensor Software

E.1 The main module

E.1.1 Initial setup

```
-module(sensor).

-behavior(application).

% Callbacks
-export([start/2, stop/1]).

% @private
start(_Type, _Args) ->
    io:format("[SENSOR] start initialization sequence~n"),
    {ok, _} = sensor_sup:start_link(),
    hera_subscribe:subscribe(self()),
    [grisp_led:flash(L, yellow, 500) || L <- [1, 2]],
    grisp:add_device(uart, pmod_maxsonar),

    config(),
    {ok, self()}.

% @private
stop(_State) -> ok.
```

E.1.2 Discovery time functions

```
config() ->
    who_am_i(),
    Id = persistent_term:get(id),
    io:format("[SENSOR] Waiting for start signal ...~n~n"),
    loop_config(Id).

who_am_i() ->
    % Computing sensor id and storing it in persistent data
    {ok, Id} = get_grisp_id(),
    io:format("[SENSOR] sensor id :~p~n",[Id]),
    persistent_term:put(sensor_name, list_to_atom("sensor_" ++
       integer_to_list(Id))),
    persistent_term:put(id, Id),
    await_connection(Id).

await_connection(Id) ->
    % Waiting for HERA to notify succesful connection
    % @param Id : Sensor's Id set by the jumpers (Integer)
    io:format("[SENSOR] WiFi setup starting...~n"),
    receive
        {hera_notify, "connected"} -> % Received when hera_com
           managed to connect to the network
            io:format("[SENSOR] WiFi setup done~n~n"),
            [grisp_led:flash(L, white, 1000) || L <- [1, 2]],
            discover_server(Id)
    after 23000 ->
        io:format("[SENSOR] WiFi setup failed:~n~n"),
        [grisp_led:flash(L, red, 750) || L <- [1, 2]],
        await_connection(Id)
    end.

discover_server(Id) ->
    % Waits forever until the server sends a Ping
    % @param Id : Sensor's Id set by the jumpers (Integer)
    io:format("[SENSOR] Waiting for ping from server~n"),
    receive
        {hera_notify, ["ping", Name, SIp, Port]} -> % Received
           upon server ping reception
            {ok, Ip} = inet:parse_address(SIp),
            IntPort = list_to_integer(Port),
            hera_com:add_device(list_to_atom(Name), Ip, IntPort),
            ack_loop(Id);
        {hera_notify, "disconnected"} -> % Received when hera
           looses WiFi connection
            io:format("~n[SENSOR] Lost connection ...~n"),
            [grisp_led:flash(L, red, 750) || L <- [1, 2]],
            await_connection(Id);
```

```
44          Msg ->
45              io:format("~p~n",[Msg])
46      after 9000 ->
47          io:format("[SENSOR] no ping from server~n~n"),
48          discover_server(Id)
49      end.
50
51  ack_loop(Id) ->
52      % Tries to pair with the server by a Hello -> Ack
53      % @param Id : Sensor's Id set by the jumpers (Integer)
54      send_udp_message(server, "Hello," ++ integer_to_list(Id), "
            UTF8"),
55      receive
56          {hera_notify, ["Ack", _]} -> % Ensures the discovery of
                the sensor by the server
57              io:format("[SENSOR] Received ACK from server~n"),
58              [grisp_led:flash(L, green, 1000) || L <- [1, 2]],
59              ok
60      after 5000 ->
61          ack_loop(Id)
62      end.
63
64  get_grisp_id() ->
65      % Computes the Id of the GRiSP board using the jumpers
66      JMP1 = grisp_gpio:open(jumper_1, #{mode => input}),
67      JMP2 = grisp_gpio:open(jumper_2, #{mode => input}),
68      JMP3 = grisp_gpio:open(jumper_3, #{mode => input}),
69      JMP4 = grisp_gpio:open(jumper_4, #{mode => input}),
70      JMP5 = grisp_gpio:open(jumper_5, #{mode => input}),
71
72      V1 = grisp_gpio:get(JMP1),
73      V2 = grisp_gpio:get(JMP2),
74      V3 = grisp_gpio:get(JMP3),
75      V4 = grisp_gpio:get(JMP4),
76      V5 = grisp_gpio:get(JMP5),
77
78      SUM = (V1) + (V2 bsl 1) + (V3 bsl 2) + (V4 bsl 3) + (V5 bsl 4)
            ,
79      {ok, SUM}.
80
81  send_udp_message(Name, Message, Type) ->
82      % Sends message
83      % @param Name : name of the device to send to (atom)
84      % @param Message : message to be sent (String/Tuple)
85      % @param Type : type of message, can be UTF8 or Binary (String
            )
86      hera_com:send_unicast(Name, Message, Type).
```

E.1.3 Config loop

```
loop_config(Id) ->
    % Main Sensor loop
    % @param Id : Sensor's Id set by the jumpers (Integer)
    receive
        {hera_notify, ["Add_Device", Name, SIp, Port]} ->  % Received at config time to register all used sensors
            add_device(Id, Name, SIp, Port);
        {hera_notify, ["Add_Link", Name, SIp, Port]} ->  % Received at config time to register the propagation links between rooms
            add_link(Id, Name, SIp, Port);
        {hera_notify, ["Init_pos", SPosx, SPosy, SAngle, SRoom]} -> % Register Robot Device initial position
            ack_message("Pos", "robot", Id),
            store_robot_position(Id, 1, SPosx, SPosy, SAngle, SRoom);
        {hera_notify, ["Pos", Ids, Xs, Ys, Hs, As, RoomS]} -> % Received at config time To get all the sensors positions (X-Axis, Y-axis, Height, Angle, Room)
            store_sensor_position(Id, Ids, Xs, Ys, Hs, As, RoomS);
        {hera_notify, ["Room_info", RoomId, TLx, TLy, BRx, BRy]} ->
            store_room_info(Id, RoomId, TLx, TLy, BRx, BRy);
        {hera_notify, ["Start", _]} -> % Received at the end of the configuration to launch the simulation
            start_measures(Id);
        {hera_notify, ["Exit"]} -> % Received when the controller is exited
            io:format("~n[SENSOR] Exit message received~n"),
            reset_state(Id);
        {hera_notify, "disconnected"} -> % Received when hera looses WiFi connection
            io:format("~n[SENSOR] Lost connection, standing by ...~n"),
            [grisp_led:flash(L, red, 1000) || L <- [1, 2]],
            loop_config(Id);
        {hera_notify, ["ping", _, _, _]} -> % Ignore the pings after server discovery
            loop_config(Id);
        {hera_notify, Msg} -> % Unhandled Message
            io:format("[SENSOR] Received unhandled message : ~p~n", [Msg]),
            loop_config(Id);
        Msg -> % Message not from hera_notify
            io:format("[SENSOR] receive strange message : ~p~n",[Msg]),
            loop_config(Id)
```

```
33      end.
34
35  add_device(Id, Name, SIp, SPort) ->
36      % Adds a device to the list of known devices
37      % @param Id : Sensor's Id set by the jumpers (Integer)
38      % @param Name : name of the device to register (String)
39      % @param SIp : IP adress (String)
40      % @param SPort : Port (String)
41      ack_message("Add_device", Name, Id),
42      SelfName = persistent_term:get(sensor_name),
43      case list_to_atom(Name) of
44          SelfName -> % Don't register self
45              ok;
46          OName ->
47              {ok, Ip} = inet:parse_address(SIp),
48              Port = list_to_integer(SPort),
49              hera_com:add_device(OName, Ip, Port)
50      end,
51      loop_config(Id).
52
53  add_link(Id, Name, SIp, SPort) ->
54      % Adds a device to the list of known links to other rooms
55      % @param Id : Sensor's Id set by the jumpers (Integer)
56      % @param Name : name of the device to register (String)
57      % @param SIp : IP adress (String)
58      % @param SPort : Port (String)
59      ack_message("Add_Link", Name, Id),
60      Links = persistent_term:get(links),
61      {ok, Ip} = inet:parse_address(SIp),
62      Port = list_to_integer(SPort),
63      case lists:member({Name, Ip, Port}, Links) of
64          false ->
65              io:format("[SENSOR] Discovered new link : ~p~n", [Name
                    ]),
66              NewLinks = [{Name, Ip, Port} | Links],
67              persistent_term:put(links, NewLinks);
68          _ ->
69              ok
70      end,
71      loop_config(Id).
72
73  store_robot_position(Id, OSeq, SPosx, SPosy, SAngle, SRoom) ->
74      % Stores the initial robot position
75      % @param Id : Sensor's Id set by the jumpers (Integer)
76      % @param SPosx : X axis position (String)
77      % @param SPosY : Y axis position (String)
78      % @param SAngle : Robot angle (String)
79      % @param SRoom : Robot room position (String)
80      Posx = list_to_float(SPosx),
```

```
81      Posy = list_to_float(SPosy),
82      Angle = list_to_float(SAngle),
83      Room = list_to_integer(SRoom),
84      case hera_data:get(robot_pos) of
85          [{_, Seq, _, [_, _, _, _]}] when Seq < OSeq->
86              hera_data:store(robot_pos, robot, OSeq, [Posx, Posy,
                    Angle, Room]),
87              propagate_pos(OSeq, SPosx, SPosy, SAngle, SRoom);
88          [{_, _, _, [_, _, _, _]}] ->
89              ok;
90          [] ->
91              hera_data:store(robot_pos, robot, 1, [Posx, Posy,
                    Angle, Room])
92      end,
93      if
94          OSeq == 1 ->
95              loop_config(Id);
96          true ->
97              loop_run(Id, OSeq)
98      end.
99
100 propagate_pos(Seq, SPosx, SPosy, SAngle, SRoom) ->
101     Msg = "Robot_pos,"++integer_to_list(Seq)++","++SPosx++","++
        SPosy++","++SAngle++","++SRoom,
102     case persistent_term:get(sensor_role) of
103         master ->
104             ok;
105         slave ->
106             [hera_com:send_link(Link, Ip, Port, Msg, "UTF8") || {
                    Link, Ip, Port} <- persistent_term:get(links)]
107     end.
108
109 store_sensor_position(Id, Ids, Xs, Ys, Hs, As, RoomS) ->
110     % Store the position of a sensor
111     % @param Id : Sensor's Id set by the jumpers (Integer)
112     % @param Xs : X axis position (String)
113     % @param Ys : Y axis position (String)
114     % @param Hs : Z axis position (String)
115     % @param As : Angle of sensor (String)
116     % @param Rooms : Room number (String)
117     X = list_to_float(Xs),
118     Y = list_to_float(Ys),
119     H = list_to_float(Hs),
120     A = list_to_integer(As),
121     Room = list_to_integer(RoomS),
122     Device_name = "sensor_" ++ Ids,
123     ack_message("Pos", Device_name, Id),
124     SensorName = list_to_atom(Device_name),
125     case hera_data:get(room, SensorName) of
```

```
126         [{_, _, _, [_]}] ->
127             ok;
128         [] ->
129             hera_data:store(room, SensorName, 1, [Room])
130     end,
131
132     case hera_data:get(pos, SensorName) of
133         [{_, _, _, [_, _, _, _]}] ->
134             ok;
135         [] ->
136             hera_data:store(pos, SensorName, 1, [X, Y, H, A])
137     end,
138
139     %io:format("[SENSOR] Sensor's ~p position : (~p,~p) in room
            n  ~p~n",[ParsedId,X,Y, Room]),
140     loop_config(Id).
141
142 store_room_info(Id, RoomIdS, TLxS, TLyS, BRxS, BRyS) ->
143     % Store the dimension of a room
144     % @param Id : Sensor's Id set by the jumpers (Integer)
145     % @param RoomIdS : Room concerned (String)
146     % @param TLxS : Top left X corner position (String)
147     % @param TLyS : Top left Y corner position (String)
148     % @param BRxS : Bottom right X corner position (String)
149     % @param BRyS : Bottom right Y corner position (String)
150     TLx = list_to_float(TLxS),
151     TLy = list_to_float(TLyS),
152     BRx = list_to_float(BRxS),
153     BRy = list_to_float(BRyS),
154     RoomId = list_to_integer(RoomIdS),
155
156     hera_data:store(room_info, RoomId, 1, [TLx, TLy, BRx, BRy]),
157     ack_message("Room_info", RoomIdS, Id),
158     loop_config(Id).
```

E.1.4 Running Loop

```
loop_run(Id, Num) ->
    receive
        {hera_notify, ["Handshake", OPriority, OTimeClock]} -> % Received from the other sensor in during the sonar sensors role distribution
            resolve_handshake(Id, Num, OPriority, OTimeClock);
        {hera_notify, ["Ok", _]} -> % Received from the other sensor to acknowledge the roles of the sensors
            end_handshake(Id, Num);
        {hera_notify, ["Exit"]} -> % Received when the controller is exited
            io:format("~n[SENSOR] Exit message received~n"),
            reset_state(Id);
        {hera_notify, ["Start", _]} -> % Received at the end of the configuration to launch the simulation
            io:format("[SENSOR] Already started~n"),
            loop_run(Id, Num);
        {hera_notify, "disconnected"} -> % Received when hera looses WiFi connection
            io:format("~n[SENSOR] Lost connection, standing by ...~n"),
            [grisp_led:flash(L, red, 1000) || L <- [1, 2]],
            loop_run(Id, Num);
        {hera_notify, ["ping", _, _, _]} -> % Ignore the pings after server discovery
            loop_run(Id, Num);
        {hera_notify, ["Clock", _]} -> % Received to update the sonar clock
            tick(Id, Num);
        {hera_notify, Msg} -> % Unhandled Message
            io:format("[SENSOR] Received unhandled message : ~p~n", [Msg]),
            loop_run(Id, Num);
        Msg -> % Message not from hera_notify
            io:format("[SENSOR] received strange message : ~p~n",[Msg]),
            loop_run(Id, Num)
    end.

tick(Id, Num) ->
    Pid = persistent_term:get(sonar_sensor),
    Pid ! clock,
    loop_run(Id, Num).
```

E.1.5 Room assembly

```
start_measures(Id) ->
    % Launch all the hera_measure modules to gather data
    % @param Id : Sensor's Id set by the jumpers (Integer)
    io:format("====================================~n"),
    io:format("~n~n[SENSOR] Start received, starting the computing phase~n"),
    find_other_sensor(),
    {ok, Sonar_Pid} = hera:start_measure(sonar_sensor, []),
    persistent_term:put(sonar_sensor, Sonar_Pid),

    [grisp_led:color(L, green) || L <- [1, 2]],
    loop_run(Id, 0).

find_other_sensor() ->
    % Sets up the affiliation with the room's sensor
    timer:sleep(300),
    SensName = persistent_term:get(sensor_name),
    case hera_data:get(room, SensName) of
      [{_, _, _, [Room]}] ->
        case get_Osensor(Room) of
            [H|_] -> % Multiple sensors in a room
                io:format("[SENSOR] Other sensor is : ~p~n", [H]),
                persistent_term:put(osensor, H),
                ok;
            _ -> % No other sensor
                io:format("[SENSOR] No other sensor in the room~n"),
                {error, no_other_sensor}
        end;

      Msg ->
        io:format("[SENSOR] Error in getting sensor pos ~p~n",[Msg]),
        timer:sleep(500),
        find_other_sensor()
    end.

get_Osensor(Room) ->
    % Finds the other sensors in the current room
    % @param Room : Room to set (Integer)
    Devices = persistent_term:get(devices),
    lists:foldl(
        fun({Name, _, _}, Acc) ->
            case Name of
                _ ->
                    case hera_data:get(room, Name) of
```

```
44                          [{_, _, _, [ORoom]}] when Room =:= ORoom
                                ->
45                                  %io:format("[SENSOR] Sens : ~p is
                                        in the same room as this sensor
                                        ~n", [Name]),
46                                  [Name | Acc];
47                          _ ->
48                              Acc
49                      end
50              end
51          end,
52          [],
53          Devices
54      ).
55
56  resolve_handshake(Id, Num, OPriority, OTimeClock) ->
57      % Sends a message with the informations concerning the sensors
            role definition
58      % @param Id : Sensor's Id set by the jumpers (Integer)
59      % @param OPriority : Other sensor's random priority (String)
60      % @param OTimeclock : Other sensor's time clock (String)
61      Pid = persistent_term:get(sonar_sensor, none),
62      case Pid of
63          none ->
64              io:format("[SENSOR] Error : Sonar sensor has not
                    spawned~n"),
65              loop_run(Id, Num);
66          _ ->
67              %io:format("[SENSOR] Sending handshake informations~n
                    "),
68              Pid ! {handshake, list_to_integer(OPriority),
                    list_to_integer(OTimeClock)},
69              loop_run(Id, Num)
70      end.
71
72  end_handshake(Id, Num)->
73      % Sends a ok message to signify the end of the handshake
            procedure
74          % @param Id : Sensor's Id set by the jumpers (Integer)
75          Pid = persistent_term:get(sonar_sensor, none),
76          case Pid of
77              none ->
78                  io:format("[SENSOR] Error : Sonar sensor has not
                        spawned~n"),
79                  loop_run(Id, Num);
80              _ ->
81                  %io:format("[SENSOR] Sending handshake ok~n"),
82                  Pid ! {ok, role},
83                  loop_run(Id, Num)
```

```
84          end.
85
86  ack_message(Message, Device, Id) ->
87      Msg = "Ack," ++ Message ++ "," ++ Device ++ "," ++
            integer_to_list(Id),
88      send_udp_message(server, Msg, "UTF8").
```

E.2 sonar_measure

E.2.1 Module Initialisation

```
1   -module(sonar_measure).
2
3   -behavior(hera_measure).
4
5   -define(ROBOT_HEIGHT, 23).
6   -define(LPF_ALPHA, 0.2).
7   -define(SMOOTHING_WINDOW, 3).
8   -define(HAMPEL_WINDOW, 7).
9   -define(N_SIG, 2.0).
10
11  -export([init/1, measure/1]).
12
13  init(_Args) ->
14      get_sensor_role(),
15      timer:sleep(200),
16      io:format("~n[SONAR_MEASURE] Starting measurements~n"),
17      State = #{
18          seq => get_init_seq(),
19          last_measure => none,
20          hampel_buffer => [],
21          smoothing_buffer => [],
22          n_sig => ?N_SIG
23      },
24      {ok, State, #{name=>sonar_measure, iter=>infinity}}.
25
26  get_sensor_role() ->
27      case persistent_term:get(osensor, none) of
28          none -> % Is alone in a room
29              io:format("[SONAR_MEASURE] No other sensor, sensor is
                    master~n"),
30              persistent_term:put(sensor_role, master);
31          Osensor -> % Start Handshake
32              case persistent_term:get(sensor_role, none) of
33                  none -> % Classical sensor bootstrap
34                      {ok, Priority} = get_rand_num(),
```

```
35                      TimeClock = erlang:monotonic_time(millisecond)
                            ,
36                      role_handshake(Osensor, Priority, TimeClock);
37                  _ ->  % Case where the sonar measure module
                        crashed and was reloaded (need to find the
                        latest seq number)
38                      io:format("[SONAR_MEASURE] Recovering from
                            crash"),
39                      ok
40              end
41      end.
42
43  role_handshake(Osensor, Priority, TimeClock) ->
44      hera_com:send_unicast(Osensor, "Handshake," ++ integer_to_list
            (Priority) ++ "," ++ integer_to_list(TimeClock), "UTF8"),
45      receive
46          {handshake, OPriority, _} ->
47              if
48                  Priority > OPriority ->
49                      io:format("[SONAR_MEASURE] Local priority
                            higher, sensor role : MASTER~n"),
50                      wait_ack(Osensor),
51                      Clock_Pid = spawn(clock_ticker, init, [
                            TimeClock]),
52                      persistent_term:put(clock, Clock_Pid),
53                      persistent_term:put(sensor_role, master);
54                  Priority < OPriority ->
55                      io:format("[SONAR_MEASURE] External priority
                            higher, sensor role : SLAVE~n"),
56                      wait_ack(Osensor),
57                      persistent_term:put(sensor_role, slave);
58                  true ->
59                      io:format("[SONAR_MEASURE] Priority collision,
                            retrying~n"),
60                      {ok, New_Priority} = get_rand_num(),
61                      role_handshake(Osensor, New_Priority,
                            TimeClock)
62              end;
63          {ok, role} ->
64              ok
65      after 500 ->
66          role_handshake(Osensor, Priority, TimeClock)
67      end.
68
69  wait_ack(Osensor) ->
70      hera_com:send_unicast(Osensor, "Ok,role", "UTF8"),
71      receive
72          {ok, _} -> ok;
73          _ -> wait_ack(Osensor)
```

```
74      after 500 ->
75          wait_ack(Osensor)
76      end.
77
78  get_init_seq() ->
79      SensorName = persistent_term:get(sensor_name),
80      case hera_data:get(distance, SensorName) of
81          [{_, Seq, _, [_]}] -> Seq +1;
82          _ -> 1
83      end.
84
85  get_rand_num() ->
86      % Returns a random number between 0 and 2000
87      persistent_term:get(id),
88      Seed = {erlang:monotonic_time(), erlang:unique_integer([
            positive]), erlang:phash2(node())},
89      rand:seed(exsplus, Seed),
90      {ok, rand:uniform(3000)}.
```

E.2.2 Measuring loop

```
1   measure(State) ->
2       receive
3           clock ->
4               #{
5                   seq := Seq,
6                   last_measure := _,
7                   hampel_buffer := _,
8                   smoothing_buffer := _,
9                   n_sig := _
10              } = State,
11              [grisp_led:color(L, green) || L <- [1, 2]],
12
13              SensorName = persistent_term:get(sensor_name),
14              {Measure, LPFMeasure, Hampel_buffer, Smoothing_buffer}
                    = get_measure(State),
15              hera_com:send_unicast(server, "Distance,"++
                   float_to_list(Measure)++","++atom_to_list(
                   SensorName),"UTF8"),
16
17              hera_data:store(distance, SensorName, Seq, [Measure]),
18
19              NewState = State#{
20                  seq => Seq+1,
21                  last_measure => LPFMeasure,
22                  hampel_buffer => Hampel_buffer,
23                  smoothing_buffer => Smoothing_buffer,
```

```
24                  n_sig := ?N_SIG
25              },
26              {ok, [Measure], distance, SensorName, NewState}
27      after 1000 ->
28          [grisp_led:color(L, blue) || L <- [1, 2]],
29          {undefined, State}
30      end.
31
32  get_measure(State) ->
33      % Get the Pmod Maxsonar measure and transform it into the
            right format
34      % @param State : the internal state of the module (tuple)
35      % @param SensorName : the name of the current sensor (atom)
36
37      Dist_inch = pmod_maxsonar:get(),
38      Dist_cm = Dist_inch * 2.54,
39      SensorName = persistent_term:get(sensor_name),
40      LPF_filtered = low_pass_filter(Dist_cm, State),
41      {Hampel_Measure, Hampel_buffer} = hampel_filter(LPF_filtered,
            State),
42      {Smoothed_measure, Smoothing_buffer} = smooth_measure(
            Hampel_Measure, State),
43
44      case get_ground_distance(SensorName, Smoothed_measure) of
45          {ok, Ground_measure} ->
46              {Ground_measure, LPF_filtered, Hampel_buffer,
                    Smoothing_buffer};
47          {stop, cannot_get_height} ->
48              {stop, cannot_get_height}
49      end.
50
51  get_ground_distance(SensorName, D) ->
52      % Uses the basic pythagorian formula to transform the distance
            based on the sensor's height
53      % @param State : the internal state of the module (tuple)
54      % @param SensorName : the name of the current sensor (atom)
55      % @param D : Sonar measure in cm (integer)
56
57      case hera_data:get(pos, SensorName) of
58          [{_, _, _, [_ , _, H, _]}] ->
59              Height_diff = (H*100)-?ROBOT_HEIGHT,
60              if
61                  H*100 > ?ROBOT_HEIGHT ->
62                      if
63                          D > Height_diff ->
64                              Ground_measure = math:sqrt(math:pow(D,
                                    2) - math:pow(Height_diff, 2)); %
                                Taking the height of the sonar into
                                    account
```

```
65                          true ->
66                              Ground_measure = Height_diff
67                      end;
68                  true ->
69                      Ground_measure = D % The robot is bigger than
                           the sensor's height, no need for correction
70              end,
71
72              {ok, Ground_measure};
73          Msg ->
74              io:format("[SONAR_MEASURE] Cannot get sensor height : 
                    ~p~n",[Msg]),
75              {stop, cannot_get_height}
76      end.
77
78  low_pass_filter(Ground_measure, State) ->
79      #{
80        seq := _,
81        last_measure := Last,
82        hampel_buffer := _,
83        smoothing_buffer := _,
84        n_sig := _
85      } = State,
86
87      case Last of
88        none -> Ground_measure;
89        _ -> ?LPF_ALPHA * Last + (1-?LPF_ALPHA)*Ground_measure
90      end.
91
92  hampel_filter(Measure, State) ->
93      #{
94        seq := _,
95        last_measure := _,
96        hampel_buffer := BufH,
97        smoothing_buffer := _,
98        n_sig := N_sig
99      } = State,
100
101     New_BufH = append_buf(BufH, Measure, 2*?HAMPEL_WINDOW+1),
102     Buffer_Median  = median(New_BufH),
103     MAD = median([abs(X - Buffer_Median) || X <- New_BufH]),
104     if
105         abs(Measure - Buffer_Median) > N_sig * MAD ->
106             {Buffer_Median, New_BufH};
107         true ->
108             {Measure, New_BufH}
109     end.
110
111 smooth_measure(Measure, State) ->
```

```
112     #{
113       seq := _,
114       last_measure := _,
115       hampel_buffer := _,
116       smoothing_buffer := BufS,
117       n_sig := _
118     } = State,
119
120     New_BufS = append_buf(BufS, Measure, ?SMOOTHING_WINDOW),
121     {median(New_BufS), New_BufS}.
122
123 append_buf(Buf, X, Len) ->
124     Buf2 = lists:append(Buf, [X]),
125     Excess = length(Buf2) - Len,
126     case Excess > 0 of
127       true  -> lists:sublist(Buf2, Excess+1, Len);
128       false -> Buf2
129     end.
130
131 median(List) when List =/= [] ->
132     S = lists:sort(List),
133     L = length(S),
134     case L rem 2 of
135       1 -> lists:nth((L+1) div 2, S);
136       0 ->
137         A = lists:nth(L div 2, S),
138         B = lists:nth(L div 2+1, S),
139         (A + B)/2.0
140     end.
```

E.3 clock_ticker module

```
1  -module(clock_ticker).
2
3  -export([init/1]).
4
5  -define(TIMESLOT_SIZE, 300).
6  -define(STARTUP_MARGIN, 0.11).
7
8  init(TimeClock) ->
9      io:format("[CLOCK_TICKER] Starting ...~n"),
10     process_flag(priority, max),
11     Osensor = persistent_term:get(osensor),
12     Sonar_Pid = persistent_term:get(sonar_measure),
13     SensName = persistent_term:get(sensor_name),
14     {Ax, Ay, Bx, By} = get_sensor_room(SensName),
```

```
15      loop(TimeClock, Sonar_Pid, Osensor, {Ax, Ay, Bx, By}, 1).
16
17  loop(TimeClock, Sonar_Pid, Osensor, RoomInfo, Count)->
18      Offset = (Count * ?TIMESLOT_SIZE) div 2,
19      Next_measure = TimeClock + Offset,
20
21      case is_in_room(RoomInfo) of
22          true ->
23              io:format("[CLOCK] ROBOT IN ROOM~n"),
24              case Count rem 2 of %determine who measures (slave or
                    master)
25                  0 ->
26                      case wait_for_time(Next_measure) of
27                          ok ->
28                              hera_com:send_unicast(server, "Clock,"
                                    ++ integer_to_list(Count) ++ ","
                                   ++ integer_to_list(Next_measure), "
                                   UTF8"),
29                              Sonar_Pid ! clock;
30                          skip -> loop(TimeClock, Sonar_Pid, Osensor
                                , RoomInfo, Count + 1)
31                      end;
32                  1 ->
33                      case wait_for_time(Next_measure) of
34                          ok -> hera_com:send_unicast(Osensor, "
                               Clock," ++ integer_to_list(Count), "
                               UTF8");
35                          skip -> loop(TimeClock, Sonar_Pid, Osensor
                                , RoomInfo, Count + 1)
36                      end
37              end;
38          false ->
39              case wait_for_time(Next_measure) of
40                  ok -> ok;
41                  skip -> loop(TimeClock, Sonar_Pid, Osensor,
                        RoomInfo, Count + 1)
42              end,
43              ok
44      end,
45
46      loop(TimeClock, Sonar_Pid, Osensor, RoomInfo, Count + 1).
47
48  get_sensor_room(SensName) ->
49      case hera_data:get(room, SensName) of
50          [{_, _, _, [Room]}] ->
51              case hera_data:get(room_info, Room) of
52                  [{_, _, _, [Ax, Ay, Bx, By]}] ->
53                      {Ax, Ay, Bx, By};
54                  [] ->
```

```
55                      io:format("[CLOCK] Error in pos determination~
                            n"),
56                      {0, 0, 0, 0}
57              end;
58          [] ->
59              io:format("[CLOCK] Error in room determination~n"),
60              {0, 0, 0, 0}
61      end.
62
63  is_in_room({Ax, Ay, Bx, By}) ->
64      case hera_data:get(robot_pos, robot) of
65          [{_, _, _, [Xpos, Ypos, _, _]}] when ((Ax-?STARTUP_MARGIN
                < Xpos andalso Xpos < Bx + ?STARTUP_MARGIN) andalso (Ay
                 - ?STARTUP_MARGIN < Ypos andalso Ypos < By + ?
                STARTUP_MARGIN))->
66              true;
67          [{_, _, _, [_Xpos, _Ypos, _, _]}] ->
68              false;
69          _ ->
70              false
71      end.
72
73  wait_for_time(Next_measure) ->
74      Now = erlang:monotonic_time(millisecond),
75      Time_to_wait = Next_measure - Now,
76
77      if
78          Time_to_wait > 0 ->
79              timer:sleep(Time_to_wait),
80              ok;
81          true ->
82              skip
83      end.
```

Appendix F

Balancing Robot software

F.1 Balancing_robot module

```
-module(balancing_robot).

-behavior(application).

-export([start/2, stop/1]).


start(_Type, _Args) ->
    {ok, Supervisor} = balancing_robot_sup:start_link(),
    [grisp_led:flash(L, yellow, 500) || L <- [1, 2]],
        _ = grisp:add_device(spi2, pmod_nav),
    pmod_nav:config(acc, #{odr_g => {hz,238}}),
    numerl:init(),
    timer:sleep(2000),
    spawn(stability_layer, robot_init, []),
    hera_subscribe:subscribe(self()),
    config(),
    loop_config(),
    {ok, Supervisor}.

stop(_State) -> ok.



config() ->
    persistent_term:put(name, list_to_atom("robot")),
    await_connection(),
    io:format("[ROBOT] Waiting for start signal ...~n~n").

await_connection() ->
```

```
31      % Waiting for HERA to notify succesful connection
32
33      io:format("[ROBOT] WiFi setup starting...~n"),
34      receive
35          {hera_notify, "connected"} -> % Received when hera_com
                managed to connect to the network
36              io:format("[ROBOT] WiFi setup done~n~n"),
37              grisp_led:flash(2, white, 1000),
38              discover_server()
39      after 18000 ->
40          io:format("[ROBOT] WiFi setup failed:~n~n"),
41          grisp_led:flash(2, red, 750),
42          await_connection()
43      end.
44
45  discover_server() ->
46      % Waits forever until the server sends a Ping
47      io:format("[ROBOT] Waiting for ping from server~n"),
48      receive
49          {hera_notify, ["ping", Name, SIp, Port]} -> % Received
                upon server ping reception
50              {ok, Ip} = inet:parse_address(SIp),
51              IntPort = list_to_integer(Port),
52              hera_com:add_device(list_to_atom(Name), Ip, IntPort),
53              ack_loop()
54      after 9000 ->
55          io:format("[ROBOT] no ping from server~n~n"),
56          discover_server()
57      end.
58
59  ack_loop() ->
60      % Tries to pair with the server by a Hello -> Ack
61      % @param Id : Sensor's Id set by the jumpers (Integer)
62      send_udp_message(server, "Hello,robot", "UTF8"),
63      receive
64          {hera_notify, ["Ack", _]} -> % Ensures the discovery of
                the sensor by the server
65              io:format("[ROBOT] Received ACK from server~n"),
66              [grisp_led:flash(L, green, 1000) || L <- [1, 2]],
67              ok
68      after 5000 ->
69          ack_loop()
70      end.
71
72  send_udp_message(Name, Message, Type) ->
73      % Sends message
74      % @param Name : name of the device to send to (atom)
75      % @param Message : message to be sent (String/Tuple)
```

```
76      % @param Type : type of message, can be UTF8 or Binary (String
            )
77      hera_com:send_unicast(Name, Message, Type).
78
79
80
81  loop_config() ->
82      receive
83          {hera_notify, ["Add_Device", Name, SIp, Port]} ->  %
                Received at config time to register all used sensors
84              add_device(Name, SIp, Port);
85          {hera_notify, ["Init_pos", SPosx, SPosy, SAngle, SRoom]}
                -> % Register Robot Device initial position
86              store_robot_position(SPosx, SPosy, SAngle, SRoom);
87          {hera_notify, ["Pos", Ids, Xs, Ys, Hs, As, RoomS]} -> %
                Received at config time To get all the sensors
                positions (X-Axis, Y-axis, Height, Angle, Room)
88              store_sensor_position(Ids, Xs, Ys, Hs, As, RoomS);
89          {hera_notify, ["Room_info", RoomId, TLx, TLy, BRx, BRy]}
                ->
90              store_room_info(RoomId, TLx, TLy, BRx, BRy);
91          {hera_notify, ["Start", _]} -> % Received at the end of
                the configuration to launch the simulation
92              start_measures();
93          {hera_notify, ["Exit"]} -> % Received when gracefully
                exited the controller
94              io:format("~n[ROBOT] Exit message received~n"),
95              reset_state();
96          {hera_notify, ["ping", _, _, _]} -> % Ignore the pings
                after server discovery
97              loop_config();
98          {hera_notify, Msg} -> % Unhandled Message
99              io:format("[ROBOT] Received unhandled message : ~p~n",
                    [Msg]),
100             loop_config();
101         Msg -> % Message not from hera_notify
102             io:format("[ROBOT] receive strange message : ~p~n",[
                    Msg]),
103             loop_config()
104     end.
105
106 loop_run() ->
107     receive
108         {hera_notify, ["Start", _]} -> % Received at the end of
                the configuration to launch the simulation
109             io:format("~n[ROBOT] Already started~n"),
110             loop_run();
111         {hera_notify, ["Exit"]} -> % Received when gracefully
                exited the controller
```

```
112             io:format("~n[ROBOT] Exit message received~n"),
113             reset_state();
114         {hera_notify, ["ping", _, _, _]} -> % Ignore the pings
                after server discovery
115             loop_run();
116         {hera_notify, Msg} -> % Unhandled Message
117             io:format("[ROBOT] Received unhandled message : ~p~n",
                    [Msg]),
118             loop_run();
119         Msg -> % Message not from hera_notify
120             io:format("[ROBOT] receive strange message : ~p~n",[
                    Msg]),
121             loop_run()
122     end.
123
124
125
126 add_device(Name, SIp, SPort) ->
127     % Adds a device to the list of known devices
128     % @param Id : Sensor's Id set by the jumpers (Integer)
129     % @param Name : name of the device to register (String)
130     % @param SIp : IP adress (String)
131     % @param SPort : Port (String)
132     ack_message("Add_device", Name),
133     case list_to_atom(Name) of
134         robot -> % Don't register self
135             ok;
136         OName ->
137             {ok, Ip} = inet:parse_address(SIp),
138             Port = list_to_integer(SPort),
139             hera_com:add_device(OName, Ip, Port)
140     end,
141     loop_config().
142
143 store_robot_position(SPosx, SPosy, SAngle, SRoom) ->
144     % Stores the initial robot position
145     % @param SPosx : X axis position (String)
146     % @param SPosY : Y axis position (String)
147     % @param SAngle : Robot angle (String)
148     % @param SRoom : Robot room position (String)
149     Posx = list_to_float(SPosx),
150     Posy = list_to_float(SPosy),
151     Angle = list_to_float(SAngle),
152     Room = list_to_integer(SRoom),
153     ack_message("Pos", "robot"),
154     case hera_data:get(robot_pos) of
155         [{_, _, _, [_, _, _, _]}] ->
156             ok;
157         [] ->
```

```
158             hera_data:store(robot_pos, robot, 1, [Posx, Posy,
                    Angle, Room])
159     end,
160     loop_config().
161
162 store_sensor_position(Ids, Xs, Ys, Hs, As, RoomS) ->
163     % Store the position of a sensor
164     % @param Id : Sensor's Id set by the jumpers (Integer)
165     % @param Xs : X axis position (String)
166     % @param Ys : Y axis position (String)
167     % @param Hs : Z axis position (String)
168     % @param As : Angle of sensor (String)
169     % @param Rooms : Room number (String)
170     X = list_to_float(Xs),
171     Y = list_to_float(Ys),
172     H = list_to_float(Hs),
173     A = list_to_integer(As),
174     Room = list_to_integer(RoomS),
175     Device_name = "sensor_" ++ Ids,
176     ack_message("Pos", Device_name),
177     SensorName = list_to_atom(Device_name),
178     case hera_data:get(room, SensorName) of
179         [{_, _, _, [_]}] ->
180             ok;
181         [] ->
182             hera_data:store(room, SensorName, 1, [Room])
183     end,
184
185     case hera_data:get(pos, SensorName) of
186         [{_, _, _, [_, _, _, _]}] ->
187             ok;
188         [] ->
189             hera_data:store(pos, SensorName, 1, [X, Y, H, A])
190     end,
191
192     loop_config().
193
194 store_room_info(RoomIdS, TLxS, TLyS, BRxS, BRyS) ->
195     % Store the dimension of a room
196     % @param Id : Sensor's Id set by the jumpers (Integer)
197     % @param RoomIdS : Room concerned (String)
198     % @param TLxS : Top left X corner position (String)
199     % @param TLyS : Top left Y corner position (String)
200     % @param BRxS : Bottom right X corner position (String)
201     % @param BRyS : Bottom right Y corner position (String)
202     TLx = list_to_float(TLxS),
203     TLy = list_to_float(TLyS),
204     BRx = list_to_float(BRxS),
205     BRy = list_to_float(BRyS),
```

```
206     RoomId = list_to_integer(RoomIdS),
207
208     hera_data:store(room_info, RoomId, 1, [TLx, TLy, BRx, BRy]),
209     ack_message("Room_info", RoomIdS),
210     loop_config().
211
212 start_measures() ->
213     % Launch all the hera_measure modules to gather data
214     % @param Id : Sensor's Id set by the jumpers (Integer)
215     io:format("~n~n[ROBOT] Start received, starting the computing
            phase~n"),
216     {ok, Position_Pid} = hera:start_measure(position_layer, []),
217     persistent_term:put(position_layer, Position_Pid),
218     [grisp_led:color(L, green) || L <- [1, 2]],
219     loop_run().
220
221 reset_state() ->
222     % Kills all hera_measures modules, resets all data and jump
            back to server discovery
223     % @param Id : Sensor's Id set by the jumpers (Integer)
224     exit_module(position_layer),
225
226     timer:sleep(500),
227     reset_data(),
228
229     grisp_led:flash(2, white, 1000),
230     grisp_led:flash(1, green, 1000),
231
232     discover_server(),
233     io:format("[ROBOT] Waiting for start signal ...~n~n"),
234     loop_config().
235
236 reset_data() ->
237     % Delete all config dependent and hera_measures data
238
239     persistent_term:erase(position_layer),
240     hera_com:reset_devices(),
241     hera_data:reset(),
242     io:format("[ROBOT] Data resetted~n~n~n~n"),
243
244 exit_module(Name) ->
245     % Kills a module stored in persistent term
246     % @param Name : the name of the module (atom)
247     case persistent_term:get(Name, none) of
248         none ->
249             io:format("[ROBOT] module doesn't exist : ~p~n", [Name
                    ]),
250             ok;
251         Pid ->
```

```
252             exit(Pid, shutdown)
253     end.
254
255
256 ack_message(Message, Device) ->
257     % Used to send the acknowledgment message to the controller.
258     % @param Message : The initial type message received by the
            robot (String)
259     % @param Device : The name of the device concerned by the
            message (String)
260     Msg = "Ack," ++ Message ++ "," ++ Device ++ ",robot",
261     send_udp_message(server, Msg, "UTF8").
```

F.2 Position_layer module

```
1  -module(position_layer).
2
3  -behavior(hera_measure).
4
5  -export([init/1, measure/1]).
6
7
8
9  -define(VAR_Q, 0.0002).
10 -define(VAR_R, 0.15).
11
12
13 -define(VAR_P, 0.0002).
14 -define(VAR_S, 0.0075).
15
16 -define(RAD_TO_DEG, 180.0/math:pi()).
17 -define(DEG_TO_RAD, math:pi()/180.0).
18
19
20 init(_Args) ->
21     timer:sleep(1000),
22     io:format("~n[KALMAN_MEASURE] Starting measurements~n"),
23     calibrate_speed(),
24     State = #{
25         t0 => erlang:system_time()/1.0e6,
26         x_pos => mat:matrix([[0],[0]]),
27         p_pos => mat:diag([1,1]),
28         x_or => mat:matrix([[1],[0],[0],[0]]),
29         p_or => mat:diag([1,1,1,1]),
30         seq => 2,
31         seqS1 => 0,
```

```
32          seqS2 => 0
33      },
34
35      {ok, State, #{
36          name => kalman_measure,
37          iter => infinity,
38          timeout => 100
39      }}.
40
41  measure(State) ->
42      #{
43          t0 := T0,
44          x_pos := Xpos,
45          p_pos := Ppos,
46          x_or := Xor,
47          p_or := Por,
48          seq  := Seq,
49          seqS1 := SeqS1,
50          seqS2 := SeqS2
51
52      } = State,
53
54      case hera_data:get(robot_pos, robot) of
55          [{_, _, _, [_OldX,_OldY, _OldAngle, OldRoom]}] ->
56
57
58
59              {V_mes_mm,_} = i2c_read(),
60              V_mes = V_mes_mm / 100,
61
62
63              T1 = erlang:system_time()/1.0e6,
64
65
66              %%%%%% Kalman Orientation %%%%%%
67
68              {Xor1,Por1} = kalman_orientation(Xor, Por, T1, T0),
69              Offset = 12 * ?DEG_TO_RAD,
70              Theta_mes = - quat_to_yaw(normalize_quat(mat:to_array(
                    Xor1))) - Offset,
71
72              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
73
74
75              Q  = mat:diag([?VAR_P, ?VAR_P]),
76
77              % Fonction de transition f(x)
78              Dt = (T1 - T0) / 1000.0,
79
```

```
80              F = fun(X) ->
81                  [Yc, Zc] = mat:to_array(X),
82
83                  Yp = Yc - V_mes *  math:cos(Theta_mes)* Dt,
84                  %io:format("ThetaC : ~p et cos(thetaC) : ~p ~n ",[
                        Thetac,math:cos(Thetac)]),
85                  Zp = Zc - V_mes * math:sin(Theta_mes) * Dt,
86                  mat:matrix([[Yp], [Zp]])
87              end,
88
89              % Jacobienne de f
90              Jf = fun(_X) ->
91                  mat:matrix([
92                  [1, 0],
93                  [0, 1]
94                  ])
95              end,
96
97              {Xpred,Ppred} = hera_kalman:extended_predict({Xpos,
                    Ppos}, {F, Jf}, Q),
98
99              case get_new_robot_pos(OldRoom) of
100                 {no_intersection, { _, _}} ->
101                     [Xf, Yf] = mat:to_array(Xpred),
102                     ThetaDegrees = Theta_mes * ?RAD_TO_DEG,
103                     NewState = #{
104                         t0   => T1,
105                         x_pos => Xpred,
106                         p_pos => Ppred,
107                         x_or => Xor1,
108                         p_or => Por1,
109                         seq  => Seq +1,
110                         seqS1 => SeqS1,
111                         seqS2 => SeqS2
112                     };
113
114
115
116                 {{X_mes, Y_mes},{Seqsensor1,Seqsensor2}} ->
117                     case {SeqS1 < Seqsensor1 andalso SeqS2 <
                            Seqsensor2} of
118                         {true} ->
119
120                             Z = mat:matrix([[X_mes],[Y_mes]]),
121                             io:format("Valeur␣re u␣sonar,␣~p,␣~p␣
                                ~n", [X_mes,Y_mes]),
122
123                             R  = mat:diag([?VAR_S, ?VAR_S]),
124
```

```
125
126                     % Fonction de mesure h(x) = x
127                     H = fun(X) ->
128                         [Xc, Yc] = mat:to_array(X),
129
130                         mat:matrix([[Xc],[Yc]])
131                     end,
132                     Jh = fun(_X) ->
133                         mat:matrix([
134                         [1,0],
135                         [0,1]
136                         ])
137                     end,
138                     {Xnew, Pnew} = hera_kalman:
                            extended_update({Xpred, Ppred}, {H,
                             Jh}, R, Z),
139
140                     [Xf, Yf] = mat:to_array(Xnew),
141                     ThetaDegrees =  Theta_mes * ?
                            RAD_TO_DEG,
142
143                     NewState = #{
144                         t0   => T1,
145                         x_pos => Xnew,
146                         p_pos => Pnew,
147                         x_or => Xor1,
148                         p_or => Por1,
149                         seq  => Seq +1,
150                         seqS1 =>Seqsensor1,
151                         seqS2 =>Seqsensor2
152
153                     };
154                 {false} ->
155
156                     [Xf, Yf] = mat:to_array(Xpred),
157                     ThetaDegrees = Theta_mes * ?RAD_TO_DEG
                            ,
158                     %io:format("False : ~p ~n", [
                            ThetaDegrees]),
159
160                     NewState = #{
161                         t0   => T1,
162                         x_pos => Xpred,
163                         p_pos => Ppred,
164                         x_or => Xor1,
165                         p_or => Por1,
166                         seq  => Seq +1,
167                         seqS1 =>SeqS1,
168                         seqS2 =>SeqS2
```

```
169
170                             }
171                     end
172
173             end,
174
175             io:format("pos␣:␣~p,␣~p␣:␣~p,␣v␣:␣~p␣~n", [Xf,Yf,T1,
                    V_mes]),
176             io:format("angle␣:~p␣,␣~p~n␣", [ThetaDegrees,T1]),
177             Room = determine_robot_room(Xf, Yf, OldRoom),
178
179
180             hera_data:store(robot_pos, robot, Seq, [Xf, Yf,
                    ThetaDegrees, Room]),
181             send_robot_pos(Seq, [Xf, Yf,  ThetaDegrees, Room]),
182             {no_share, NewState}; % If no propag graph, delete
                    prev line and replace no_share by a ok clause (see
                    hera_measure module)
183         [] ->
184             {undefined, State}
185
186     end.
187
188
189
190 calibrate_speed() ->
191     I2Cbus = persistent_term:get(i2c),
192     N = 10,
193     RawSpeeds = [grisp_i2c:transfer(I2Cbus, [{read, 16#40, 1, 5}])
            || _ <- lists:seq(1, N)],
194     Decoded = [hera_com:decode_half_float([<<SL1, SL2>>, <<SR1,
            SR2>>]) ||
195                   [<<SL1, SL2, SR1, SR2, _>>] <- RawSpeeds],
196     SpeedsL = [L || [L, _] <- Decoded],
197     SpeedsR = [R || [_, R] <- Decoded],
198     OffsetL = lists:sum(SpeedsL) / N,
199     OffsetR = lists:sum(SpeedsR) / N,
200     persistent_term:put(i2c_offset, {OffsetL, OffsetR}).
201
202
203
204 send_robot_pos(Seq, [Xf,Yf, ThetaDegrees, Room]) ->
205     Msg = "Robot_pos," ++ integer_to_list(Seq) ++","++
            float_to_list(Xf) ++ "," ++ float_to_list(Yf) ++ "," ++
            float_to_list(ThetaDegrees) ++ "," ++ integer_to_list(Room)
            ,
206     Adjacent_sensors = get_adjacent_sensors(Room), % To impose the
            propagation graph, not mandatory
```

```
207     [hera_com:send_unicast(Device, Msg, "UTF8") || Device <- [
            server | Adjacent_sensors]].
208
209 get_new_robot_pos(Room) ->
210     [Sensor1, Sensor2] = get_room_sensors(Room),
211
212     io:format("[KALMAN_MEASURE] The two sensors in the current
            room are : ~p and ~p ~n",[Sensor1, Sensor2]),
213     {X1, Y1, _} = get_sensor_pos(Sensor1),
214     {X2, Y2, _} = get_sensor_pos(Sensor2),
215     [{_, Seqsensor1,_,[Dist1]}] = hera_data:get(distance, Sensor1)
            , % distance on the ground
216     [{_, Seqsensor2,_,[Dist2]}] = hera_data:get(distance, Sensor2)
            ,
217     {TLx, TLy, BRx, BRy} = get_room_info(Room),
218     {get_pos({X1, Y1}, {X2, Y2}, {Dist1} , {Dist2},{TLx, TLy, BRx,
            BRy}),{Seqsensor1,Seqsensor2}}.
219
220 get_room_sensors(Room) ->
221     % Returns the two sensors in the current room
222     Devices = persistent_term:get(devices),
223     lists:foldl(
224         fun({Name, _, _}, Acc) ->
225             case Name of
226                 _ ->
227                     case hera_data:get(room, Name) of
228                         [{_, _, _, [ORoom]}] when Room =:= ORoom
                                ->
229                             [Name | Acc];
230                         _ ->
231                             Acc
232                     end
233             end
234         end,
235         [],
236         Devices
237     ).
238
239 get_adjacent_sensors(Room) ->
240     % Returns all the sensors in the current room or in the
            adjacent rooms.
241     Devices = persistent_term:get(devices),
242     lists:foldl(
243         fun({Name, _, _}, Acc) ->
244             case Name of
245                 _ ->
246                     case hera_data:get(room, Name) of
247                         [{_, _, _, [ORoom]}] when ((Room =:= ORoom
                                ) orelse (Room+1 =:= ORoom) orelse (
```

```
                                Room-1 =:=ORoom)) ->
248                                 [Name | Acc];
249                             _ ->
250                                 Acc
251                         end
252                 end
253             end,
254             [],
255             Devices
256         ).
257
258 get_sensor_pos(SensorName) ->
259     case hera_data:get(pos, SensorName) of
260         [{_, _, _, [X, Y, _, A]}] ->
261             {X, Y, A};
262         _ ->
263             io:format("[KALMAN_MEASURE] Can't get the pos of
                sensor : ~p~n", [SensorName])
264     end.
265
266 get_room_info(OldRoom) ->
267     case hera_data:get(room_info,OldRoom) of
268         [{_, _, _, [TLx, TLy, BRx, BRy]}] ->
269             {TLx, TLy, BRx, BRy};
270         _ ->
271             io:format("[KALMAN_MEASURE] Can't get the pos of room
                n  ~p~n", [OldRoom])
272     end.
273
274 get_pos({X1,Y1}, {X2,Y2}, {Dist1}, {Dist2},{TLx, TLy, BRx, BRy})
    ->
275     Dx = (X2 - X1) * 100,
276     Dy = (Y2 - Y1) * 100,
277     D = math:sqrt(Dx*Dx + Dy * Dy),
278     case (D > Dist1 + Dist2) orelse (D < abs(Dist1 - Dist2))
        orelse (D == 0 andalso Dist1 == Dist2) of
279         true ->
280             no_intersection;
281         false ->
282             A = (Dist1 * Dist1 - Dist2 * Dist2 + D*D) / (2*D),
283             H = math:sqrt(Dist1*Dist1 - A*A),
284
285             Px = (X1*100) + A * (Dx/D),
286             Py = (Y1*100) + A * (Dy/D),
287
288             Rx = -Dy * (H/D),
289             Ry =  Dx * (H/D),
290
291             Xout1 = Px + Rx,
```

```
292             Yout1 = Py + Ry,
293             Xout2 = Px - Rx,
294             Yout2 = Py - Ry,
295             check_good_point(Xout1, Yout1, Xout2, Yout2, TLx, TLy,
                    BRx, BRy)
296     end.
297
298 check_good_point(Xout1, Yout1, Xout2, Yout2, TLx, TLy, BRx, BRy)
    ->
299
300     MaxRoomX = lists:max([TLx, BRx])*100,
301     MaxRoomy = lists:max([TLy, BRy])*100,
302     MinRoomX = lists:min([TLx, BRx])*100,
303     MinRoomy = lists:min([TLy, BRy])*100,
304     case {MinRoomX =< Xout1 andalso Xout1 =< MaxRoomX, MinRoomy =<
            Yout1 andalso Yout1 =< MaxRoomy} of
305         {true, true} ->
306             {Xout1/100, Yout1/100};
307         _ ->
308             case {MinRoomX =< Xout2 andalso Xout2 =< MaxRoomX,
                    MinRoomy =< Yout2 andalso Yout2 =< MaxRoomy} of
309                 {true, true} ->
310                     {Xout2/100, Yout2/100};
311                 _ ->
312                     no_intersection
313             end
314     end.
315
316
317 determine_robot_room(X, Y, OldRoom) ->
318     determine_robot_room(X, Y, OldRoom, 0).
319 determine_robot_room(X, Y, OldRoom, RoomNum) ->
320     case hera_data:get(room_info, RoomNum) of
321         [{_, _, _, [TLx, TLy, BRx, BRy]}] ->
322             if
323                 (X > TLx andalso X < BRx) andalso (Y > TLy andalso
                        Y < BRy) ->
324
325                     RoomNum;
326                 true ->
327                     determine_robot_room(X, Y, OldRoom, RoomNum+1)
328             end;
329         [] ->
330             io:format("[KALMAN_MEASURE] Error: Not in a known room
                    ~n"),
331             OldRoom
332     end.
333
334 scale(List, Factor) ->
```

```
335     [X*Factor || X <- List].
336
337 get_val_nav_2(R) ->
338
339     [Ax, Ay, Az] = pmod_nav:read(acc, [out_x_xl, out_y_xl,
            out_z_xl]),
340     [Gx, Gy, Gz] = pmod_nav:read(acc, [out_x_g, out_y_g, out_z_g])
            ,
341     [Mx, My, Mz] = pmod_nav:read(mag, [out_x_m, out_y_m, out_z_m])
            ,
342     {Ax0, Ay0, Az0} = persistent_term:get(acc_init),
343     %Acc = scale([Ax - Ax0, Ay - Ay0, Az - Az0], 9.81),
344     Acc = scale([ (Az - Az0), (Ay - Ay0), -(Ax - Ax0)], 9.81),
345
346
347     {GBx, GBy, GBz} = persistent_term:get(gyro_init),
348     %Gyro = scale([Gx-GBx,Gy-GBy,Gz-GBz],math:pi()/180),
349     Gyro = scale([(Gz-GBz),(Gy-GBy),-(Gx-GBx)],math:pi()/180),
350
351     {MBx,MBy,MBz} = persistent_term:get(mag_init),
352     %Mag = mat:matrix([[Mx-MBx,My-MBy,Mz-MBz]]),
353     Mag = mat:matrix([[(Mz-MBz),(My-MBy),-(Mx-MBx)]]),
354
355
356     AccRot = mat:'*'(mat:matrix([Acc]), mat:tr(R)),  % rotation
            dans le rep re monde
357     RotAcc = mat:'-'(AccRot, mat:matrix([[9.81, 0, 0]])),  %
            compensation gravit
358
359     %R0 = ahrs([Ax,Ay,Az], [(Mx-MBx),My-MBy,(Mz-MBz)]),
360     R0 = ahrs([Az,Ay,-Ax], [(Mz-MBz),(My-MBy),-(Mx-MBx)]),
361     mat:tr(R0),
362
363     {mat:matrix([Acc]), RotAcc, mat:matrix([Gyro]), Mag,R0}.
364
365 i2c_read() ->
366     %Receive I2C and conversion
367     I2Cbus = persistent_term:get(i2c),
368     [<<SL1,SL2,SR1,SR2,CtrlByte>>] = grisp_i2c:transfer(I2Cbus, [{
            read, 16#40, 1, 5}]),
369     {OffsetL,OffsetR} = persistent_term:get(i2c_offset),
370     [Speed_L,Speed_R] = hera_com:decode_half_float([<<SL1, SL2>>,
            <<SR1, SR2>>]),
371     Speed2 = ((Speed_L - OffsetL) + (Speed_R - OffsetR))/2,
372     Speed = case erlang:abs(Speed2)/100 < 0.08 of
373         true -> 0.0;
374         false -> Speed2
375     end,
376     if
```

```
377     (Speed_L < 0 andalso Speed_R > 0) orelse (Speed_L > 0 andalso
            Speed_R < 0) ->
378         io:format("Signes␣oppos s~n"),
379 
380         {0.0, CtrlByte};
381     true ->
382         io:format("Autre␣cas~n"),
383         {Speed, CtrlByte}
384 
385     end.
386 
387 quat_to_yaw([[Q0], [Q1], [Q2], [Q3]]) ->
388     math:atan2(2*(Q0*Q3 + Q1*Q2), 1 - 2*(Q2*Q2 + Q3*Q3)).
389 normalize_quat([Q0, Q1, Q2, Q3]) ->
390     Norm = math:sqrt(Q0*Q0 + Q1*Q1 + Q2*Q2 + Q3*Q3),
391     [[Q0 / Norm], [Q1 / Norm], [Q2 / Norm], [Q3 / Norm]].
392 
393 kalman_orientation(Xor,Por,T1,T0) ->
394     Dtor = (T1-T0)/1000.0,
395     Rorien = q2dcm(mat:to_array(Xor)),
396     {Acc, _Acclin, Gyro, Mag, R0} = get_val_nav_2(Rorien),
397     [Acx,Acy,Acz] = mat:to_array(Acc),
398     [Mx,My,Mz] = mat:to_array(Mag),
399     R1 = ahrs([Acx,Acy,Acz], [Mx,My,Mz]),
400     Quat = dcm2quat(mat:'*'(R1,R0)),
401     [Wxx,Wyy,Wzz] = mat:to_array(Gyro),
402     [Wx,Wy,Wz] = [Wxx,Wyy,Wzz],
403     Omega = mat:matrix([
404         [0,Wx,Wy,Wz],
405         [-Wx,0,-Wz,Wy],
406         [-Wy,Wz,0,-Wx],
407         [-Wz,-Wy,Wx,0]
408     ]),
409 
410     For = mat:'+'(mat:eye(4), mat:'*'(0.5 * Dtor, Omega)),
411     Qor = mat:diag([?VAR_Q,?VAR_Q,?VAR_Q,?VAR_Q]),
412     Hor = mat:eye(4),
413     Zor = mat:tr(Quat),
414     Ror = mat:diag([?VAR_R,?VAR_R,?VAR_R,?VAR_R]),
415 
416     {Xor00, Por0} = hera_kalman:predict({Xor,Por}, For, Qor),
417     Xor0 = normalize_vec4(Xor00),
418     Zor_used = case qdot(mat:to_array(Zor), mat:to_array(Xor0)) >
            0 of
419             true  -> Zor;
420             false -> mat:'*'(-1, Zor)
421         end,
422     {Xor11, Por1} = hera_kalman:update({Xor0, Por0}, Hor, Ror,
            Zor_used),
```

```
423     Xor1 = normalize_vec4(Xor11),
424
425     {Xor1,Por1}.
426
427 q2dcm([Q0, Q1, Q2, Q3]) ->
428     R00 = 2 * (Q0 * Q0 + Q1 * Q1) - 1,
429     R01 = 2 * (Q1 * Q2 - Q0 * Q3),
430     R02 = 2 * (Q1 * Q3 + Q0 * Q2),
431
432     R10 = 2 * (Q1 * Q2 + Q0 * Q3),
433     R11 = 2 * (Q0 * Q0 + Q2 * Q2) - 1,
434     R12 = 2 * (Q2 * Q3 - Q0 * Q1),
435
436     R20 = 2 * (Q1 * Q3 - Q0 * Q2),
437     R21 = 2 * (Q2 * Q3 + Q0 * Q1),
438     R22 = 2 * (Q0 * Q0 + Q3 * Q3) - 1,
439
440     mat:matrix([
441     [R00, R01, R02],
442     [R10, R11, R12],
443     [R20, R21, R22]
444     ]).
445
446 dcm2quat(R) ->
447     [R00,R01,R02,
448      R10,R11,R12,
449      R20,R21,R22] = mat:to_array(R),
450     Tr = R00 + R11 + R22,
451     {Q0,Q1,Q2,Q3} =
452       case Tr > 0 of
453         true ->
454           S = math:sqrt(Tr + 1.0) * 2.0,
455           {0.25*S,
456            (R21 - R12) / S,
457            (R02 - R20) / S,
458            (R10 - R01) / S};
459         false ->
460           case (R00 > R11) andalso (R00 > R22) of
461             true ->
462               S = math:sqrt(1.0 + R00 - R11 - R22) * 2.0,
463               {(R21 - R12) / S,
464                0.25*S,
465                (R01 + R10) / S,
466                (R02 + R20) / S};
467             false ->
468               case R11 > R22 of
469                 true ->
470                   S = math:sqrt(1.0 + R11 - R00 - R22) * 2.0,
471                   {(R02 - R20) / S,
```

```
                    (R01 + R10) / S,
                    0.25*S,
                    (R12 + R21) / S};
                false ->
                  S = math:sqrt(1.0 + R22 - R00 - R11) * 2.0,
                  {(R10 - R01) / S,
                   (R02 + R20) / S,
                   (R12 + R21) / S,
                   0.25*S}
            end
        end
    end,
    N = math:sqrt(Q0*Q0 + Q1*Q1 + Q2*Q2 + Q3*Q3),
    mat:matrix([[Q0/N, Q1/N, Q2/N, Q3/N]]).

ahrs(Acc, Mag) ->
    Down = unit([-A || A <- Acc]),
    East = unit(cross_product(Down, unit(Mag))),
    North = unit(cross_product(East, Down)),
    mat:tr(mat:matrix([North, East, Down])).

unit(Vec) ->
    Norm = math:sqrt(lists:sum([X*X || X <- Vec])),
    case Norm of
        0 -> Vec;
        _ -> [X / Norm || X <- Vec]
    end.

cross_product([U1,U2,U3], [V1,V2,V3]) ->
    [U2*V3-U3*V2, U3*V1-U1*V3, U1*V2-U2*V1].

qdot([Q11, Q12, Q13, Q14], [Q21, Q22, Q23, Q24]) ->
    Q11*Q21 + Q12*Q22 + Q13*Q23 + Q14*Q24.

normalize_vec4(Q) ->
  [Q0,Q1,Q2,Q3] = mat:to_array(Q),
  N = math:sqrt(Q0*Q0 + Q1*Q1 + Q2*Q2 + Q3*Q3),
  mat:matrix([[Q0/N],[Q1/N],[Q2/N],[Q3/N]]).

wrap_angle_deg(Angle) ->
    Wrapped = math:fmod(Angle + 180, 360),
    case Wrapped < 0 of
        true -> Wrapped + 360 - 180;
        false -> Wrapped - 180
    end.
```

F.3 Stability_layer module

```
-module(stability_layer).

-export([robot_init/0]).

-define(RAD_TO_DEG, 180.0/math:pi()).
-define(DEG_TO_RAD, math:pi()/180.0).

-define(ADV_V_MAX, 30.0).
-define(TURN_V_MAX, 80.0).

robot_init() ->

    process_flag(priority, max),

    calibrate(),
    calibrate2(),

    {X0, P0} = init_kalman(),

    %I2C bus
    case open_i2C_bus() of
        ok ->
            %PIDs initialisation
            Pid_Speed = spawn(hera_pid_controller, pid_init,
                [-0.12, -0.07, 0.0, -1, 60.0, 0.0]),
            Pid_Stability = spawn(hera_pid_controller, pid_init,
                [17.0, 0.0, 4.0, -1, -1, 0.0]),
            persistent_term:put(controllers, {Pid_Speed,
                Pid_Stability}),
            persistent_term:put(freq_goal, 300.0),

            T0 = erlang:system_time()/1.0e6,

            io:format("[ROBOT] Stability ready.~n"),

            State = #{
                robot_state => {rest, false}, %{Robot_State,
                    Robot_Up}
                kalman_state => {T0, X0, P0}, %{Tk, Xk, Pk}
                move_speed => {0.0, 0.0}, % {Adv_V_Ref, Turn_V_Ref
                    }
                frequency => {0, 0, 200.0, T0} %{N, Freq,
                    Mean_Freq, T_End}
            },

            robot_loop(State);
```

```
41          error ->
42              io:format("[ROBOT] Stability could not start, retrying
                    in 2 seconds.~n"),
43              timer:sleep(2000),
44              robot_init()
45      end.
46
47
48  robot_loop(State) ->
49
50
51      {Robot_State, Robot_Up} = maps:get(robot_state, State),
52      {Tk, Xk, Pk} = maps:get(kalman_state, State),
53      {Adv_V_Ref, Turn_V_Ref} = maps:get(move_speed, State),
54      {N, Freq, Mean_Freq, T_End} = maps:get(frequency, State),
55
56
57      T1 = erlang:system_time()/1.0e6,
58          Dt = (T1- Tk)/1000.0,
59
60
61      [Gy,Ax,Az] = pmod_nav:read(acc, [out_y_g, out_x_xl, out_z_xl],
            #{g_unit => dps}),
62
63
64      {Speed, CtrlByte} = i2c_read(),
65      [Arm_Ready, _, _, Get_Up, Forward, Backward, Left, Right] =
            hera_com:get_bits(CtrlByte),
66
67      Adv_V_Goal = speed_ref(Forward, Backward),
68      Turn_V_Goal = turn_ref(Left, Right),
69
70
71      [Angle, {X1, P1}] = kalman_angle(Dt, Ax, Az, Gy, Xk, Pk),
72
73
74      {Acc, Adv_V_Ref_New, Turn_V_Ref_New} = control_engine:
            controller({Dt, Angle, Speed}, {Adv_V_Goal, Adv_V_Ref}, {
            Turn_V_Goal, Turn_V_Ref}),
75
76
77      Robot_Up_New = is_robot_up(Angle, Robot_Up),
78      Next_Robot_State = get_robot_state({Robot_State, Robot_Up,
            Get_Up, Arm_Ready, Angle}),
79      Output_Byte = get_output_state(Next_Robot_State, Angle),
80
81
82      i2c_write(Acc, Turn_V_Ref_New, Output_Byte),
83
```

```
84
85      {N_New, Freq_New, Mean_Freq_New} = frequency_computation(Dt, N
            , Freq, Mean_Freq),
86      smooth_frequency(T_End, T1),
87
88
89      T_End_New = erlang:system_time()/1.0e6,
90      NewState = State#{
91          robot_state => {Next_Robot_State, Robot_Up_New},
92          kalman_state => {T1, X1, P1},
93          move_speed => {Adv_V_Ref_New, Turn_V_Ref_New},
94          frequency => {N_New, Freq_New, Mean_Freq_New, T_End_New}
95      },
96
97      robot_loop(NewState).
98
99
100 open_i2C_bus() ->
101     case grisp_i2c:open(i2c1) of
102         {error, _} ->
103             io:format("[ROBOT] Error while opening the i2C bus~n")
                    ,
104             grisp_led:flash(1, red, 500),
105             error;
106         I2Cbus ->
107             persistent_term:put(i2c, I2Cbus),
108             ok
109     end.
110
111 calibrate() ->
112     grisp_led:flash(1, green, 500),
113     io:format("[ROBOT] Calibrating... Do not move the pmod_nav!~n"
            ),
114     N = 500,
115     Y_List = [pmod_nav:read(acc, [out_y_g]) || _ <- lists:seq(1, N
            )],
116     Gy0 = lists:sum([Y || [Y] <- Y_List]) / N,
117     io:format("[ROBOT] Done calibrating~n"),
118     grisp_led:flash(1, green, 500),
119     persistent_term:put(gy0, Gy0).
120
121 calibrate2() ->
122     N=500,
123     Gyro_data = [ pmod_nav:read(acc, [out_x_g, out_y_g, out_z_g])
124                   || _ <- lists:seq(1, N) ],
125
126     G_x_List = [ X || [X,_,_] <- Gyro_data ],
127     G_y_List = [ Y || [_,Y,_] <- Gyro_data ],
128     G_z_List = [ Z || [_,_,Z] <- Gyro_data ],
```

```
129
130     AngVel_data = [pmod_nav:read(mag, [out_x_m, out_y_m, out_z_m])
            || _ <- lists:seq(1, N)],
131
132     M_x_List = [ X || [X,_,_] <- AngVel_data ],
133     M_y_List = [ Y || [_,Y,_] <- AngVel_data ],
134     M_z_List = [ Z || [_,_,Z] <- AngVel_data ],
135
136     Acc_data = [ pmod_nav:read(acc, [out_x_xl, out_y_xl, out_z_xl
            ]) || _ <- lists:seq(1, N) ],
137
138     Accc_x_List = [ X || [X,_,_] <- Acc_data ],
139     Accc_y_List = [ Y || [_,Y,_] <- Acc_data ],
140     Accc_z_List = [ Z || [_,_,Z] <- Acc_data ],
141
142     [Gx0_pos, Gy0_pos, Gz0_pos] = [lists:sum(List) / N || List <-
            [G_x_List, G_y_List, G_z_List]],
143     [Mx0, My0, Mz0] = [lists:sum(List) / N || List <- [M_x_List,
            M_y_List, M_z_List]],
144     [Accx0, Accy0, Accz0] = [lists:sum(List) / N || List <- [
            Accc_x_List, Accc_y_List, Accc_z_List]],
145     io:format("[KALMAN_MEASURE] Done calibrating~n"),
146
147     persistent_term:put(gyro_init, {Gx0_pos, Gy0_pos, Gz0_pos}),
148     persistent_term:put(mag_init, {Mx0, My0, Mz0}),
149     persistent_term:put(acc_init, {Accx0, Accy0, Accz0}).
150
151 init_kalman() ->
152     % Initiating kalman constants
153     R = mat:matrix([[3.0, 0.0], [0, 3.0e-6]]),
154     Q = mat:matrix([[3.0e-5, 0.0], [0.0, 10.0]]),
155     Jh = fun (_) -> mat:matrix([[1, 0],[0, 1]])
156                 end,
157     persistent_term:put(kalman_constant, {R, Q, Jh}),
158
159     % Initial State and Covariance matrices
160     X0 = mat:matrix([[0], [0]]),
161     P0 = mat:matrix([[0.1, 0], [0, 0.1]]),
162     {X0, P0}.
163
164
165
166 kalman_angle(Dt, Ax, Az, Gy, X0, P0) ->
167     Gy0 = persistent_term:get(gy0),
168     {R, Q, Jh} = persistent_term:get(kalman_constant),
169
170     F = fun (X) -> [Th, W] = mat:to_array(X),
171                                 mat:matrix([[Th+Dt*W],[W]])
172                 end,
```

```
173     Jf = fun (_) -> mat:matrix([[1, Dt],[0, 1]])
174                     end,
175     H = fun (X) -> [Th, W] = mat:to_array(X),
176                                 mat:matrix([[Th],[W]])
177                 end,
178
179     Z = mat:matrix([[math:atan(Az / (-Ax))], [(Gy-Gy0)*?DEG_TO_RAD
            ]]),
180     {X1, P1} = hera_kalman:extended_filter({X0, P0}, {F, Jf}, {H,
            Jh}, Q, R, Z),
181
182     [Th_Kalman, _W_Kalman] = mat:to_array(X1),
183     Angle = Th_Kalman * ?RAD_TO_DEG,
184     [Angle, {X1, P1}].
185
186
187
188 get_robot_state(Robot_State) -> % {Robot_state, Robot_Up, Get_Up,
        Arm_ready, Angle}
189     case Robot_State of
190         {rest, _, true, _, _} -> raising;
191         {rest, _, _, _, _} -> rest;
192         {raising, true, _, _, _} -> stand_up;
193         {raising, _, false, _, _} -> soft_fall;
194         {raising, _, _, _, _} -> raising;
195         {stand_up, _, false, _, _} -> wait_for_extend;
196         {stand_up, false, _, _, _} -> rest;
197         {stand_up, _, _, _, _} -> stand_up;
198         {wait_for_extend, _, _, _, _} -> prepare_arms;
199         {prepare_arms, _, _, true, _} -> free_fall;
200         {prepare_arms, _, true, _, _} -> stand_up;
201         {prepare_arms, false, _, _, _} -> rest;
202         {prepare_arms, _, _, _, _} -> prepare_arms;
203         {free_fall, _, _, _, Angle} ->
204             case abs(Angle) >10 of
205                 true -> wait_for_retract;
206                 _ ->free_fall
207             end;
208         {wait_for_retract, _, _, _, _} -> soft_fall;
209         {soft_fall, _, _, true, _} -> rest;
210         {soft_fall, _, true, _, _} -> raising;
211         {soft_fall, _, _, _, _} -> soft_fall
212     end.
213
214 get_output_state(State, Angle) ->
215     Move_direction = get_movement_direction(Angle),
216     % Output bits = [Power, Freeze, Extend, Robot_Up_Bit,
            Move_direction, 0, 0, 0]
217     case State of
```

```
218         rest ->
219             get_byte([0, 0, 0, 0, Move_direction, 0, 0, 0]);
220         raising ->
221             get_byte([1, 0, 1, 0, Move_direction, 0, 0, 0]);
222         stand_up ->
223             get_byte([1, 0, 0, 1, Move_direction, 0, 0, 0]);
224         wait_for_extend ->
225             get_byte([1, 0, 1, 1, Move_direction, 0, 0, 0]);
226         prepare_arms ->
227             get_byte([1, 0, 1, 1, Move_direction, 0, 0, 0]);
228         free_fall ->
229             get_byte([1, 1, 1, 1, Move_direction, 0, 0, 0]);
230         wait_for_retract ->
231             get_byte([1, 0, 0, 0, Move_direction, 0, 0, 0]);
232         soft_fall ->
233             get_byte([1, 0, 0, 0, Move_direction, 0, 0, 0])
234     end.
235
236 is_robot_up(Angle, Robot_Up) ->
237     if
238         Robot_Up and (abs(Angle) > 20) ->
239             false;
240         not Robot_Up and (abs(Angle) < 18) ->
241             true;
242         true ->
243             Robot_Up
244     end.
245
246 get_movement_direction(Angle) ->
247     if
248         Angle > 0.0 ->
249             1;
250         true ->
251             0
252     end.
253
254 get_byte(List) ->
255     [A, B, C, D, E, F, G, H] = List,
256     A*128 + B*64 + C*32 + D*16 + E*8 + F*4 + G*2 + H.
257
258
259
260 i2c_read() ->
261     %Receive I2C and conversion
262     I2Cbus = persistent_term:get(i2c),
263     [<<SL1,SL2,SR1,SR2,CtrlByte>>] = grisp_i2c:transfer(I2Cbus, [{
        read, 16#40, 1, 5}]),
264     [Speed_L,Speed_R] = hera_com:decode_half_float([<<SL1, SL2>>,
        <<SR1, SR2>>]),
```

```
265     Speed = (Speed_L + Speed_R)/2,
266     {Speed, CtrlByte}.
267
268 i2c_write(Acc, Turn_V_Ref_New, Output_Byte) ->
269     I2Cbus = persistent_term:get(i2c),
270     [HF1, HF2] = hera_com:encode_half_float([Acc, Turn_V_Ref_New])
            ,
271     grisp_i2c:transfer(I2Cbus, [{write, 16#40, 1, [HF1, HF2, <<
            Output_Byte>>]}]).
272
273
274
275 speed_ref(Forward, Backward) ->
276     if
277         Forward ->
278             Adv_V_Goal = ?ADV_V_MAX;
279         Backward ->
280             Adv_V_Goal = - ?ADV_V_MAX;
281         true ->
282             Adv_V_Goal = 0.0
283     end,
284     Adv_V_Goal.
285
286 turn_ref(Left, Right) ->
287     if
288         Right ->
289             Turn_V_Goal = ?TURN_V_MAX;
290         Left ->
291             Turn_V_Goal = - ?TURN_V_MAX;
292         true ->
293             Turn_V_Goal = 0.0
294     end,
295     Turn_V_Goal.
296
297 frequency_computation(Dt, N, Freq, Mean_Freq) ->
298     if
299         N == 100 ->
300             N_New = 0,
301             Freq_New = 0,
302             Mean_Freq_New = Freq;
303         true ->
304             N_New = N+1,
305             Freq_New = ((Freq*N)+(1/Dt))/(N+1),
306             Mean_Freq_New = Mean_Freq
307     end,
308     {N_New, Freq_New, Mean_Freq_New}.
309
310 smooth_frequency(T_End, T1)->
311     T2 = erlang:system_time()/1.0e6,
```

```
312     Freq_Goal = persistent_term:get(freq_goal),
313     Delay_Goal = 1.0/Freq_Goal * 1000.0,
314     if
315         T2-T_End < Delay_Goal ->
316             timer:sleep(Delay_Goal-(T2-T1));
317         true ->
318             ok
319     end.
```

F.4 Control_engine module

```
1  -module(control_engine).
2
3  -export([controller/3]).
4
5  -define(ADV_V_MAX, 30.0).
6  -define(ADV_ACCEL, 75.0).
7
8  -define(TURN_V_MAX, 80.0).
9  -define(TURN_ACCEL, 400.0).
10
11 controller({Dt, Angle, Speed}, {Adv_V_Goal, Adv_V_Ref}, {
       Turn_V_Goal, Turn_V_Ref}) ->
12     {Pid_Speed, Pid_Stability} = persistent_term:get(controllers),
13
14     %Saturate advance acceleration
15     if
16         Adv_V_Goal > 0.0 ->
17             Adv_V_Ref_New = hera_pid_controller:saturation(
                   Adv_V_Ref+?ADV_ACCEL*Dt, ?ADV_V_MAX);
18         Adv_V_Goal < 0.0 ->
19             Adv_V_Ref_New = hera_pid_controller:saturation(
                   Adv_V_Ref- ?ADV_ACCEL*Dt, ?ADV_V_MAX);
20         true ->
21             if
22                 Adv_V_Ref > 0.5  ->
23                     Adv_V_Ref_New = hera_pid_controller:saturation
                           (Adv_V_Ref- ?ADV_ACCEL*Dt, ?ADV_V_MAX);
24                 Adv_V_Ref < -0.5 ->
25                     Adv_V_Ref_New = hera_pid_controller:saturation
                           (Adv_V_Ref+?ADV_ACCEL*Dt, ?ADV_V_MAX);
26                 true ->
27                     Adv_V_Ref_New = 0.0
28             end
29     end,
30
```

```
31      %Saturate turning acceleration
32      if
33          Turn_V_Goal > 0.0 ->
34              Turn_V_Ref_New = hera_pid_controller:saturation(
                    Turn_V_Ref+?TURN_ACCEL*Dt, ?TURN_V_MAX);
35          Turn_V_Goal < 0.0 ->
36              Turn_V_Ref_New = hera_pid_controller:saturation(
                    Turn_V_Ref- ?TURN_ACCEL*Dt, ?TURN_V_MAX);
37          true ->
38              if
39                  Turn_V_Ref > 0.5  ->
40                      Turn_V_Ref_New = hera_pid_controller:
                            saturation(Turn_V_Ref- ?TURN_ACCEL*Dt, ?
                            TURN_V_MAX);
41                  Turn_V_Ref < -0.5 ->
42                      Turn_V_Ref_New = hera_pid_controller:
                            saturation(Turn_V_Ref+?TURN_ACCEL*Dt, ?
                            TURN_V_MAX);
43                  true ->
44                      Turn_V_Ref_New = 0.0
45              end
46      end,
47
48      %Speed PI
49      Pid_Speed ! {self(), {set_point, Adv_V_Ref_New}},
50      Pid_Speed ! {self(), {input, Speed}},
51      receive {_, {control, Target_angle}} -> ok end,
52
53      %Stability PD
54      Pid_Stability ! {self(), {set_point, Target_angle}},
55      Pid_Stability ! {self(), {input, Angle}},
56      receive {_, {control, Acc}} -> ok end,
57
58      {Acc, Adv_V_Ref_New, Turn_V_Ref_New}.
```

Appendix G

User Controller Software

G.1 Run bash script

```
#!/bin/bash
#Created with the help of ChatGPT

MAX_TRIES=10
COUNT=0

DEVICE_IP=$(ip route get 1.1.1.1 | awk '{for(i=1;i<=NF;i++) if($i=="src") print $(i+1)}')
BROADCAST_IP=$(ip -o -f inet addr show | awk '/scope global/ {print $6}' | head -n1)

while [ $COUNT -lt $MAX_TRIES ]; do
    if [ $# -eq 1 ]; then
        FILENAME="$1"
        python3 Controller.py "$DEVICE_IP" "$BROADCAST_IP" "$FILENAME"
    fi

    if [ $# -lt 1 ]; then
        python3 Controller.py "$DEVICE_IP" "$BROADCAST_IP"
    fi

    EXIT_CODE=$?
    if [ $EXIT_CODE -eq 0 ]; then
        echo "Controller.py exited successfully."
        exit 0
    else
        COUNT=$((COUNT + 1))
        echo "Controller.py failed (attempt $COUNT/$MAX_TRIES). Retrying in 0.5s..."
```

```
        sleep 0.5
    fi
done

echo "Controller.py failed $MAX_TRIES times. Exiting."
exit 1
```

G.2 Pygame Controller

```
import pygame
import pygame_gui
import sys
import numpy as np
import serial
from Server import Server
from components.Room import Room
from components.Robot import Robot
from pathlib import Path
import time

class User_interface:

    # App General State
    RESIZE = 3.5 # Resizing factor for the rooms
    running = True # True until quit command received
    image_dict = {} # Contains all the images object
    rect_dict = {} # Contains all the images rect

    # Pop Up
    in_popup = False # True if there's an active popup
    active_popup = None # Actual pop-up object
    UI_elements = {} # Elements of the current popup
    temp_origin = None # Used to keep track of the origin button
        in a popup

    # Robot controll
    x = 0 # Robot command
    string = "" # String to be printed on screen


    # App Room state
    room_grid = ((0,0),(0,0)) # Building grid (from top left to
        bottom right)
    rooms = [] # List of defined rooms
    sensor = [] # List of defined sensors

```

```
36      # Predefined trajectory
37      trajectory = [] # List of predefined commands
38      trajectory_idx = 0 # Command index
39      current_action = "" # Name of the current command
40      action_start_time = 0 # Time at the start of the trajectory
41      action_duration = 0 # Total time duration of the command
42      is_trajectory_started = False # Set to true when the
            trajectory begins
43      timer = 0 # Keeps track of the time since started
44
45      # Robot UI
46      robot = None # Robot object
47
48      # Robot state
49      message = 0  #Message to send to the robot
50      run = True
51      stand = False
52      kalman = True
53      release_space = True
54      release_enter = True
55      release_t = True
56      release_tab = True
57
58      # Saved Files
59      saved_files = []
60
61      def __init__(self, IP, BRD_IP, trajectory):
62
63          pygame.init()
64          self.ser = serial.Serial(port="/dev/ttyACM0", baudrate
                =115200)
65          info = pygame.display.Info()
66          self.WIDTH, self.HEIGHT = info.current_w, info.current_h
67
68
69          self.screen = pygame.display.set_mode((self.WIDTH, self.
                HEIGHT), pygame.RESIZABLE)
70          pygame.display.set_caption("Robot␣Controller")
71
72          self.manager = pygame_gui.UIManager((self.WIDTH, self.
                HEIGHT))
73          self.clock = pygame.time.Clock()
74          self.clock.tick(200)
75
76          self.server = Server(IP, BRD_IP)
77          self.defined_trajectory(trajectory)
78
79          self.load_figures()
80
```

```
81      def load_figures(self): # Loads all the images that will
            appear on screen
82          arrow_img = pygame.image.load('./img/arrow.png')
83          arrow_img = pygame.transform.scale(arrow_img, (arrow_img.
                get_width() // 4, arrow_img.get_height() // 4))
84
85          circle_img = pygame.image.load('./img/point.png')
86          circle_img = pygame.transform.scale(circle_img, (
                circle_img.get_width() // 2, circle_img.get_height() //
                 2))
87
88          stop_img = pygame.image.load('./img/Stop_sign.png')
89          stop_img = pygame.transform.scale(stop_img, (stop_img.
                get_width() // 10, stop_img.get_height() // 10))
90
91          robot = pygame.image.load('./img/Robot.png')
92          robot = pygame.transform.scale(robot, (robot.get_width()
                //6, robot.get_height()//6))
93
94          plus_img = pygame.image.load('./img/plus.png')
95          plus_img = pygame.transform.scale(plus_img, (plus_img.
                get_width() // 5, plus_img.get_height() // 5))
96
97          minus_img = pygame.image.load('./img/minus.png')
98          minus_img = pygame.transform.scale(minus_img, (minus_img.
                get_width() // 5, minus_img.get_height() // 5))
99
100         start_img = pygame.image.load('./img/button_start.png')
101         start_img = pygame.transform.scale(start_img, (start_img.
                get_width(), start_img.get_height()))
102
103         start_img_pressed = pygame.image.load('./img/start_pressed
                .png')
104         start_img_pressed = pygame.transform.scale(
                start_img_pressed, (start_img_pressed.get_width(),
                start_img_pressed.get_height()))
105
106         save_img = pygame.image.load('./img/button_save.png')
107         save_img = pygame.transform.scale(save_img, (save_img.
                get_width(), save_img.get_height()))
108
109         load_img = pygame.image.load('./img/button_load.png')
110         load_img = pygame.transform.scale(load_img, (load_img.
                get_width(), load_img.get_height()))
111
112         place_robot = pygame.image.load('./img/button_place_robot.
                png')
113         place_robot = pygame.transform.scale(place_robot, (
                place_robot.get_width(), place_robot.get_height()))
```

```
114
115         zoom_in = pygame.image.load('./img/zoom_in.png')
116         zoom_in = pygame.transform.scale(zoom_in, (zoom_in.
                get_width()//8, zoom_in.get_height()//8))
117
118         zoom_out = pygame.image.load('./img/zoom_out.png')
119         zoom_out = pygame.transform.scale(zoom_out, (zoom_out.
                get_width()//8, zoom_out.get_height()//8))
120
121         self.image_dict["arrow"] = arrow_img
122         self.image_dict["circle"] = circle_img
123         self.image_dict["stop"] = stop_img
124         self.image_dict["plus_L_0"] = plus_img
125         self.image_dict["minus"] = minus_img
126         self.image_dict["start"] = start_img
127         self.image_dict["save"] = save_img
128         self.image_dict["load"] = load_img
129         self.image_dict["place_robot"] = place_robot
130         self.image_dict["start_pressed"] = start_img_pressed
131         self.image_dict["zoom_in"] = zoom_in
132         self.image_dict["zoom_out"] = zoom_out
133         self.image_dict["robot"] = robot
134
135     def event_handler(self):
136         for event in pygame.event.get():
137             if event.type == pygame.QUIT:
138                 self.server.send("Exit", "brd")
139                 time.sleep(0.25)
140                 self.running = False
141
142             if event.type == pygame.MOUSEBUTTONDOWN:
143                 self.event_click(event)
144
145             if event.type == pygame_gui.UI_BUTTON_PRESSED:
146                 self.event_interact_popup(event)
147
148             self.manager.process_events(event)
149
150     def event_click(self, event):
151         if self.in_popup:
152             return
153
154         for room in range(len(self.rooms)):
155             for side in ["L", "R", "T", "B"]:
156                 name = "plus_" + side + "_" + str(room)
157                 if self.is_click_image(name, event) :
158                     self.in_popup = True
159                     self.temp_origin = name
160                     self.create_choice_popup()
```

```
161
162             for corner in ["TL", "TR", "BL", "BR"]:
163                 name = "plus_" + corner + "_" + str(room)
164                 if self.is_click_image(name, event) :
165                     self.in_popup = True
166                     self.temp_origin = name
167                     self.create_sensor_popup()
168
169             room_obj = self.rooms[room]
170
171             room_rect = pygame.Rect(0, 0, room_obj.width, room_obj
                    .height)
172             room_rect.center = room_obj.pos
173
174         if len(self.rooms) == 0 and self.is_click_image("plus_L_0"
                , event):
175             self.in_popup = True
176             self.temp_origin = "plus_L_0"
177             self.create_room_popup()
178
179         elif self.is_click_image("start", event) :
180             self.server.send_config()
181             time.sleep(2)
182             self.is_trajectory_started = True
183             self.timer = pygame.time.get_ticks()/1000
184
185         elif self.is_click_image("place_robot", event):
186             self.in_popup = True
187             self.create_robot_popup()
188
189     def event_interact_popup(self, event):
190         if event.ui_element == self.UI_elements.get("Room_Submit")
                :
191             width = self.UI_elements.get("Width").get_text()
192             height = self.UI_elements.get("Height").get_text()
193             if width != "" and height != "":
194                 try :
195                     screen_width, screen_height = self.
                            compute_screen_size(float(width), float(
                            height))
196                     side = self.temp_origin[5]
197                     x, y = self.compute_pos_room(screen_width,
                            screen_height, side, len(self.rooms))
198
199                     room = Room(screen_width, screen_height, x, y,
                             len(self.rooms))
200                     self.add_sides(room)
201                     self.rooms.append(room)
202                     self.get_new_grid()
```

```
203                     TLx, TLy = room.compute_pos("TR")
204                     BRx, BRy = room.compute_pos("BL")
205                     TLpos = self.get_real_pos(TLx, TLy)
206                     BRpos = self.get_real_pos(BRx, BRy)
207                     self.server.add_edges(BRpos, TLpos)
208                 except :
209                     print("[ERROR] : Problem with width and height
                        values")
210             self.close_popup()
211             self.temp_origin = None
212         elif event.ui_element == self.UI_elements.get("Save_Submit
            "):
213             filename = self.UI_elements.get("Filename").get_text()
214             self.create_save_file(filename)
215             self.close_popup()
216         elif event.ui_element == self.UI_elements.get("Sensor"):
217             self.close_popup()
218             self.create_sensor_popup()
219         elif event.ui_element == self.UI_elements.get("
            SensH_Submit"):
220             height = self.UI_elements.get("Sens_height").get_text
                ()
221 
222             height = float(height)
223             self.server.update_sens_height(self.temp_origin,
                height)
224 
225             self.close_popup()
226             self.temp_origin = None
227         elif event.ui_element == self.UI_elements.get("Room"):
228             self.close_popup()
229             self.create_room_popup()
230         elif event.ui_element == self.UI_elements.get("
            Submit_robot_pos"):
231             Robot_x = self.UI_elements.get("Robot_x").get_text()
232             Robot_y = self.UI_elements.get("Robot_y").get_text()
233             Robot_angle = self.UI_elements.get("Robot_angle").
                get_text()
234 
235             try :
236                 self.server.update_robot((float(Robot_x),float(
                    Robot_y)), int(Robot_angle))
237                 self.robot = self.server.robot
238                 self.robot.confirmed = True
239             except:
240                 print("[CONTROLLER] Error with the robot placement
                    ")
241             self.close_popup()
242 
```

```
243             #Check sensor choice
244             sensors = self.server.get_sensors()
245             for id in sensors:
246                 if event.ui_element == self.UI_elements.get("Sensor␣
                        choice␣" + str(id)):
247                     self.close_popup()
248                     splitted_origin = self.temp_origin.split("_")
249                     room = int(splitted_origin[2])
250                     side = splitted_origin[1]
251 
252                     room = self.rooms[room]
253 
254                     ix, iy = room.compute_pos(side)
255 
256                     x, y = self.get_real_pos(ix, iy)
257                     self.create_sensor_height_popup()
258                     self.server.update_sens(id, side, room.room_num, x
                            , y)
259                     self.draw_sensor()
260                     self.temp_origin = id
261 
262             #File loader choice
263             for filename in self.saved_files:
264                 if event.ui_element == self.UI_elements.get("SavedFile
                        " + filename[:-4]) :
265                     self.rooms = SaveParser.parse(filename, self.
                            HEIGHT, self.RESIZE)
266                     self.close_popup()
267 
268             self.in_popup = False
269 
270         def check_keys_movement(self, keys):
271             if keys[pygame.K_SPACE]:
272                 if self.release_space:
273                     self.release_space = False
274                     if self.message < 10000000:
275                         self.run = True
276                     else:
277                         self.run = False
278                         self.is_trajectory_started = False
279                         self.current_action = ""
280                         self.action_duration = 0
281                         self.trajectory_idx = 0
282             elif keys[pygame.K_z] or keys[pygame.K_UP] or self.
                    current_action == "front":
283                 self.x += -1
284             elif keys[pygame.K_s] or keys[pygame.K_DOWN] or self.
                    current_action == "back":
285                 self.x += 1
```

```
286         elif keys[pygame.K_q] or keys[pygame.K_LEFT] or self.
                current_action == "left":
287             self.x += 1j
288         elif keys[pygame.K_d] or keys[pygame.K_RIGHT] or self.
                current_action == "right":
289             self.x += -1j
290         elif keys[pygame.K_ESCAPE]:
291             self.running = False
292 
293         else:
294             self.release_space = True
295 
296     def check_keys_kalman(self, keys):
297         if keys[pygame.K_k]:
298             self.kalman = True
299         elif keys[pygame.K_c]:
300             self.kalman = False
301         elif keys[pygame.K_TAB]:
302             if self.release_tab:
303                 self.kalman = not self.kalman
304                 self.release_tab = False
305         else:
306             self.release_tab = True
307 
308     def check_test(self, keys):
309         if keys[pygame.K_t] and self.release_t:
310                 self.test, self.release_t = True, False
311         else:
312             self.test, self.release_t =  False, True
313 
314     def check_standing(self, keys):
315         if keys[pygame.K_RETURN] or self.current_action == "stand"
                :
316             if self.release_enter:
317                 self.stand = not self.stand
318                 self.release_enter = False
319         else:
320             self.release_enter = True
321 
322     def update_screen_size(self):
323         self.WIDTH, self.HEIGHT = self.screen.get_size()
324         self.manager.set_window_resolution((self.WIDTH, self.
                HEIGHT))
325 
326     def draw_move_ctrl(self):
327         if self.message < 10000000:
328             self.draw_image("stop", self.WIDTH //2, 100)
329         elif abs(self.x) == 0:
330             self.draw_image("circle", self.WIDTH //2, 100)
```

```
331         else:
332             angle = np.angle(-1*self.x, deg=True)
333             rotated_arrow = pygame.transform.rotate(self.
                    image_dict.get("arrow"), angle)
334             rotated_rect = rotated_arrow.get_rect(center = (self.
                    WIDTH//2, 100))
335             self.screen.blit(rotated_arrow, rotated_rect.topleft)
336 
337     def draw_string(self):
338         font = pygame.font.Font(None, 36)
339         self.string += "DOWN␣\n" if not self.stand else "UP␣\n"
340         self.string += "Kalman␣filter\n" if self.kalman else "
                Complementary␣filter\n"
341         self.string += "Running\n" if self.run else "Stopped\n"
342         self.string += "Message:␣" + str(self.message) + "\n"
343         if self.is_trajectory_started :
344             self.string += "Timer␣:␣" + str(round((pygame.time.
                    get_ticks()/1000) - self.timer, 1))
345         else :
346             self.string += "Timer␣:␣0"
347 
348         for i, line in enumerate(self.string.split("\n")):
349             text = font.render(line, True, (0, 128, 0))
350             self.screen.blit(text, (10, 10 + i * 30))
351 
352     def draw_add_room(self):
353         if len(self.rooms) == 0:
354             self.draw_image("plus_L_0", self.WIDTH//2, self.HEIGHT
                    //2)
355 
356     def draw_image(self, name, x, y, angle=0):
357         image = self.image_dict.get(name)
358         image = pygame.transform.rotate(image, 360-angle)
359         image_rect = image.get_rect(center = (x, y))
360         self.screen.blit(image, image_rect)
361         self.rect_dict[name] = image_rect
362 
363     def draw_room(self, room):
364         room_rect = pygame.Rect(0, 0, room.width, room.height)
365         room_rect.center = (room.pos[0], room.pos[1])
366 
367         pygame.draw.rect(self.screen, (0, 0, 0), room_rect, width
                =10)
368         self.draw_room_sides(room)
369 
370     def draw_room_sides(self, room):
371         for side in ["L", "R", "T", "B"]:
372             if type(room.sides[side]) != Room:
```

```
373                 self.load_image(room.room_num, room.sides[side],
                        side)
374                 self.draw_image(room.sides[side].type + "_" + side
                        + "_" + str(room.room_num), room.sides[side].
                        pos[0], room.sides[side].pos[1])
375             else:
376                 x, y = room.compute_pos(side)
377                 self.draw_door(x, y)
378         for corner in ["TL", "TR", "BL", "BR"]:
379             self.load_image(room.room_num, room.corners[corner],
                    corner)
380             self.draw_image(room.corners[corner].type + "_" +
                    corner + "_" + str(room.room_num), room.corners[
                    corner].pos[0], room.corners[corner].pos[1])
381
382     def draw_sensor(self):
383         if self.temp_origin[7] =="_":
384             room_origin = int(self.temp_origin[8])
385             side_origin = self.temp_origin[5:7]
386             self.sensor.append(self.rooms[room_origin].corners[
                    side_origin])
387         else :
388             room_origin = int(self.temp_origin[7])
389             side_origin = self.temp_origin[5]
390             self.sensor.append(self.rooms[room_origin].sides[
                    side_origin])
391         self.rooms[room_origin].modify_side(side_origin, "./img/
                sensor.png", "sensor")
392
393     def draw_door(self, x, y):
394         width, height = self.compute_screen_size(0.3, 0.3)
395         door_rect = pygame.Rect(0, 0, width, height)
396         door_rect.center = (x, y)
397         pygame.draw.rect(self.screen, (255, 255, 255), door_rect)
398
399     def draw_grid(self):
400         RED = (255, 0, 0)
401
402         point1 = (self.room_grid[0][0], self.room_grid[1][0])
403         point2 = (self.room_grid[0][1], self.room_grid[1][0])
404         point3 = (self.room_grid[0][1], self.room_grid[1][1])
405
406         pygame.draw.line(self.screen, RED, point1, point2, width
                =10)
407         pygame.draw.line(self.screen, RED, point2, point3, width
                =10)
408
409     def draw_buttons(self):
410         if self.is_trajectory_started :
```

```
411             self.draw_image("start_pressed", self.WIDTH-200, 100)
412         else :
413             self.draw_image("start", self.WIDTH-200, 100)
414         if len(self.rooms) > 0:
415             self.draw_image("place_robot", self.WIDTH-450, 100)
416 
417     def draw_robot(self):
418         if self.robot != None and self.robot.confirmed:
419             x, y = self.get_screen_pos(self.robot.real_pos[0],
                        self.robot.real_pos[1])
420             self.draw_image("robot", x, y, self.robot.angle)
421 
422     def create_choice_popup(self):
423         button_width = self.WIDTH // 2 - self.WIDTH // 20
424         button_height = min(self.HEIGHT // 20, 60)
425         popup_width = self.WIDTH // 2
426         popup_height = self.HEIGHT // 3
427         margin_left = (self.WIDTH - button_width)//20
428         margin = 20
429 
430         # Center the popup on the screen
431         popup_rect = pygame.Rect(
432             (self.WIDTH - popup_width) // 2,
433             (self.HEIGHT - popup_height) // 2,
434             popup_width,
435             popup_height
436         )
437 
438         popup_window = pygame_gui.elements.UIWindow(
439             rect=popup_rect,
440             manager=self.manager,
441             window_display_title='Add␣Component'
442         )
443 
444         self.active_popup = popup_window
445 
446         current_y = margin
447 
448         header_label = pygame_gui.elements.UILabel(
449 
450             relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
451             text="Do␣you␣Want␣to␣add␣a␣Room␣or␣a␣sensor␣?",
452             manager=self.manager,
453             container=popup_window
454         )
455         current_y += button_height + margin
456 
457         self.UI_elements["Sensor"] = pygame_gui.elements.UIButton(
```

```
458             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
459             text="Sensor",
460             manager=self.manager,
461             container=popup_window
462         )
463 
464         current_y += button_height + margin
465 
466         self.UI_elements["Room"] = pygame_gui.elements.UIButton(
467             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
468             text="Room",
469             manager=self.manager,
470             container=popup_window
471         )
472 
473     def create_room_popup(self):
474         # Calculate sizes for buttons and popup dimensions
475         button_width = self.WIDTH // 2 - self.WIDTH // 20
476         button_height = min(self.HEIGHT // 20, 60)
477         popup_width = self.WIDTH // 2
478         popup_height = self.HEIGHT // 2
479         margin_left = (self.WIDTH - button_width)//20
480         margin = 20
481 
482         # Center the popup on the screen
483         popup_rect = pygame.Rect(
484             (self.WIDTH - popup_width) // 2,
485             (self.HEIGHT - popup_height) // 2,
486             popup_width,
487             popup_height
488         )
489 
490         popup_window = pygame_gui.elements.UIWindow(
491             rect=popup_rect,
492             manager=self.manager,
493             window_display_title='Room␣Creation'
494         )
495 
496         self.active_popup = popup_window
497 
498         current_y = margin
499 
500         header_label = pygame_gui.elements.UILabel(
501 
502             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
503             text="Enter␣the␣room␣size:",
```

```
            manager=self.manager,
            container=popup_window
        )
        current_y += button_height + margin

        width_label = pygame_gui.elements.UILabel(
            relative_rect=pygame.Rect(margin_left, current_y,
                button_width, button_height),
            text="Width (m):",
            manager=self.manager,
            container=popup_window
        )
        current_y += button_height + margin

        self.UI_elements["Width"] = pygame_gui.elements.
            UITextEntryLine(
            relative_rect=pygame.Rect(margin_left, current_y,
                button_width, button_height),
            manager=self.manager,
            container=popup_window
        )

        current_y += button_height + margin

        height_label = pygame_gui.elements.UILabel(
            relative_rect=pygame.Rect(margin_left, current_y,
                button_width, button_height),
            text="Height (m):",
            manager=self.manager,
            container=popup_window
        )
        current_y += button_height + margin

        self.UI_elements["Height"] = pygame_gui.elements.
            UITextEntryLine(
            relative_rect=pygame.Rect(margin_left, current_y,
                button_width, button_height),
            manager=self.manager,
            container=popup_window
        )

        current_y += int(1.75 * button_height) + margin

        self.UI_elements["Room_Submit"] = pygame_gui.elements.
            UIButton(
            relative_rect=pygame.Rect(margin_left, current_y,
                button_width, button_height),
            text="Submit",
            manager=self.manager,
```

```
545             container=popup_window
546         )
547 
548         self.manager.draw_ui(self.screen)
549         pygame.display.update()
550 
551     def create_sensor_popup(self):
552         # Calculate sizes for buttons and popup dimensions
553         button_width = self.WIDTH // 2 - self.WIDTH // 20
554         button_height = min(self.HEIGHT // 20, 60)
555         popup_width = self.WIDTH // 2
556         popup_height = self.HEIGHT // 2
557         margin_left = (self.WIDTH - button_width)//20
558         margin = 20
559 
560         # Retrieve sensors Ids
561         sensors = self.server.get_sensors()
562 
563         # Center the popup on the screen
564         popup_rect = pygame.Rect(
565             (self.WIDTH - popup_width) // 2,
566             (self.HEIGHT - popup_height) // 2,
567             popup_width,
568             popup_height
569         )
570 
571         popup_window = pygame_gui.elements.UIWindow(
572             rect=popup_rect,
573             manager=self.manager,
574             window_display_title='Sensor␣Choice'
575         )
576 
577         self.active_popup = popup_window
578 
579         current_y = margin
580 
581         header_label = pygame_gui.elements.UILabel(
582 
583             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
584             text="Choose␣a␣sensor:",
585             manager=self.manager,
586             container=popup_window
587         )
588         current_y += button_height + margin
589 
590         for Id in sensors :
591             self.UI_elements["Sensor␣choice␣" + str(Id)] =
                    pygame_gui.elements.UIButton(
```

```
592                 relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
593                 text=str(Id),
594                 manager=self.manager,
595                 container=popup_window
596             )
597 
598             current_y += button_height + margin
599 
600         self.manager.draw_ui(self.screen)
601         pygame.display.update()
602 
603     def create_sensor_height_popup(self):
604         # Calculate sizes for buttons and popup dimensions
605         button_width = self.WIDTH // 2 - self.WIDTH // 20
606         button_height = min(self.HEIGHT // 20, 60)
607         popup_width = self.WIDTH // 2
608         popup_height = self.HEIGHT // 5
609         margin_left = (self.WIDTH - button_width)//20
610         margin = 20
611 
612         # Center the popup on the screen
613         popup_rect = pygame.Rect(
614             (self.WIDTH - popup_width) // 2,
615             (self.HEIGHT - popup_height) // 2,
616             popup_width,
617             popup_height
618         )
619 
620         popup_window = pygame_gui.elements.UIWindow(
621             rect=popup_rect,
622             manager=self.manager,
623             window_display_title='Sensor␣Height'
624         )
625 
626         self.active_popup = popup_window
627 
628         current_y = margin
629 
630         header_label = pygame_gui.elements.UILabel(
631 
632             relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
633             text="At␣what␣height␣is␣the␣sensor␣(count␣to␣the␣
                        center␣of␣the␣sonar):",
634             manager=self.manager,
635             container=popup_window
636         )
637         current_y += button_height + margin
```

```
638
639         self.UI_elements["Sens_height"] = pygame_gui.elements.
                UITextEntryLine(
640             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
641             manager=self.manager,
642             container=popup_window
643         )
644         current_y += button_height + margin
645
646         self.UI_elements["SensH_Submit"] = pygame_gui.elements.
                UIButton(
647             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
648             text="Submit",
649             manager=self.manager,
650             container=popup_window
651         )
652
653         self.manager.draw_ui(self.screen)
654         pygame.display.update()
655
656     def create_robot_popup(self):
657         # Calculate sizes for buttons and popup dimensions
658         button_width = self.WIDTH // 2 - self.WIDTH // 20
659         button_height = min(self.HEIGHT // 20, 60)
660         popup_width = self.WIDTH // 2
661         popup_height = self.HEIGHT
662         margin_left = (self.WIDTH - button_width)//20
663         margin = 20
664
665         # Center the popup on the screen
666         popup_rect = pygame.Rect(
667             (self.WIDTH - popup_width) // 2,
668             (self.HEIGHT - popup_height) // 2,
669             popup_width,
670             popup_height
671         )
672
673         popup_window = pygame_gui.elements.UIWindow(
674             rect=popup_rect,
675             manager=self.manager,
676             window_display_title='Place␣the␣robot␣?'
677         )
678
679         self.active_popup = popup_window
680
681         current_y = margin
682
```

```
683             header_label = pygame_gui.elements.UILabel(
684 
685                 relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
686                 text="Where␣do␣you␣want␣to␣place␣the␣robot␣?",
687                 manager=self.manager,
688                 container=popup_window
689             )
690             current_y += button_height + margin
691 
692             width_label = pygame_gui.elements.UILabel(
693                 relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
694                 text="Width␣(m):",
695                 manager=self.manager,
696                 container=popup_window
697             )
698             current_y += button_height + margin
699 
700             self.UI_elements["Robot_x"] = pygame_gui.elements.
                UITextEntryLine(
701                 relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
702                 manager=self.manager,
703                 container=popup_window
704             )
705             current_y += button_height + margin
706 
707             width_label = pygame_gui.elements.UILabel(
708                 relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
709                 text="Height␣(m):",
710                 manager=self.manager,
711                 container=popup_window
712             )
713             current_y += button_height + margin
714 
715             self.UI_elements["Robot_y"] = pygame_gui.elements.
                UITextEntryLine(
716                 relative_rect=pygame.Rect(margin_left, current_y,
                        button_width, button_height),
717                 manager=self.manager,
718                 container=popup_window
719             )
720 
721             current_y += button_height + margin
722 
723             width_label = pygame_gui.elements.UILabel(
```

```
724             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
725             text="Angle␣(  ):",
726             manager=self.manager,
727             container=popup_window
728         )
729         current_y += button_height + margin
730 
731         self.UI_elements["Robot_angle"] = pygame_gui.elements.
                UITextEntryLine(
732             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
733             manager=self.manager,
734             container=popup_window
735         )
736 
737         current_y += button_height + 2*margin
738 
739         self.UI_elements["Submit_robot_pos"] = pygame_gui.elements
                .UIButton(
740             relative_rect=pygame.Rect(margin_left, current_y,
                    button_width, button_height),
741             text="Validate",
742             manager=self.manager,
743             container=popup_window
744         )
745 
746         self.manager.draw_ui(self.screen)
747         pygame.display.update()
748 
749     def serial_comm(self):
750         data = self.run << 7 | self.kalman << 6 | self.test << 5 |
                self.stand << 4 | (self.x.real == 1) << 3 | (self.x.
                real == -1) << 2 | (
751                     self.x.imag == 1) << 1 | (self.x.imag == -1)
752         self.ser.write(bytes([data]))
753 
754         Content = self.ser.readline()
755         Content = Content.decode().replace("\r\n", "")
756         self.message = int(Content)
757 
758     def defined_trajectory(self, file):
759         if file != None:
760             with open("./trajectories/" + file + ".txt") as file:
761                 lines = file.readlines()
762                 for line in lines:
763                     action, duration = line.split("␣:␣")
764                     self.trajectory.append((action, duration))
765 
```

```
766             else:
767                 self.trajectory = None
768 
769         def is_click_image(self, name, event):
770             return self.rect_dict.get(name) != None and self.rect_dict
                    .get(name).collidepoint(event.pos)
771 
772         def load_image(self, room_num, object, side):
773             img = pygame.image.load(object.img)
774             img = pygame.transform.scale(img, (img.get_width() // 5,
                    img.get_height() // 5))
775 
776             name = object.type + "_" + side + "_" + str(room_num)
777             self.image_dict[name] = img
778 
779         def compute_pos_room(self, adapted_height, adapted_width, side
                , room_num):
780             x, y = self.rect_dict[self.temp_origin].center[0], self.
                    rect_dict[self.temp_origin].center[1]
781             if room_num == 0 :
782                 return x, y
783             else :
784                 match side :
785                     case "L":
786                         return (x - adapted_width//2), y
787                     case "R":
788                         return (x + adapted_width//2), y
789                     case "T":
790                         return x, (y + adapted_height//2)
791                     case "B":
792                         return x, (y - adapted_height//2)
793 
794         def compute_screen_size(self, width, height):
795             return int(width * (self.HEIGHT//self.RESIZE)), int(height
                    * (self.HEIGHT//self.RESIZE))
796 
797         def close_popup(self):
798             self.active_popup.kill()
799             self.active_popup = None
800 
801         def add_sides(self, room):
802             sides = ["L", "R", "T", "B"]
803             corners = ["TL", "TR", "BL", "BR"]
804             room_origin = int(self.temp_origin[7])
805             side_origin = self.temp_origin[5]
806 
807             if len(self.rooms) !=0:
808                 if side_origin == "L":
809                     sides.remove("R")
```

```
                room.add_room("R", self.rooms[room_origin])
            elif side_origin == "R":
                sides.remove("L")
                room.add_room("L", self.rooms[room_origin])
            elif side_origin == "T":
                sides.remove("B")
                room.add_room("B", self.rooms[room_origin])
            elif side_origin == "B":
                sides.remove("T")
                room.add_room("T", self.rooms[room_origin])

            self.rooms[room_origin].add_room(side_origin, room)

        for side in sides:
            room.modify_side(side, "./img/plus.png", "plus")
        for corner in corners:
            room.modify_side(corner, "./img/plus.png", "plus")

    def get_new_grid(self):
        leftmost_room = None
        rightmost_room = None

        upmost_room = None
        downmost_room = None

        x_min = self.WIDTH+1
        x_max = 0
        y_min = self.HEIGHT+1
        y_max = 0
        for room in self.rooms:

            if room.pos[0] < x_min:
                x_min = room.pos[0]
                leftmost_room = room
            if room.pos[0] > x_max:
                x_max = room.pos[0]
                rightmost_room = room

            if room.pos[1] < y_min:
                y_min = room.pos[1]
                upmost_room = room
            if room.pos[1] > y_max:
                y_max = room.pos[1]
                downmost_room = room

        x_min -= leftmost_room.width//2
        x_max += rightmost_room.width//2

        y_min -= upmost_room.height//2
```

```
859             y_max += downmost_room.height//2
860
861             self.room_grid = ((x_min, x_max), (y_min, y_max))
862
863         def get_real_pos(self, x, y):
864             grid_x = round((x - self.room_grid[0][0])/(self.HEIGHT/
                    self.RESIZE), 2)
865             grid_y = round((y - self.room_grid[1][0])/(self.HEIGHT/
                    self.RESIZE), 2)
866
867             return grid_x, grid_y
868
869         def get_screen_pos(self, grid_x, grid_y):
870             x = self.room_grid[0][0] + grid_x * (self.HEIGHT / self.
                    RESIZE)
871             y = self.room_grid[1][0] + grid_y * (self.HEIGHT / self.
                    RESIZE)
872             return x, y
873
874         def check_trajectory(self):
875             if self.is_trajectory_started:
876                     if self.trajectory != None :
877                         current_time = pygame.time.get_ticks() /
                                1000.0
878                         if self.action_start_time + self.
                                action_duration <= current_time and len(
                                self.trajectory) >= self.trajectory_idx +1:
879                             self.current_action = self.trajectory[self
                                    .trajectory_idx][0]
880                             self.action_duration = float(self.
                                    trajectory[self.trajectory_idx][1])
881                             self.action_start_time = pygame.time.
                                    get_ticks() / 1000.0
882                             self.trajectory_idx += 1
883
884         def main_loop(self):
885             while self.running:
886                 self.event_handler()
887                 self.update_screen_size()
888
889                 keys = pygame.key.get_pressed()
890                 self.x = 0
891                 self.string = ""
892
893                 if not self.in_popup:
894
895                     self.check_keys_movement(keys)
896                     self.check_keys_kalman(keys)
897                     self.check_test(keys)
```

```
898                     self.check_standing(keys)
899
900                 self.screen.fill((255, 255, 255))
901
902                 for room in self.rooms:
903                     self.draw_room(room)
904
905                 self.draw_move_ctrl()
906                 self.draw_buttons()
907                 self.draw_add_room()
908                 self.draw_string()
909                 self.draw_robot()
910                 #self.draw_grid()
911
912                 self.check_trajectory()
913
914                 self.manager.update(self.clock.tick(60)/1000)
915                 self.manager.draw_ui(self.screen)
916                 pygame.display.flip()
917
918                 try:
919                     self.serial_comm()
920                 except:
921                     print("[CONTROLLER] Error with LoRa Communication"
                        )
922
923         # Quit
924         pygame.quit()
925         sys.exit()
926
927 if __name__ == '__main__':
928     if len(sys.argv) == 3:
929         ui = User_interface(sys.argv[1], sys.argv[2], None)
930     else :
931         ui = User_interface(sys.argv[1], sys.argv[2], sys.argv[3])
932     ui.main_loop()
```

G.3 Components: Robot

```
class Robot:
    real_pos = (0,0)
    angle = 0
    room = -1
    ip = "1"
    port = 0

    def __init__(self):
        self.confirmed = False

    def update_adress(self, ip: str, port: int) -> None:
        self.ip = ip
        self.port = port

    def update_pos(self, x:float, y: float, angle: int, room: int)
        -> None:
        print("[ROBOT] is at " + str((x, y)) + " with an angle of
            " + str(angle) + " in room " + str(room))
        self.real_pos = (x, y)
        self.angle = angle
        self.room = room
```

G.4 Components: Room

```
from components.Side import Side

class Room:

    def __init__(self, width, height, x, y, room_num):
        self.width = width
        self.height = height
        self.pos = (x,y)
        self.sides = {
            "L":None,
            "R":None,
            "T":None,
            "B":None
                    }
        self.corners = {
            "TL":None,
            "TR":None,
            "BL":None,
            "BR":None
        }
```

```
21          self.room_num = room_num
22
23      def modify_side(self, side, img, img_type):
24          x, y = self.compute_pos(side)
25          if side in self.sides.keys() :
26              self.sides[side] = Side(x, y, img, img_type)
27          else :
28              self.corners[side] = Side(x, y, img, img_type)
29
30      def compute_pos(self, side):
31          match side :
32              case "L":
33                  side_x, side_y = self.pos[0] - self.width//2, self
                        .pos[1]
34              case "R":
35                  side_x, side_y = self.pos[0] + self.width//2, self
                        .pos[1]
36              case "T":
37                  side_x, side_y = self.pos[0], self.pos[1] + self.
                        height//2
38              case "B":
39                  side_x, side_y = self.pos[0], self.pos[1] - self.
                        height//2
40              case "TL":
41                  side_x, side_y = self.pos[0] - self.width//2, self
                        .pos[1] + self.height//2
42              case "TR":
43                  side_x, side_y = self.pos[0] + self.width//2, self
                        .pos[1] + self.height//2
44              case "BL":
45                  side_x, side_y = self.pos[0] - self.width//2, self
                        .pos[1] - self.height//2
46              case "BR":
47                  side_x, side_y = self.pos[0] + self.width//2, self
                        .pos[1] - self.height//2
48
49          return side_x, side_y
50
51
52      def add_room(self, side, room):
53          self.sides[side] = room
54
55      def update_size(self, resize, new_resize, height):
56          self.width, self.height = help_fun.compute_current_size(
                self.width, self.height, height, height, resize,
                new_resize)
57          for side in ["L", "R", "T", "B"]:
58              x, y = self.compute_pos(side)
59              self.sides[side].pos = (x,y)
```

```
60      def compute_current_size(self, width, height, past_height,
            current_height,  past_resize, current_resize):
61          new_width, new_height = int(width / (past_height//
                past_resize)), int(height / (past_height//past_resize))
62          new_width, new_height = int(new_width * (current_height//
                current_resize)), int(new_height * (current_height//
                current_resize))
63          return new_width, new_height
```

G.5 Components: Sensor

```
1  class Sensor :
2
3      def __init__(self, IP, Port, ID):
4          self.ip = IP
5          self.port = Port
6          self.id = ID
7          self.room = -1
8          self.x = -1
9          self.y = -1
10         self.height = 0
11         self.angle = 0
12         self.distance = -1
13
14     def set_angle(self, side):
15         match side:
16             case "L":
17                 self.angle = 0
18             case "TL":
19                 self.angle = 45
20             case "T":
21                 self.angle = 90
22             case "TR":
23                 self.angle = 135
24             case "R":
25                 self.angle = 180
26             case "BR":
27                 self.angle = 225
28             case "B":
29                 self.angle = 270
30             case "BL":
31                 self.angle = 315
32
33     def update_pos(self, room, x_corner, y_corner):
34         self.room = room
35
```

```
36          match self.angle:
37              case 0:
38                  x = x_corner
39                  y = y_corner
40              case 45:
41                  x = round(x_corner + 0.12, 2)
42                  y = round(y_corner - 0.16, 2)
43
44              case 90:
45                  x = x_corner
46                  y = y_corner
47              case 135:
48                  x = round(x_corner - 0.16, 2)
49                  y = round(y_corner - 0.12, 2)
50              case 180:
51                  x = x_corner
52                  y = y_corner
53              case 225:
54                  x = round(x_corner - 0.16, 2)
55                  y = round(y_corner + 0.12, 2)
56              case 270:
57                  x = x_corner
58                  y = y_corner
59              case 315:
60                  x = round(x_corner + 0.16, 2)
61                  y = round(y_corner + 0.12, 2)
62          self.x = x
63          self.y = y
64          print("[SENSOR_" + str(self.id) + "] is at (" + str(self.x
                ) + ", " + str(self.y) + ") in room number " + str(self
                .room) + " with an angle of " + str(self.angle))
65
66      def update_data(self, distance):
67          self.distance = distance
68
69      def update_height(self, height):
70          print("[SENSOR_" + str(self.id) + "] is at " + str(height)
                +"m from the ground")
71          self.height = height
```

G.6 Components: Side

```
class Side:

    def __init__(self, x, y, img, img_type):
        self.pos = (x,y)
        self.img = img
        self.type = img_type
```

G.7 csv_saver

```
import csv
import threading
import datetime
import time
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


class CSV_saver:
    def __init__(self):
        plt.switch_backend('Agg')
        date = datetime.datetime.now()
        self.clock_updater_instantiated = False
        self.sonar_pos_instantiated = False
        self.sonar_dist_instantiated = False
        self.kalman_pos_instantiated = False
        self.timestamp = str(date.year) + "_" + str(date.month) +
            "_" + str(date.day) + "_" + str(date.hour) + "_" + str(
            date.minute)

    def save_clock(self, num, clock):
        threading.Thread(target=self.csv_update_clock, args=(num,
            clock), daemon=True).start()

    def csv_update_clock(self, num, clock):
        tmstp = datetime.datetime.now().timestamp()

        with open("./data/clock_tick_" + self.timestamp + ".csv",
            "a") as csv_file:
            fieldnames = ["timestamp", "num", "clock"]
            csv_writer = csv.DictWriter(csv_file, fieldnames=
                fieldnames)
            row = {
                "timestamp": tmstp,
```

```
                "num": num,
                "clock": clock
                }

            csv_writer.writerow(row)

    def save_robot_pos_sonar(self, x, y, angle, room):
        threading.Thread(target=self.csv_update_pos_sonar, args=(x
            ,y,angle,room), daemon=True).start()

    def csv_update_pos_sonar(self, x, y, angle, room):
        tmstp = datetime.datetime.now().timestamp()
        if not self.sonar_pos_instantiated:
            self.sonar_pos_timestamp = tmstp
            self.sonar_pos_instantiated = True

        with open("./data/sonar_pos_" + self.timestamp + ".csv", "
            a") as csv_file:
            fieldnames = ["timestamp", "x", "y", "angle", "room"]
            csv_writer = csv.DictWriter(csv_file, fieldnames=
                fieldnames)
            row = {
                "timestamp": tmstp - self.sonar_pos_timestamp,
                "x": x,
                "y": y,
                "angle": angle,
                "room": room
                }

            csv_writer.writerow(row)

    def save_robot_pos_kalman(self, x, y, angle, room):
        threading.Thread(target=self.csv_update_pos_kalman, args=(
            x,y,angle,room), daemon=True).start()

    def csv_update_pos_kalman(self, x, y, angle, room):
        tmstp = datetime.datetime.now().timestamp()
        if not self.kalman_pos_instantiated:
            self.kalman_pos_timestamp = tmstp
            self.kalman_pos_instantiated = True

        with open("./data/kalman_pos_" + self.timestamp + ".csv",
            "a") as csv_file:
            fieldnames = ["timestamp", "x", "y", "angle", "room"]
            csv_writer = csv.DictWriter(csv_file, fieldnames=
                fieldnames)
            row = {
                "timestamp": tmstp - self.kalman_pos_timestamp,
                "x": x,
```

```
74                  "y": y,
75                  "angle": angle,
76                  "room": room
77                  }
78
79              csv_writer.writerow(row)
80
81      def save_distance_sonar(self, name, distance):
82          threading.Thread(target=self.csv_update_distance_sonar,
                args=(name, distance), daemon=True).start()
83
84      def csv_update_distance_sonar(self, name, distance):
85          tmstp = datetime.datetime.now().timestamp()
86
87          with open("./data/sonar_dist_" + name + "_" + self.
                timestamp + ".csv", "a") as csv_file:
88              fieldnames = ["timestamp", "dist"]
89              csv_writer = csv.DictWriter(csv_file, fieldnames=
                    fieldnames)
90              row = {
91                  "timestamp": tmstp,
92                  "dist": distance
93                  }
94
95              csv_writer.writerow(row)
96
97      def print_plots(self):
98          try :
99              self.create_dist_plots()
100             self.create_pos_kalman_plots()
101         except:
102             print("[CSV_SAVER] Error: something went wrong
                    printing the plots")
103
104
105     def create_dist_plots(self):
106
107         df_clock = pd.read_csv(f"./data/clock_tick_{self.timestamp
                }.csv", header=None, names=["timestamp", "num", "clock"
                ])
108         df1 = pd.read_csv(f"./data/sonar_dist_sensor_1_{self.
                timestamp}.csv",header=None, names=["timestamp", "dist"
                ])
109         df2 = pd.read_csv(f"./data/sonar_dist_sensor_2_{self.
                timestamp}.csv",header=None, names=["timestamp", "dist"
                ])
110         t0 = df_clock["timestamp"].iloc[0]
111
112         df_clock["timestamp"] -= t0
```

```
113         df1["timestamp"] -= t0
114         df2["timestamp"] -= t0
115 
116         xt = df_clock["timestamp"]
117         x1 = df1["timestamp"]
118         y1 = df1["dist"].astype(float).round(4)
119         x2 = df2["timestamp"]
120         y2 = df2["dist"].astype(float).round(4)
121 
122         fig, ax = plt.subplots(figsize=(20, 10))
123         ax.plot(x1, y1, label="Measured distance sensor 1",
                linewidth=1, marker='.')
124         ax.plot(x2, y2, label="Measured distance sensor 2",
                linewidth=1, marker='.')
125 
126         ax.set_ylim([0, 150])
127         ax.set_yticks(np.arange(0, 100, step=10))
128         ymin, ymax = ax.get_ylim()
129         ax.vlines(xt, ymin, ymax, linewidth=0.25, label="Clock
                tick", colors="red")
130 
131         ax.set_title("Sonar Distance Measurements with Clock Ticks
                ")
132         ax.set_xlabel("Time (s)")
133         ax.set_ylabel("Ground distance (cm)")
134 
135         ax.legend()
136         ax.grid(True)
137 
138         plt.savefig(f"./plots/dist_kalman_{self.timestamp}.png")
139 
140     def create_pos_kalman_plots(self):
141         pos_data = pd.read_csv(f"./data/kalman_pos_{self.timestamp
                }.csv", header=None, names=["timestamp", "x", "y", "
                angle", "room"])
142         x = pos_data['timestamp']
143         x_pos = pos_data['x'].astype(float)
144 
145         y_pos = pos_data['y'].astype(float)
146 
147         fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(20, 10),
                constrained_layout=True)
148 
149         ax1.plot(x, x_pos, label='Position on x_axis', linewidth
                =2, marker='.')
150         ax1.set_title('Variation of the position of the robot over
                 time')
151         ax1.set_ylabel('Position x axis')
152         ax1.set_ylim([0,1])
```

```
153         ax1.legend()
154         ax1.grid(True)
155 
156         ax2.plot(x, y_pos, label='Position␣on␣y_axis', linewidth
                =2, marker='.')
157         ax2.set_xlabel('Time')
158         ax2.set_ylabel('Position␣Y␣axis')
159         ax2.set_ylim([0,1])
160         ax2.legend()
161         ax2.grid(True)
162 
163         plt.savefig(f'./plots/pos_kalman_{self.timestamp}.png')
```

G.8 Server

```
1  import socket
2  import threading
3  from components.Sensor import Sensor
4  from components.Robot import Robot
5  from helping_package.csv_saver import CSV_saver
6  import time
7 
8  class Server:
9 
10     PORT = 5000
11     buffer = []
12     sensors = {}
13     room_edges = []
14     started = False
15 
16     def __init__(self, IP, BRD_IP):
17         self.HOST = IP
18         self.BRD_IP = BRD_IP
19         self.robot = Robot()
20         self.rcvServer = threading.Thread(target=self.rcv_server,
               daemon=True)
21         self.rcvServer.start()
22         self.pinger = threading.Thread(target=self.ping_server,
               daemon=True)
23         self.pinger.start()
24 
25         self.csv_saver = CSV_saver()
26 
27     def ping_server(self): # Broadcasts a ping message every 3
           seconds until the system is started
28         while not self.started :
```

```
29              time.sleep(3)
30              message = "ping : server , " + self.HOST + " , " + str
                    (self.PORT)
31              self.send(message, "brd")
32
33      def rcv_server(self): # Processes the messages received from
            the different devices
34          with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as
                server_socket:
35              server_socket.bind((self.HOST, self.PORT))
36              print(f"[SERVER] Listening for UDP packets on {self.
                    HOST}:{self.PORT}")
37
38              while True:
39                  data, addr = server_socket.recvfrom(1024)
40                  try :
41                      data = data.decode()
42                      data_split = data.split(",")
43
44                      if data_split[0] == "Hello": # Received from a
                             device when it has discovered the sever
45                          self.handle_hello(data_split, addr)
46                      elif data_split[0] == "Robot_pos": # Received
                            from devices at each iteration of the
                            kalman measure (only saves robot update)
47                          self.update_robot_pos_kalman(data, addr)
48                      elif data_split[0] == "Robot_sonar_pos": #
                            Received from devices at each iteration of
                            the kalman measure (only saves robot update
                            )
49                          self.update_robot_pos_sonar(data, addr)
50                      elif data_split[0] == "Ack": # Received from
                            devices after each configuration message
51                          self.handle_ack(data)
52                      elif data_split[0] == "Distance":
53                          self.csv_saver.save_distance_sonar(
                                data_split[2], data_split[1])
54                      elif data_split[0] == "Clock":
55                          self.csv_saver.save_clock(data_split[1],
                                data_split[2])
56                      else : # Default case
57                          print("[SERVER] received strange data : "
                                + data)
58                  except :
59                      pass
60
61      def handle_hello(self, data, addr):
62          # Processes the hello message received, creates new sensor
                in the case of a sensor, updates robot otherwise
```

```
63          # @param data : the decoded data received (String)
64          # @param addr : the Ip address and Port associated with
                the received message (List)
65
66          id = data[1]
67          if id == "robot":
68              print("[SERVER] Received hello from Robot on (" + addr
                    [0] + ", " + str(addr[1]) + ")")
69              self.robot.update_adress(addr[0], addr[1])
70              self.send("Ack, server", "uni", "robot")
71          else :
72              id = int(id)
73              self.sensors[id] = Sensor(addr[0], addr[1], id)
74              print("[SERVER] Received hello from sensor_" + str(id)
                     + " on (" + str(addr[0]) + ", " + str(addr[1]) + "
                    )")
75              self.send("Ack , server", "uni", id)
76
77      def update_robot_pos_sonar(self, data, addr):
78          # Updates the robot position based on the message received
79          # @param data : the decoded data received (String)
80          # @param addr : the Ip address and Port associated with
                the received message (List)
81
82          data_split = data.strip().split(",")
83          if addr[0] == self.robot.ip:
84              self.csv_saver.save_robot_pos_sonar(float(data_split
                    [1]), float(data_split[2]), float(data_split[3]),
                    int(data_split[4]))
85
86      def update_robot_pos_kalman(self, data, addr):
87          # Updates the robot position based on the message received
88          # @param data : the decoded data received (String)
89          # @param addr : the Ip address and Port associated with
                the received message (List)
90
91          data_split = data.strip().split(",")
92          if addr[0] == self.robot.ip:
93              self.csv_saver.save_robot_pos_kalman(float(data_split
                    [2]), float(data_split[3]), float(data_split[4]),
                    int(data_split[5]))
94              self.robot.update_pos(float(data_split[2]), float(
                    data_split[3]), float(data_split[4]), int(
                    data_split[5]))
95
96      def handle_ack(self, data):
97          # Processes the ack message, updates the ack list
98          # @param data : the decoded data received (String)
99
```

```
100         data_split = data.split(",")
101         config_message = data_split[1]
102         id = data_split[2]
103         origin = data_split[3]
104         if origin != "robot":
105             if config_message == "Pos":
106                 self.ack_pos.get(id)[int(origin)-1] = True
107             elif config_message == "Room_info":
108                 self.ack_rooms.get(int(id))[int(origin)-1] = True
109             elif config_message == "Add_Link":
110                 self.ack_propag.get("sensor_"+origin)[int(id.split
                        ("_")[1])-1] = True
111             else :
112                 self.ack_devices.get(id)[int(origin)-1] = True
113         else :
114             if config_message == "Pos":
115                 self.ack_pos.get(id)[len(self.sensors.keys())] =
                        True
116             elif config_message == "Room_info":
117                 self.ack_rooms.get(int(id))[len(self.sensors.keys
                        ())] = True
118             else :
119                 self.ack_devices.get(id)[len(self.sensors.keys())]
                         = True
120         print("[SERVER] Received Ack " + config_message + " for "
               + id + " from " + origin)
121
122     def send(self, message, type, id=None):
123         # Creates a sending server thread of the specified type
                and passes arguments
124         # @param message: the message to be sent (String)
125         # @param type: the way the message has to be sent, can be
                "brd" for broadcast and "uni" for unicast (String)
126         # @param id: the identifier of the device to update (
                String, Integer, None)
127         if message == "Exit":
128             self.csv_saver.print_plots()
129
130         if type == "brd":
131             threading.Thread(target=self.brd_server, args=(message
                    ,), daemon=True).start()
132         elif type == "uni":
133             threading.Thread(target=self.uni_server, args=(message
                    , id), daemon=True).start()
134
135     def brd_server(self, message):
136         # Sends a broadcast message
137         # @param message: the message to be sent (String)
138
```

```
139         with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as
                srv_socket:
140             srv_socket.setsockopt(socket.SOL_SOCKET, socket.
                    SO_BROADCAST, 1)
141 
142             port = 9000
143             srv_socket.sendto(message.encode(), (self.BRD_IP, port
                    ))
144             if message[:4] != "ping":
145                 print(f"[SERVER] Broadcasted to ({self.BRD_IP}, {
                        port}): {message}")
146 
147     def uni_server(self, message, id):
148         # Sends a unicast message to the specified device
149         # @param message: the message to be sent (String)
150         # @param id: the identifier of the device to update (
                String, Integer)
151 
152         with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as
                srv_socket:
153             if id == "robot":
154                 ip = self.robot.ip
155                 port = self.robot.port
156                 srv_socket.sendto(message.encode(), (ip, port))
157                 print("[SERVER] Sent to Robot on (" + str(ip) + ",
                        " + str(port) + ") : " + str(message))
158             else :
159                 sensor = self.sensors.get(id)
160                 ip = sensor.ip
161                 port = sensor.port
162                 srv_socket.sendto(message.encode(), (ip, port))
163                 print("[SERVER] Sent to " + str(sensor.id) + " on
                        (" + str(ip) + ", " + str(port) + ") : " + str(
                        message))
164 
165     def send_config(self): # Creates a worker_send_config thread
166         threading.Thread(target=self.worker_send_config, daemon=
                True).start()
167 
168     def worker_send_config(self): # Sends the whole config to all
            the devices to setup the system
169         self.started = True
170         self.ack_devices = {}
171         self.ack_propag = {}
172         self.ack_pos = {}
173         self.ack_rooms = {}
174 
175         for sensor in self.sensors.values() :
176             sensor_config_ok = self.send_sensor_infos(sensor)
```

```
177
178         if sensor_config_ok:
179             room_config_ok = self.send_rooms_infos()
180         else:
181             room_config_ok = False
182
183         if room_config_ok:
184             self.send_robot_info()
185
186             time.sleep(1)
187             self.started = True
188             for i in range(4):
189                 message = "Start " + self.HOST
190                 self.send(message, "brd")
191                 time.sleep(0.5)
192
193         else :
194             self.send("Exit", "brd")
195
196     def send_sensor_infos(self, sensor):
197         # Sends all the informations about a sensor to all the
                devices
198         # @param sensor: the actual sensor from which we draw the
                info (Sensor)
199         # @return a ack boolean if the informations of this sensor
                were successfully delivered to everyone, false
                otherwise
200
201         # Init ack status
202         self.ack_devices["sensor_" + str(sensor.id)] = [False for
                i in range(len(self.sensors.keys()) + 1)]
203         self.ack_devices["robot"] = [False for i in range(len(self
                .sensors.keys()) + 1)]
204         self.ack_propag["sensor_"+str(sensor.id)] = [True for i in
                range(len(self.sensors.keys())+1)]
205         self.ack_pos["sensor_" + str(sensor.id)] = [False for i in
                range(len(self.sensors.keys()) + 1)]
206         self.ack_pos["robot"] = [False for i in range(len(self.
                sensors.keys()) + 1)]
207         self.set_unknown_devices(sensor)
208
209         if sensor.x != -1 :
210             ack = False
211             LIMIT = 0
212             while (not ack) and (LIMIT < 10):
213                 message = "Add_Device : sensor_" + str(sensor.id)
                        + " , " + sensor.ip + " , " + str(sensor.port)
214                 for i in range(len(self.ack_devices["sensor_"+str(
                        sensor.id)])-1):
```

```
215                     osensor = self.ack_devices["sensor_"+str(
                            sensor.id)][i]
216                     if not osensor :
217                         self.send(message, "uni", i+1)
218 
219                 if not self.ack_devices["sensor_"+str(sensor.id)][
                        len(self.sensors.keys())]:
220                     self.send(message, "uni", "robot")
221                 time.sleep(0.25)
222                 message = "Pos " + str(sensor.id) + " : " + str(
                        sensor.x) + " , " + str(sensor.y) + " , " + str
                        (sensor.height) + " , " + str(sensor.angle) + "
                         , " + str(sensor.room)
223                 for i in range(len(self.ack_pos["sensor_"+str(
                        sensor.id)])-1):
224                     osensor = self.ack_pos["sensor_"+str(sensor.id
                            )][i]
225                     if not osensor :
226                         self.send(message, "uni", i+1)
227 
228                 if not self.ack_pos["sensor_"+str(sensor.id)][len(
                        self.sensors.keys())]:
229                     self.send(message, "uni", "robot")
230                 time.sleep(0.25)
231 
232 
233                 #Propag config
234                 for i in range(len(self.ack_propag["sensor_" + str
                        (sensor.id)])-1):
235                     is_acked = self.ack_propag["sensor_"+str(
                            sensor.id)][i]
236                     osensor = self.sensors[i+1]
237                     if not is_acked :
238                         message = "Add_Link : sensor_" + str(
                                osensor.id)+","+ osensor.ip + "," + str
                                (osensor.port)
239                         self.send(message, "uni", sensor.id)
240 
241                 time.sleep(0.25)
242                 ack = self.check_ack("sensor_" + str(sensor.id), "
                        sensor")
243                 LIMIT += 1
244 
245             return ack
246 
247     def send_robot_info(self): # Sends all the informations about
            the robot to all the devices
248         if self.robot.ip != "0":
249             ack = False
```

```
250             LIMIT = 0
251             while (not ack) and (LIMIT < 10):
252                 message = "Add_Device␣:␣robot␣,␣" + self.robot.ip
                        + "␣,␣" + str(self.robot.port)
253                 self.send(message, "brd")
254                 time.sleep(0.5)
255                 message = "Init_pos␣:␣" + str(self.robot.real_pos
                        [0]) + "␣,␣" + str(self.robot.real_pos[1]) + "␣
                        ,␣" + str(self.robot.angle) + "␣,␣" + str(self.
                        robot.room)
256                 self.send(message, "brd")
257
258                 ack = self.check_ack("robot", "sensor")
259                 LIMIT +=1
260
261     def send_rooms_infos(self):
262
263         for room_idx in range(len(self.room_edges)):
264             self.ack_rooms[room_idx] = [False for i in range(len(
                    self.sensors.keys())+1)]
265             ack = False
266
267             LIMIT = 0
268             while (not ack) and (LIMIT < 10):
269                 message = "Room_info," + str(room_idx)
270                 message += "," + str(self.room_edges[room_idx
                        ][0][0])
271                 message += "," + str(self.room_edges[room_idx
                        ][0][1])
272                 message += "," + str(self.room_edges[room_idx
                        ][1][0])
273                 message += "," + str(self.room_edges[room_idx
                        ][1][1])
274                 self.send(message, "brd")
275                 time.sleep(0.5)
276
277                 ack = self.check_ack(room_idx, "room")
278                 LIMIT += 1
279
280             if not ack:
281                 return False
282         return True
283
284     def check_ack(self, id, type):
285         # Checks that all devices have acknowledged all the
                informations about the current device
286         # @return True if all devices acked, False otherwise
287
288         if type == "sensor":
```

```
289             for i in range(len(self.sensors.keys())+1):
290                 if not self.ack_devices.get(id)[i]:
291                     return False
292                 if not self.ack_pos.get(id)[i]:
293                     return False
294                 if id != "robot" and not self.ack_propag.get(id)[i
                        ]:
295                     return False
296             return True
297         elif type == "room":
298             for i in range(len(self.sensors.keys())+1):
299                 if not self.ack_rooms.get(id)[i]:
300                     return False
301             return True
302 
303     def set_unknown_devices(self, sensor):
304         for i in range(len(self.sensors.keys())):
305             sensor2 = self.sensors[i+1]
306             if (sensor2.room!=sensor.room):
307                 self.ack_devices["sensor_"+str(sensor.id)][i] =
                        True
308                 self.ack_pos["sensor_"+str(sensor.id)][i] = True
309 
310             if (sensor2.room == sensor.room-1) or (sensor2.room ==
                    sensor.room+1):
311                 self.ack_propag["sensor_"+str(sensor.id)][i] =
                        False
312 
313         print(f"for␣sensor_{sensor.id}␣propag:␣{self.ack_propag["
            sensor_"+str(sensor.id)]}")
314         print(f"for␣sensor_{sensor.id}␣devices:␣{self.ack_devices[
            "sensor_"+str(sensor.id)]}")
315         print(f"for␣sensor_{sensor.id}␣pos:␣{self.ack_pos["sensor_
            "+str(sensor.id)]}")
316 
317     def get_sensors(self):
318         # @return all the known sensors ids
319 
320         return self.sensors.keys()
321 
322     def update_sens(self, id, side, room, x, y):
323         # Updates the position of a sensor in a room
324         # @param id: the id of the sensor (Integer)
325         # @param side: the side or corner in which the sensor has
                been placed (String)
326         # @param room: the room in which the sensor has been
                placed (Integer)
327         # @param x: the x-axis position of the sensor in the grid
                (float)
```

```
328         # @param y: the y-axis position of the sensor in the grid
                (float)
329
330         sens = self.sensors.get(id)
331         sens.set_angle(side)
332         sens.update_pos(room, x, y)
333
334     def update_sens_height(self, id, height):
335         # update the height of the sensor
336         # @param id: the id of the sensor to modify (Integer)
337         # @param height: the height of the sensor (Float)
338
339         self.sensors.get(id).update_height(height)
340
341     def update_robot(self, real_pos, angle):
342         # Updates the robot position
343         # @param real_pos: the position of the robot on the grid (
                Tuple)
344         # @param angle: the angle of the robot (0 <= Integer <=
                360)
345         # @param room: the room in which the robot is placed (
                Integer)
346
347         self.robot.update_pos(real_pos[0], real_pos[1], angle,
                self.determine_robot_room(real_pos[0], real_pos[1]))
348
349     def add_edges(self, TLpos, BRpos):
350
351         self.room_edges.append((TLpos, BRpos))
352
353     def determine_robot_room(self, x, y):
354         for room_idx in range(len(self.room_edges)):
355             if self.is_in_x_range(x, room_idx) and self.
                    is_in_y_range(y, room_idx):
356                 return room_idx
357         return -1
358
359     def is_in_x_range(self, x, room_idx):
360         return x > self.room_edges[room_idx][0][0] and x < self.
                room_edges[room_idx][1][0]
361
362     def is_in_y_range(self, y, room_idx):
363         return y > self.room_edges[room_idx][0][1] and y < self.
                room_edges[room_idx][1][1]
```

G.8.1 Runner script

```
#!/bin/bash

#Created with the help of ChatGPT

MAX_TRIES=10
COUNT=0

DEVICE_IP=$(ip route get 1.1.1.1 | awk '{for(i=1;i<=NF;i++) if($i=="src") print $(i+1)}')
BROADCAST_IP=$(ip -o -f inet addr show | awk '/scope global/ {print $6}' | head -n1)

while [ $COUNT -lt $MAX_TRIES ]; do
    if [ $# -eq 1 ]; then
        FILENAME="$1"
        python3 Controller.py "$DEVICE_IP" "$BROADCAST_IP" "$FILENAME"
    fi

    if [ $# -lt 1 ]; then
        python3 Controller.py "$DEVICE_IP" "$BROADCAST_IP"
    fi

    EXIT_CODE=$?
    if [ $EXIT_CODE -eq 0 ]; then
        echo "Controller.py exited successfully."
        exit 0
    else
        COUNT=$((COUNT + 1))
        echo "Controller.py failed (attempt $COUNT/$MAX_TRIES). Retrying in 0.5s..."
        sleep 0.5
    fi
done

echo "Controller.py failed $MAX_TRIES times. Exiting."
exit 1
```

Appendix H

LilyGo Software

H.1 LilyGo Robot software

```
#include <SPI.h>
#include <LoRa.h>
#include <Arduino.h>
#include <Wire.h>
#include <WiFi.h>

#include "motor_engine.h"

#define BAND 433E6
#define CONFIG_MOSI 27
#define CONFIG_MISO 19
#define CONFIG_CLK  5
#define CONFIG_NSS  18
#define CONFIG_RST  23
#define CONFIG_DIO0 26

#define SDCARD_MOSI 15
#define SDCARD_MISO 2
#define SDCARD_SCLK 14
#define SDCARD_CS   13

#define I2C_SLAVE_ADDR 0x40

// trapezoidal speed command parametter for turning, migration on
   GRiSP for this part
#define max_turn_speed 80
#define turn_acc 400

const char* ssid = "RobotNet";
const char* password = "oui123456";
```

```
30
31 WiFiServer server(80);
32
33 float I2C_command[2] = {0.0, 0.0}; // value received from GRiSP :
      {wheels acceleration , turn speed}
34
35 float freq_lim [13] =
      {300,200,175,150,125,100,90,80,70,60,50,40,30};
36 int size_test_freq = sizeof(freq_lim)/sizeof(freq_lim[0]);
37 int index_lim = 0;
38
39 // time mesure variable
40 unsigned long t_GRiSP;
41 unsigned long t_LORA;
42 unsigned long t_test;
43 unsigned long t_ESP;
44
45 // freq and period variable
46 float dt_GRiSP = 10;
47 float freq_GRiSP = 200;
48 float dt_ESP = 0;
49
50 // control byte received
51 byte cmd = 0; // received from LoRa communication and transfered
      to GRiSP
52 byte GRiSP_flags = 0; // Received from GRiSP
53
54 //control flag
55 bool new_cmd =false;
56 bool test = false;
57 bool disturb = false;
58 bool ext_end = true;
59
60
61 void setup() {
62   Serial.begin(115200);
63
64   setup_wifi();
65
66   setup_LoRa();
67
68   setup_I2C_slave();
69
70   setup_motor();
71
72   // time init
73   t_GRiSP = millis();
74   t_LORA = t_GRiSP;
75   t_ESP = t_GRiSP;
```

```
}


void loop() {
  unsigned long new_t_ESP = millis();
  dt_ESP = (new_t_ESP - t_ESP) / 1000.0;
  t_ESP = new_t_ESP;
  LoRa_receiver();
  Event_handle();
  delay(1);
}

void setup_wifi(){
  Serial.println("[ROBOT] RobotAP setting up ...");
  WiFi.softAP(ssid, password, 1, false, 8);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("[ROBOT] AP IP address : ");
  Serial.println(IP);
  server.begin();
  Serial.println("[ROBOT] RobotAP ready !");
}

void setup_LoRa(){
  Serial.println("[ROBOT] LoRa setting up ...");
  SPI.begin(CONFIG_CLK, CONFIG_MISO, CONFIG_MOSI, CONFIG_NSS);
  LoRa.setPins(CONFIG_NSS, CONFIG_RST, CONFIG_DIO0);
  if (!LoRa.begin(BAND)) {
    Serial.println("[ROBOT] Starting LoRa failed!");
    while (1);
  }
  Serial.println("[ROBOT] LoRa Ready !");
}

void setup_I2C_slave(){
  // I2C Slave init, work with IRQ so no need to incorporate into
     the main loop
  Wire.begin(I2C_SLAVE_ADDR);
  Wire.onReceive(GRiSP_receiver);
  Wire.onRequest(GRiSP_sender);
}

void setup_motor(){
  // motor init, works on core n 2
  engine_init();
  delay(1000);
  set_speed(0, 0);
  set_acceleration(0, 0);
}
```

```
124
125 void GRiSP_receiver(int howMany) {
126   if(howMany == 5){ // check if the packet match the expected
        lenght
127     unsigned long new_t_GRiSP = millis();
128     dt_GRiSP = (new_t_GRiSP - t_GRiSP) / 1000.0;
129     freq_GRiSP = freq_GRiSP * 0.99 + 1.0 / dt_GRiSP * 0.01;
130     t_GRiSP = new_t_GRiSP;
131
132     byte A;
133     byte B;
134     if (Wire.available()) {
135
136       // read and decode the wheel acceleration
137       A = Wire.read();
138       B = Wire.read();
139       I2C_command[0] = decoder(A, B);
140
141       // read and decode the differential turn speed
142       A = Wire.read();
143       B = Wire.read();
144       I2C_command[1] = decoder(A, B);
145
146       // read the flags
147       GRiSP_flags = Wire.read();
148     }
149
150     //set acceleration
151     if(!disturb && bitRead(GRiSP_flags, 4) && !bitRead(GRiSP_flags
          , 6)){
152       set_acceleration(I2C_command[0], I2C_command[0]);
153     }
154
155     // Free fall, null wheel speed
156     if(bitRead(GRiSP_flags, 6)){
157       set_acceleration(0, 0);
158       set_speed(0, 0);
159     }
160
161     // extension/retraction of the rising system
162     if(bitRead(GRiSP_flags, 5)){
163       ext_end = stand(-48,30.0);
164     } else {
165       ext_end = stand(0,30.0);
166     }
167
168     // wheel counter rotation activation and direction selection
          during rise
169     int stand_speed_dir = 0;
```

```
170     if(!bitRead(GRiSP_flags, 4)){ // if down
171       stand_speed_dir = (bitRead(GRiSP_flags, 3)) ? -1 : 1;
172     }
173     raise_dir(stand_speed_dir); // -1 back, 0 null, 1 front
174   }
175
176   // Empty the stack
177   while(Wire.available()){
178     Wire.read();
179   }
180 }
181
182 void GRiSP_sender() {
183   byte v[5];
184   float* speeds = get_speed();
185   encoder(v, speeds[0]);
186   encoder(v+2, speeds[2]);
187
188   // control byte send to GRiSP witth : finsish extension/
         retraction flag and the command inputs
189   v[4] = (cmd & 127) | (is_ready() * 128);
190
191   //send
192   Wire.write((byte*) v, sizeof(v));
193 }
194
195 double decoder(byte X, byte Y) {
196   // decode half float to double
197
198   byte A = (X & 192);
199   if ((A & 64) == 0) A = A | 63;  // fill the missing exponnent
         bytes with the right value
200   byte B = ((X << 2) & 252) | ((Y >> 6) & 3);
201   byte C = ((Y << 2) & 252);
202
203   byte vals[] = { 0x00, 0x00, 0x00, 0x00, 0x00, C, B, A };
204   double d = 0;
205   memcpy(&d, vals, 8);
206
207   return d;
208 }
209
210 void encoder(byte* res, double X){
211   // encode double to half float
212   byte vals[8];
213   memcpy(vals, &X,8);
214   byte A = vals[7];
215   byte B = vals[6];
216   byte C = vals[5];
```

```
217
218   res[0] = (A&192)|((B>>2)&63);
219   res[1] = ((B<<6)&192)|((C>>2)&63);
220
221   return ;
222 }
223
224 void LoRa_receiver(){
225   // receiption of LoRa packets
226   if (LoRa.parsePacket()) {
227     if(LoRa.available()>=2){
228       byte cmd1 = LoRa.read();
229       byte cmd2 = LoRa.read();
230       if(cmd1 == cmd2){
231         cmd = cmd1;
232         new_cmd = true;
233       }
234       Serial.println(cmd1);
235       Serial.println(cmd2);
236     }
237     while (LoRa.available()){
238       LoRa.read();
239     }
240   }
241 }
242
243 void Event_handle(){
244
245   //emergency stop
246   emergency(!bitRead(cmd, 7) || !bitRead(GRiSP_flags, 7));
247   if (!bitRead(cmd, 7) || !bitRead(GRiSP_flags, 7)){
248     set_acceleration(0, 0);
249     set_speed(0, 0);
250   }
251
252
253   // start test procedure
254   if(bitRead(cmd, 5)){
255     test = true;
256     t_test = millis();
257   }
258   if(test){
259     //start the disturbance 500ms to let the record start
260     if(millis()> t_test + 500){
261       //disturb = true;
262       //set_acceleration(40, 40);
263     }
264     // the disturbance is only applied between t=500 and t=800
265     if(millis()> t_test + 800){
```

```
      disturb = false;
      test = false;
    }
  }

  set_turn(I2C_command[1]);

  new_cmd =false;
}
```

H.2 LilyGo User software

```
#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>


#define Pin 15
#define Buzz 13

#define LORA_PERIOD 433
#define BAND 433E6


#define CONFIG_MOSI 27
#define CONFIG_MISO 19
#define CONFIG_CLK  5
#define CONFIG_NSS  18
#define CONFIG_RST  23
#define CONFIG_DIO0 26

unsigned long t ;
int state = 0;
int prevstate = 0;
byte cmd;

void setup()
{
  // init serial
  Serial.begin(115200);
  while (!Serial);
  // init LoRa
  SPI.begin(CONFIG_CLK, CONFIG_MISO, CONFIG_MOSI, CONFIG_NSS);
  LoRa.setPins(CONFIG_NSS, CONFIG_RST, CONFIG_DIO0);
  if (!LoRa.begin(BAND)) {
    Serial.println("Starting␣LoRa␣failed!");
```

```
35      while (1);
36    }
37
38    //init Pin
39    pinMode(Pin, OUTPUT);
40    pinMode(Pin, INPUT_PULLUP);
41    pinMode(Buzz, OUTPUT);
42    digitalWrite(Buzz, LOW);
43
44    prevstate = esp_sleep_get_wakeup_cause()!=ESP_SLEEP_WAKEUP_TIMER
          && !digitalRead(Pin);
45    t = millis();
46  }
47
48  int count = 0;
49
50  //main loop
51  void loop(){
52    Keyboard_input();
53    LoRA_sender();
54  }
55
56  void LoRA_sender(){
57
58    state = LOW;
59
60    if(state==HIGH){
61      cmd = cmd & 127;
62    }
63
64    LoRa.beginPacket();
65    // send two packet for redundancy
66    LoRa.write(cmd);
67    LoRa.write(cmd);
68    LoRa.endPacket();
69    Serial.println(cmd, BIN);
70
71
72    //buzzer logic
73    if(state == HIGH && prevstate == LOW){
74      digitalWrite(Buzz, HIGH);
75      delay(100);
76      digitalWrite(Buzz, LOW);
77    }
78
79
80    if(state == LOW && prevstate == HIGH){
81      digitalWrite(Buzz, HIGH);
82      delay(100);
```

```
    digitalWrite(Buzz, LOW);
  }
  prevstate = state;
}


void Keyboard_input(){
  if (Serial.available() > 0) {
    cmd = Serial.read();
  }

  while(Serial.available()){
    Serial.read();
  }
}
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**