CentraleSupélec

La Fabrique

Robot Explorateur

Encadrant:
Frédéric Boulanger

Group:
Rodrigo Kappes Marques
Marinus Sewalt

May 31, 2021

Contents

1 Introduction

1.1 The main problem

We wish to create a robot to number the quantity of people inside a building or a certain room. The need for this information can be linked with the current coronavirus crisis and the control of the flux of people inside a room, or even to give precise information about the occupation status in the case of an emergency evacuation. Another problem taken into consideration for the scope of the mission is the guarantee of the respect of safety measurements around heavy machinery, where a crew of at least two people is needed to operate and regulate such dangerous devices.

1.2 Our mission

Taking into consideration the problem we wish to resolve, we need to create an autonomous/controllable robot that can explore a place and analyse its surroundings to flexibly determine the respect of special crew constraints. Depending on the use and application, the constraints to be respected may vary, but the overall comportment of the robot should stay the same, as it will gather and process the same kind of information.

1.3 The kit

As a prototype for our explorer robot, we use the romi kit, as it offers a great mechanical base for the electronics, and it also gives a good motor driver and motor with encoders included.

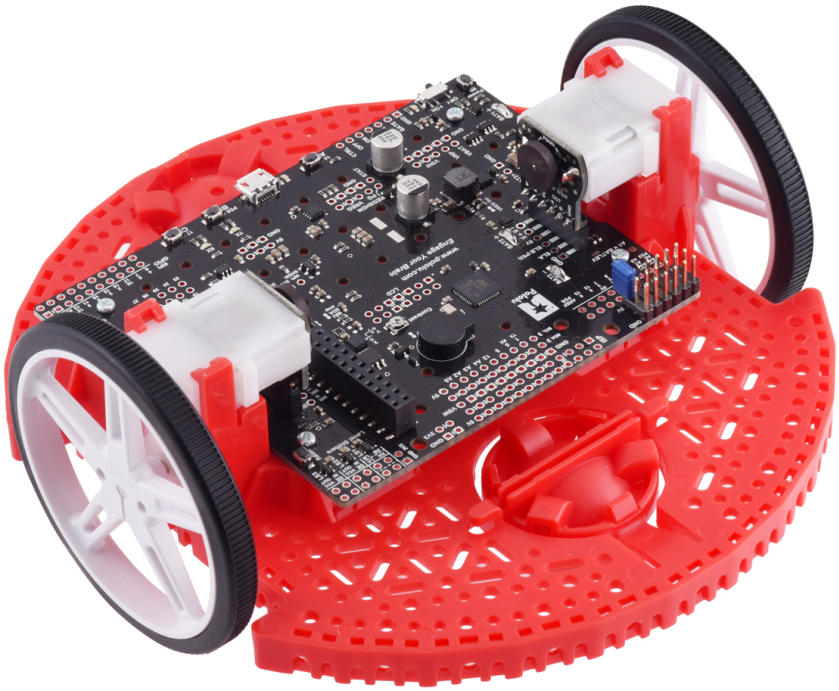


Figure 1: The romi kit already built

The kit comes unassembled and requires some work and electronic welding to be finished as shown in the picture.

1.4 Description of the work

1.4.1 Specifications

- S1 : Web interface for interaction that is able to check the robot operation and its modules, and be intuitive and easy to use
- S2 : Modularity : automatic detection of the present modules allowing the possibility to easily add new modules
- S3 : 1 to 2 hours of autonomy(several days if not used), creating a sleep mode and awake mode to increase the number of hours of operation
- S4 : Obstacle detection module, with the possibility of different obstacle detection methos for the base layer(locomotion) as well as for the top layer(web interface)
- S5 : Movement module that receives instructions from the central unit, and robust to bugs, avoiding at all costs unecessary collisions

1.4.2 Constraints

- C1 : Usage of the EPS32 to satisfy S1
- C2 : 6xAA for E3 allowing autonomy
- C3 : I2C for the E2 communication
- C4 : A Lidar module to satisfy 34, and a micro-controller with I2C for compatibility with E2 and C3
- C5 : Motor control for E5, a base layer that can be autonomous from the rest of the sustem, using arduino or pyboard, using several all-or-nothing sensors and the romi motors

1.4.3 Future constraints

- FC1 : Usage of the OpenMv
- FC2 : Facial recocnition and detection
- FC3 : Mapping the explored environment

1.4.4 Fundamental elements

- FE1 : Romi chasis
- FE2 : ESP32
- FE3 : Arduino
- FE4 : Pyboard
- FE5 : Infrared sensors
- FE6 : Rplidar
- FE7 : Motor driver and power distribution board for romi chasis
- FE8 : Raspberry Pi
- FE9 : Wires
- FE10 : Soldering station
- FE11 : Access to the laser cutter
- FE12 : Switch sensors

1.5 Description of how we worked

We worked well together considering that our group is relatively small, which contains only two people. Most of the work done in the base layer of locomotion and communication was done by Rodrigo, while most of the work done in the upper layers of network interface was done by Marinus. This proved to be the most effective division of work given our different strengths in different coding languages as well as our personal preferences.

Given the division of the work, we learned a lot about web interfaces and how to make a good and smooth control of a robot. This document was made to explain exactly what we made and what knowledge we took from this project.

2 Robot architecture

The goal of our robot is to be modular. By doing so it is easy to divide up the work among a group and it will be easier for another group to pick up our work to further develop the robot. We thus decided on making a robot with multiple micro-controllers, each one dedicated to a specific task. The micro-controllers are connected to one another over an I2C bus. The current architecture is the following:

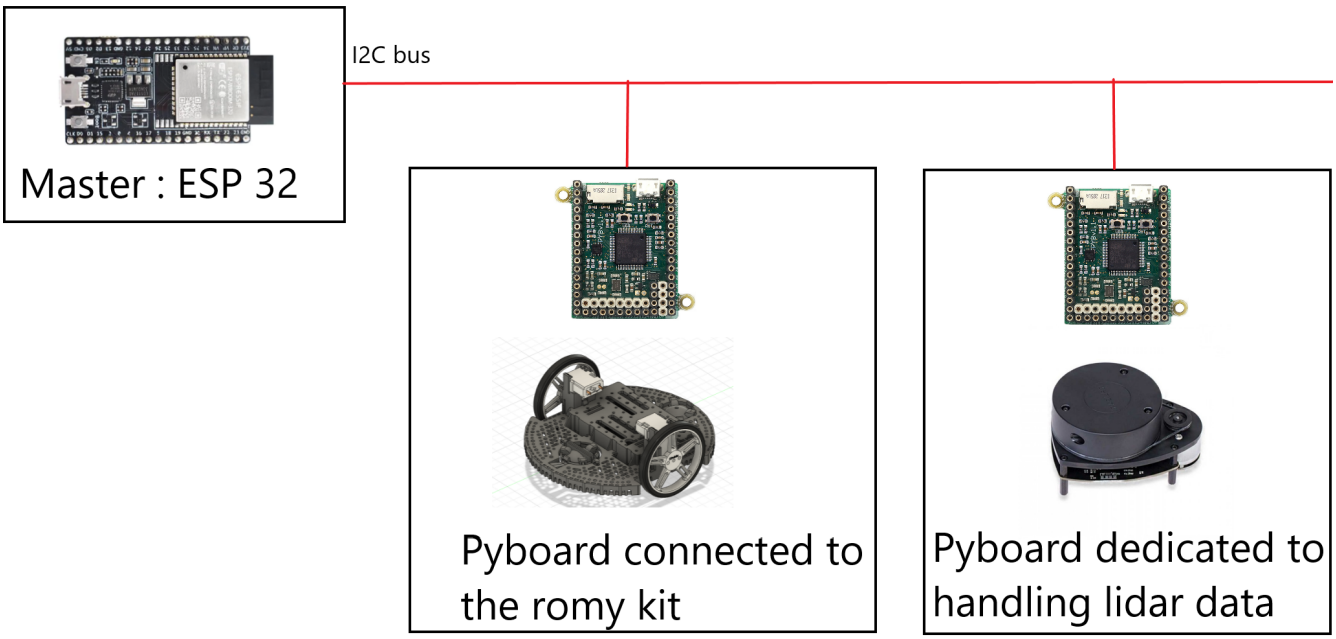


Figure 2: Robot architecture

More modules can easily be added as they are all in-dependant. Only, the code of the master module has to be adapted in order to use more modules.

3 Locomotion module

3.1 Introduction

The first part of the work was linked with the base layer of the robot, the locomotion layer. To result in a smooth and developed control, we chose the cascade PID control, as it would give us a smooth and powerful control for our motors, as well as the opportunity to expand our automation knowledge. The following diagram explains the implemented robot control.

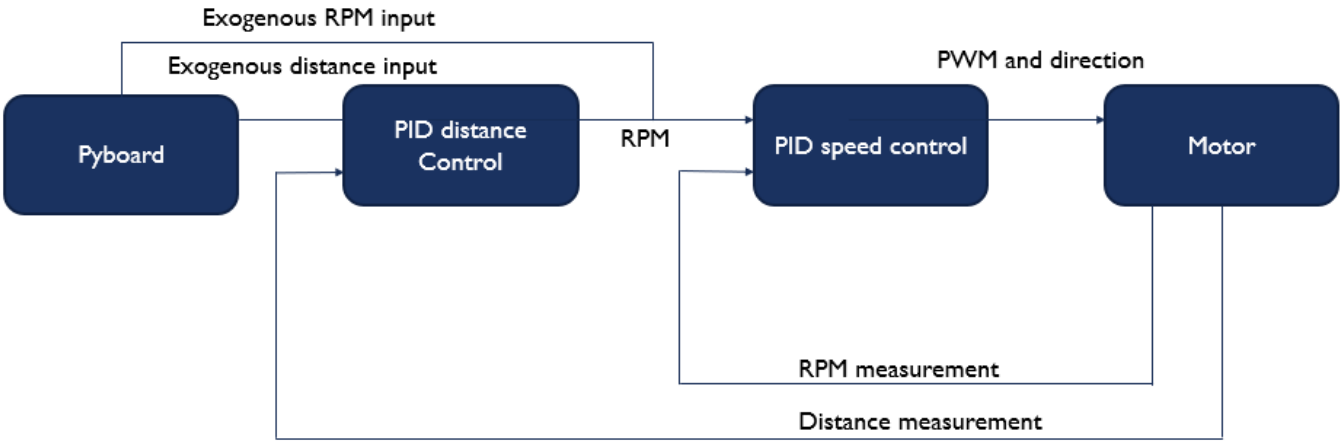


Figure 3: The 3D file of the robot

As we can see, the pyboard is the brain of the control, and it can either give an input of rotations per minute for the inner loop of the diagram, or an input of distance for the outer loop. The first input controls directly the speed in which the wheels would turn. The former, having a distance as an input, would automatically generate a speed curve to be followed by the motor.

However, this cascade control needs to be carefully implemented, as we will see in the code, the sampling time and frequency of the inner loop needs to be faster than the outer loop, as to give the motor enough time to change its speed to the desired speed input.

3.2 Velocity control PID

The inner loop of the control is the one that controls the speed in rotations per minute of the wheels. As we can see from the diagram, this PID control receives an information of desired speed as well as the current RPM of the motor deciphered from the encoders. Having this error, we implement a PID control, that tries to make the settling time the smallest possible, while giving a small overshoot. We want a small settling time, as it will influence a lot on the efficiency of the outer loop.

3.3 Distance control cascade PID

The outer loop of the control is the one that receives an information of distance as well as the distance traveled to give a smooth speed curve for the inner loop. This is necessary to improve the precision of the robot, as well as to make its movements more smooth and not only rely on an on-off control. An example of the results we were receiving of a step response of 30 tics of encoder(30 degrees of rotation of the wheel).

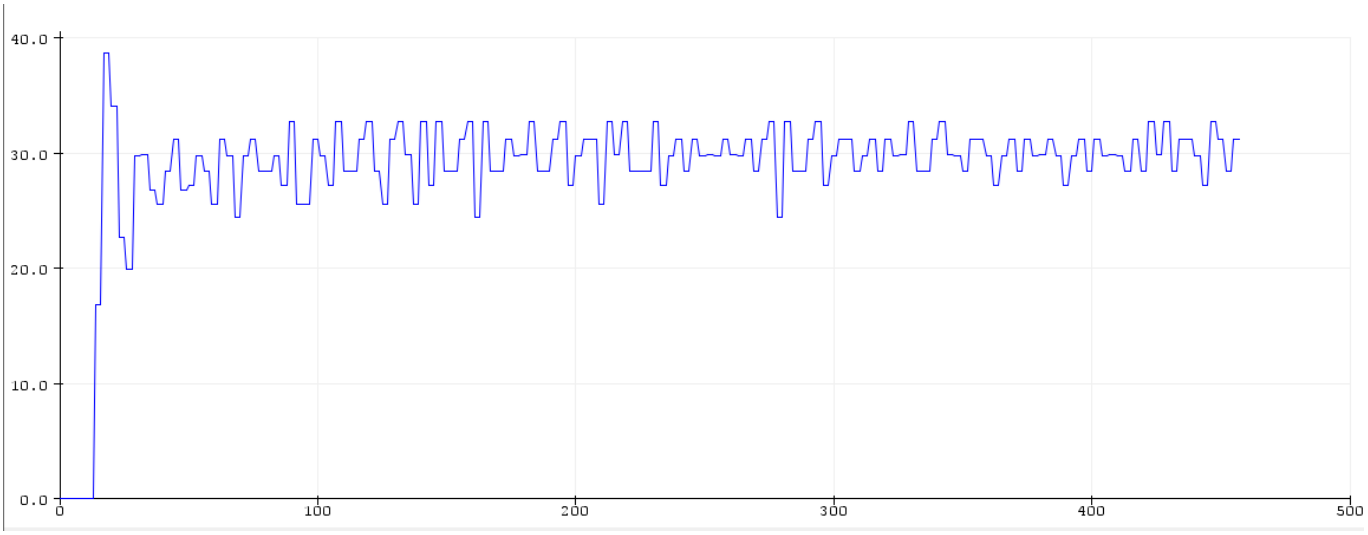


Figure 4: Step response of 30 tics of the encoder

As we can see, in less than 30 milliseconds the desired distance was already reached, while having a minimal overshoot(this was generated with the motor with no-load, knowing that this overshoot doesn't happen once motor is given a load of its weight).

3.4 Tuning the PID constants

To tune the different PID parameters of the inner and outer loop of the control we could chose two very different manners. One is by modelling the robot, and its motors response to a voltage, and given the desired values of settling time and overshoot, design the PID. Another, and the one we chose, was to experimentally do it based on the comportment of the motor. By changing the Kp Kd and Ki given the step response of the motor.

Our step-by-step actions were the following:

- Set all constants to zero
- Increase the Kp until you start having oscillations
- Increase the Kd until you damp the oscillations(achieving a critically damped state)
- Increase the Ki until it brings the motor to the correct setpoint with the desired step response

Given that the controll of the outer loop is harder to achieve, you should try to look at the output distance and try to diminish or increase the strength of the control(Kp), the damping effect and stability(Kd) and the achievability of the setpoint(Ki), achieving a smooth and controllable motor.

3.5 Arduino(C++) code

3.5.1 Introduction

As a first approach to resolve the given control constraints we chose to program in the Arduino IDE. Given the possibility to implement the reading of the encoders using external interruptions and the easy to implement PID libraries.

Our code was programmed in a modular and object oriented way, as to give an easier time implementing new functionalities that we wished to do further down the line, such as pre-programmed movements or even the implementation of computer vision to influence the movement of the robot.

We encountered some problems with the quantity of external interruption pins as they would not be enough to implement the encoders as well as some switches for contact detection. So the solution implemented was to use a multiplexer with the switches, however, this solution did not prove to be as effective, and was one of the reasons we gave up on the Arduino platform for the MicroPython one.

3.5.2 In detail

We created 3 main classes: PID, motor and robot. The PID was a general model implemented 2 times on our cascade control, with some notable information on the importance of the sampling time, as different controls require different sampling times and therefore computational powers. The motor class was the implementation of the sensor of the encoder included on the motor, and the robot class would give the information for the functioning of both motors, implementing the different desired PIDs.

After making the basic functionality of the cascade control as well as the implementation of the side switches and their preprogrammed movements, we started implementing the communication with the actual brain of the robot, which would be an ESP32 model that would give the instructions via an I2C communication channel. This was done by basically decoding the messages received by the arduino and sending out a received message to the ESP32.

3.5.3 Problems

As already commented, arduino is not a good platform to implement multi-threading with different timed interruptions, that we desired to do to expand our possibilities with the robot, and so we quickly changed our base layer controller to a pyboard, and implemented the code on the already existent code. Another reason to change microcontrolers was the limited pin out of the arduino, that could not attend all of our desires of pins nor the quantity of external interruption pins desired.

3.6 Pyboard(Python) code

3.6.1 Introduction

We didn't start with a blank project, as the last group already had a basic control code for the same romi kit. However they implemented a simple ON/OFF control that presented some glitchy and imperfect movements, therefore we decided to transport the learned knowledge from our arduino code to the already existent MicroPython code. This proved to be the most effective way of work, as with only the creation of the PID class we could start testing the robot.

3.6.2 In detail

After creating the PID class, we implemented it following the same steps as we did in the arduino IDE. Meaning that we started with the inner loop and its functionalities and then moved on to the larger one, always trying to keep the values of speed and distance as real and precise as possible, as to make it faithful to reality. Several functions inside the romi class were changed to make this happen, as well as many more were created.

Now that we had a more powerful microprocessor, we could use the different timed interrupt as a way to execute the inner and outer loop measurements and outputs. The excess external interruption pins were used to implement the environment detection with the switches, meaning that when the robot bumps and touches an object or well, it will implement a maneuver to avoid it and return to its basics paths.

3.6.3 Advantages

Apart from the faster processing time, and the more user friendly programming language, MicroPython offers more options for us, that made this project possible. First of all the pin out is more adequate to our needs, and the easy to implement timed interrupts, that were intuitive to implement, made it possible to extend the functionalities of the robot.

3.7 I2C protocol

Currently the I2C protocol allows the master module to control the velocity of the robot and to perform pre-programmed trajectories. We use the flowing protocol :

First byte Type of control (direct control by speed, or by distance, or pre-programmed movement)

- 0 velocity control
- 1 distance control (in mm)
- 2 pe-programed movement (Work in progress)

Followings bytes, depend on the control type Control data (speed, distance or preprogramed movement arguments),

- Desired speed left(2 bytes) and desired speed right (2 bytes)
- Desired speed left(2 bytes) and desired speed right (2 bytes)
- Select Movement (Work in progress)

It is easy to further extend this protocol if needed.

4 Lidar module

4.1 The lidar

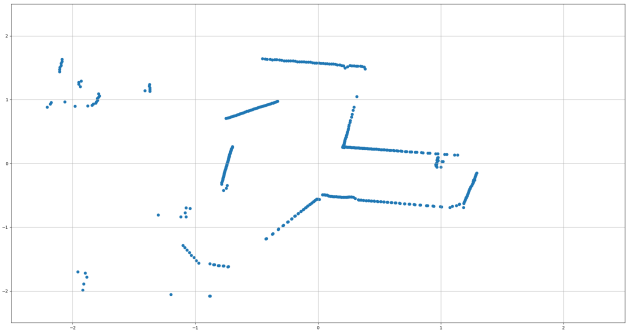
We equipped the robot with the RPLIDAR A1 from Slamtec, it allows a 360° 2D laser scan around the robot. The Lidar is placed on top of the robot so that it can pick up real-time information about the environment. It has a rotation frequency of 5.5Hz (which can be increased to 10Hz), which is sufficient for our robot. Its maximum distance measurement is 12m.

Figure 5: RPLIDAR A1 from Slamtec

Using the data provided by the lidar it is possible to obtain a 2D point cloud describing the environment of the robot.

The lidar provides the data over UART according to a protocul detailed in the documentation is available here : www.slamtec.com/en/Support.

Example of the available data from the lidar :



(a) Point cloud from the lidar

(b) The environnement

4.2 The micro-controller

To connect the lidar to the I2C bus of the robot, we used a pyboard. This choice was based on the fact that it

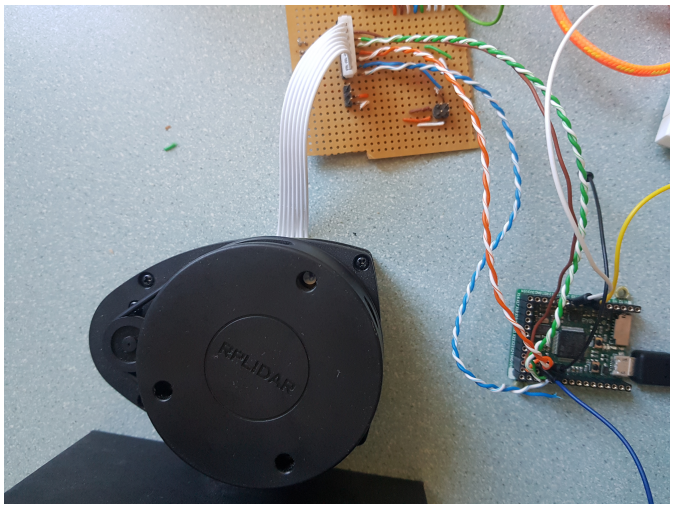


Figure 7: Pyboard connected to the lidar

would be easier for other students to work on this module. Currently the pyboard simply directly outputs lidar data over the I2C bus when the master asks for it, but one could easily for example do some signal processing on the raw lidar data and the send the processed data over the I2C bus.
The lidar continuously scans and produces new data, each piece of data that is sent over UART is the data for a single point. The pyboard, stores the data in a fixed size array (currently it stores the last 524 points scanned) and loops around the array refreshing the old data when a new point is sent by the lidar.

4.3 I2C protocol

Currently, the I2C protocol implemented allows two things to be done. The master can ask for the array of the pyboard storing the data, and the master can change the rotation speed of the lidar.

Get lidar data To get the lidar data, the master firsts sends a single byte that is equal to 1 (= \x01) over the I2C bus. The pyboard with respond by sending the data to the master. Each point is coded on 4 bytes, 2 bytes for the heading and 2 bytes for the distance.

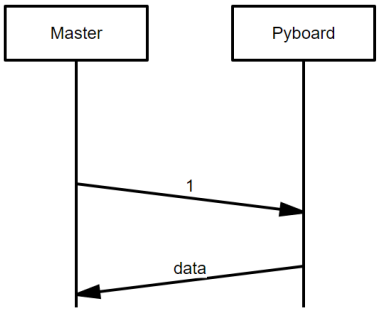


Figure 8: Get lidar data sequence chart

Set lidar rotation speed To set the lidar rotation speed, the master firsts sends a single byte that is equal to 2 (= \x02) over the I2C bus. The pyboard will then wait untils it receives the duty cycle from the master. The duty cycle sent by the master should be between 0 and 100, 100 being the maximum rotation speed for the lidar. Currently, the duty cycle is encode on 2 bytes. 2 bytes is exaggerated as a single byte is sufficient to store a number between 0 and 100, the second byte can be used for later use to encode something extra if needed.

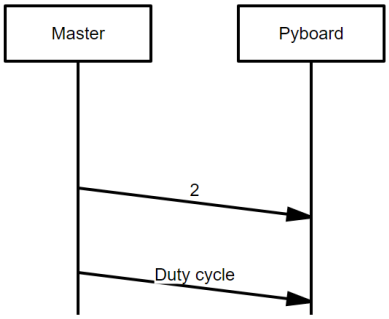


Figure 9: Set lidar speed sequence chart

4.4 Ways to improve the system

The mains issues lies in the fact that the lidar continuously outputs the data at a very fast rate. Furthermore, the current Micropython build for the pyboard does not support interrupts for UART. During the I2C communication, the pyboard remains in a timeout for a certain duration but the lidar continues to output data as a consequence a very big RX buffer is needed. It may happen that the buffer overflows, it then becomes almost impossible to further decode the data that is being sent by the lidar. Currently, when such a thing happens the pyboard will data that it is unable to decode several successive points. It will then restart the lidar. Since, in the current state the pyboard does do any computations the big RX buffer works fine as a solution. But in the future, a better solution may be needed. One can maybe implement a good algorithm using threads or the asyncio library. This probleme may also be solved by writing a C script that is then embedded is the Micropyton firmware. Other solutions can also be explored.

5 The master module

The micro-controller used as a master is an ESP-WROOM-32. The wifi capabilities of this micro-controller will allow use to remotely control the robot.

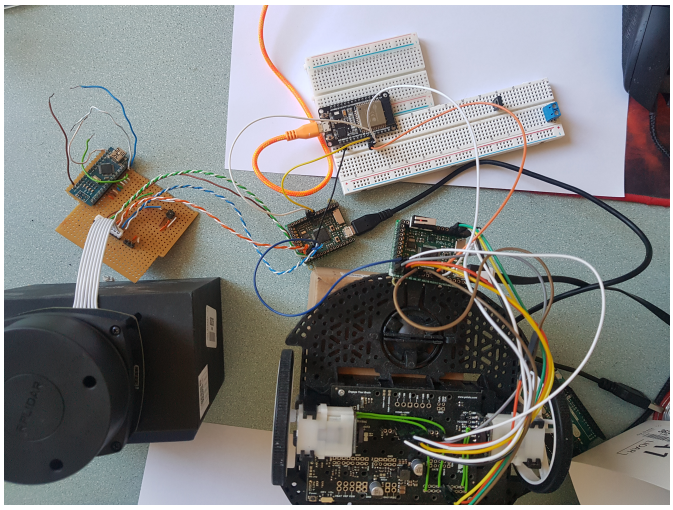


Figure 10: ESP32(on the breadboard) connected to the other modules

5.1 Web REPL server using the espressif-IDF SDK

We we started working on our project, we planned on programming the ESP32 using the espressif-idf sdk. It is the sdk that is provided by the constructor of the micro-controller. Satisfactory results were achieved using this sdk, a web server that communicated with the Locomotion module over I2C. However the development was slow, the sdk takes some getting used to, and it is based on a more low level approach. As a consequence the sdk is very powerful, one can optimize everything and full control over the hardware is available, but it comes at the cost that a lot of learning is necessary before getting anything done. Considering the fact that the robot is supposed to be passed on to another group, we decided to abandon the espressif-IDF sdk and use the Micropython firmware on the ESP32. The functioning Web REPL Server was thus abandoned, but the source code remains available on the github linked at the end of the report.

5.2 Current state : simple TCP server using micropython

In its current state the ESP32 works as a TCP server and is accepting connections port 5006. Every device on the same network can thus connect to the robot. The server will accepts commands to move the robot and it can send the lidar data to a connceted device. A python program was coded to connect to the robot. The program uses the Pygame library to draw the live lidar reading, the program also reads the keyboard inputs to control the robot. Further details on the TCP server can be found on the github off the project linked at the end of the report.

5.3 Ways to improve the system

HTTP server In its current state the ESP32 does provide a web-server. A web server could be very useful as it would increase the versatility of the robot. One could for example easily use a phone to control the robot by just using the web browser.

I2C protocol The I2C communication with the other modules works just fine. However the modularity could be improved. To add another I2C module, one must adapt the I2C_master python class. It would be great to make a more adaptable class that would for example read a json file in which all the protocols of the connected modules are described.

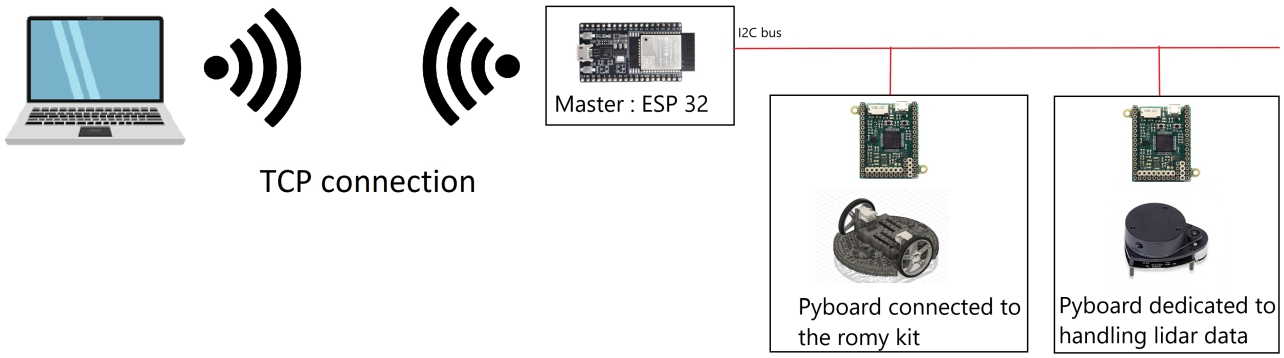


Figure 11: ESP32(on the breadboard) connected to the other modules

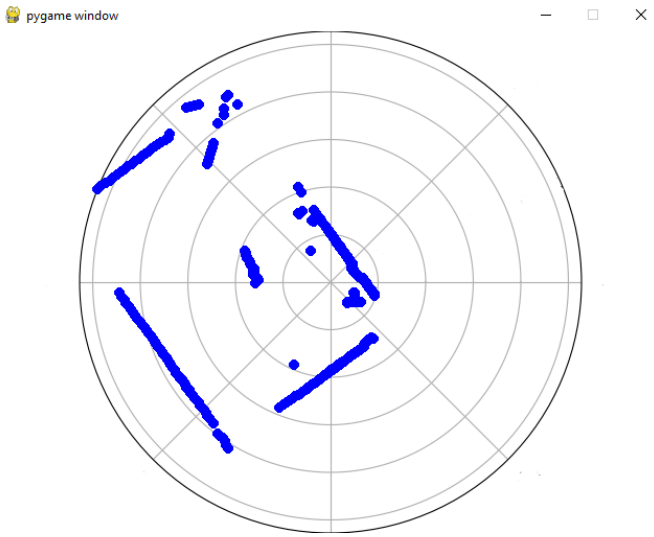


Figure 12: Live lidar readings from the robot

6 3D modeling and laser cutting

6.1 Robot modeling

Since romi is a very popular and well-known platform, we found several 3D models online that helped us make dimensions and create more 3D parts for the robot. As follows is a fusion360 model of the romi platform completely build, with real world measurements.

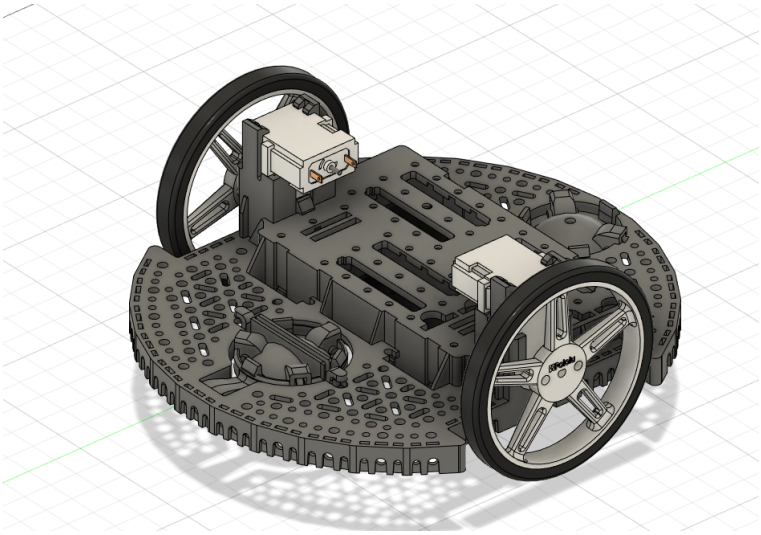
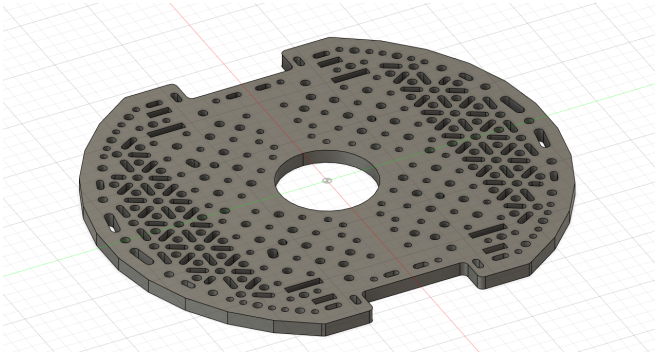


Figure 13: The 3D file of the robot

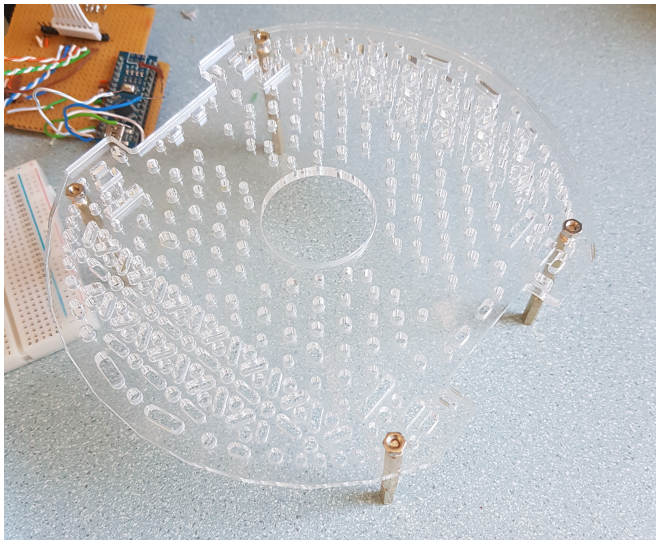
6.2 Second stage modeling

Using the measurements and dimensions of the romi 3D model, we were able to create a second stage. This stage is comprised of several holes to accept several different modules, including but not exclusive to the Lidar. While having a main hole for the cables to connect to the main processors of our robot.

With the modelling of the second stage done, we contacted several people to help us laser cut it into a real plexiglass piece, that could be attached and improve the sensor's compatibility with our robot.



(a) The 3D file of the second stage of the robot



(b) Laser cut in plexiglass

We used the second stage to mount the lidar to the robot.

Figure 15: Assembled robot

7 Conclusion

The prototype built is great minimal viable product. The robot can explore a 2D plane with ease and can be controlled remotely. However, we did not have the time to implement some aspects, such as camera vision count the number of people in a given room for example and making the robot autonomous. The limited time is a consequence of a 180 degree turn we made when switching all our microcontrollers to Micropython. This decision was made mid-semester, and thus very limited time was available to start over. That being is said, we learned a lot during the first half of the project. Now that everything is well set-up, the next group to work on this project will be able to easily pick up where we left and further develop the robot to achieve the desired goals.

7.0.1 Summary of ways to improve the robot

- Lidar module : implement signal processing
- Lidar module : implement signal processing
- Master module : implement a HTTP server
- Master module : more adaptable I2C algorithm (json file for I2C protocols)
- Mapping of the environment
- Add a camera module to the robot
- Add other modules that might be useful (module to follow a black line for example)

Link to the github:
https://github.com/kapeps/Robot-explorateur-.