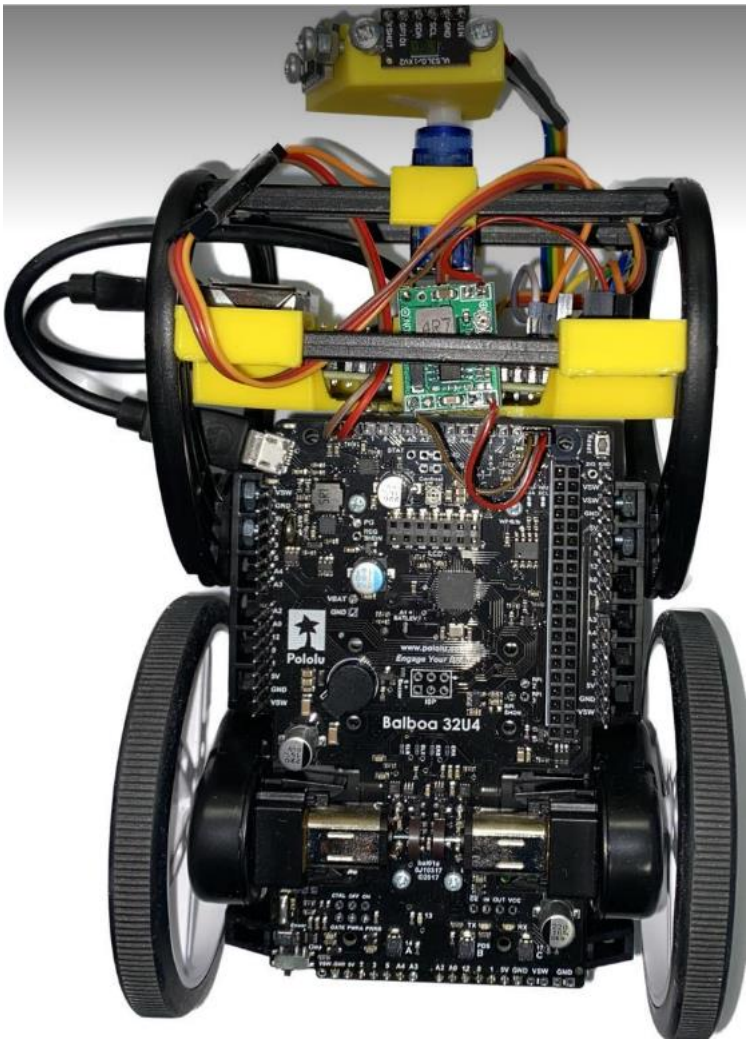
5/15/2023

Advanced control 2:

Slam robot



Florian Desneux 195126
Mohamed Boutahar 195191

Contents

1. Project introduction

As part of the Advanced control 2 course, we did a job in which we were able to choose our own subject with the mobile balboa robot. We opted for subject number 6 "SLAM ROBOT". This subject consists in creating a map of an unknown environment thanks to the remote control of our laptop. The purpose of this documentation is to understand how to reproduce our project and then improve it for the years to come. If you are interested in the details and source files for this project, please follow this link to the project github : https://github.com/DesneuxFlorian/Balboa_team_2

2. Objectives

In order to carry out this project as well as possible, we have set ourselves several objectives which are as follows :

- Laser sensor with servo-motor
- Data collected and sended to laptop with mqtt
- Laptop controls the robot with user input
- Basic odometry data computed and sended to the laptop
- Implement a SLAM algorithm

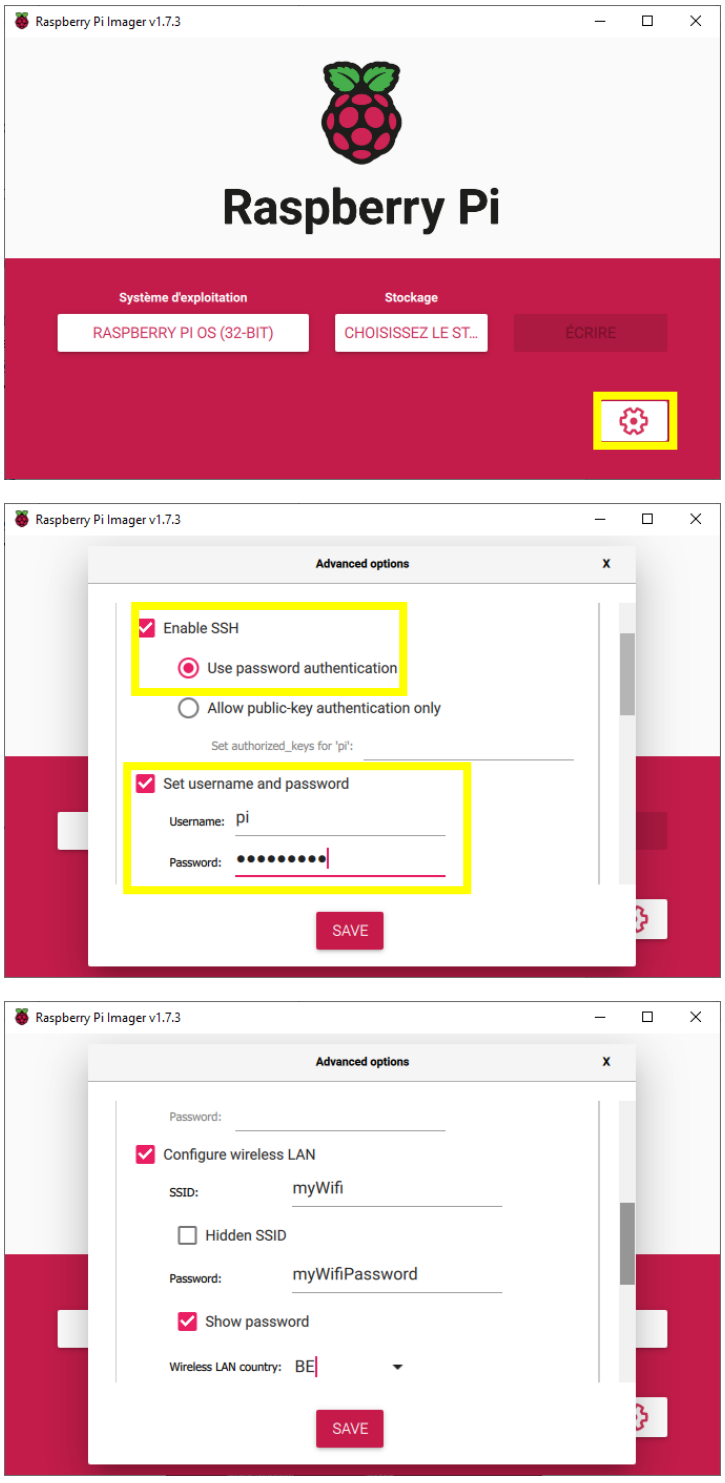
3. Raspberry pi Setup

For the project we are using a raspberry pi as the main computer. It will be in charge of the communication between the Balboa and the Laptop. Additionally, the Raspberry will be operating the lidar scanner. We used this approach to try to run a SLAM algorithm on the raspberry.

3.1 OS installation

The first step is to install a Linux OS on the raspberry pi. For this we used this tutorial: How to Install Raspberry Pi OS on Your Raspberry Pi. Because we will want to use remote access to the Raspberry using SSH, we need to add the following step during the installation:

1. Go to additional Settings.
2. Be sure to enable SSH.
3. Put a username and password that you will remember.
4. You can also set the hostname you want, it will become handy later on.
5. Enable and configure your wi-fi settings in the W-lan tab.



Save and write the OS to your SD card. You should then be able to boot the raspberry pi.

3.2 VNC setup

It is now time to connect via SSH to our raspberry pi. Once that is working, we will install a VNC viewer to access a remote desktop. To access via SSH, we need to know the IP address of the raspberry pi on the current network. To do this, we need to boot the raspberry with the SD card we just configured. When it is done booting (could take a minute or two), we need to

find the IP address that has been assigned to the raspberry pi. There are several ways to find the IP address. This tutorial will show different ways to do that: How to Find a Raspberry Pi on a Network. We used the third approach with a windows terminal and using ping command. The hostname of the raspberry pi is the hostname you have configured earlier. You should get an IP address of the type **192.XXX.X.XX.** If you get an address with another format, use this one, but you will have to find the IP address once you are connected because VNC viewer only supports the first format.

A second way of finding the IP address remotely is to connect to your router's login page. Most internet providers allow you to oversee all the connected devices on your network and you should be able to find the IP address of every connected device, including your Raspberry pi. If you cannot find your raspberry pi, it is possible it isn't connecting properly to your network. Follow the troubleshooting steps of the tutorial to try to find your problem.

We can now test to connect via SSH to the raspberry pi before moving on to the next step. On windows open a command terminal and use the command: **ssh pi@192.XXX.X.XX** replacing "192.XXX.X.XX" with your raspberry IP address. Enter your password and you should be connected to the raspberry. If you got a IP address of the other type, we need to get the real IP address of the raspberry pi for the VNC viewer. Use the command **sudo hostname -I** in the command terminal to find it.

You can now install Real VNC and connect to the pi using it's IP address. You should get access to the raspberry pi desktop.

3.3 MQTT Setup

To communicate with the laptop, our code will use MQTT. So we need to install mosquito on the raspberry pi. We have followed this tutorial enabling remote access without authentication: Install Mosquitto MQTT Broker on Raspberry Pi. We also need to install the python package to use mosquito with python. Open a terminal and run **pip3 install paho-mqtt**.

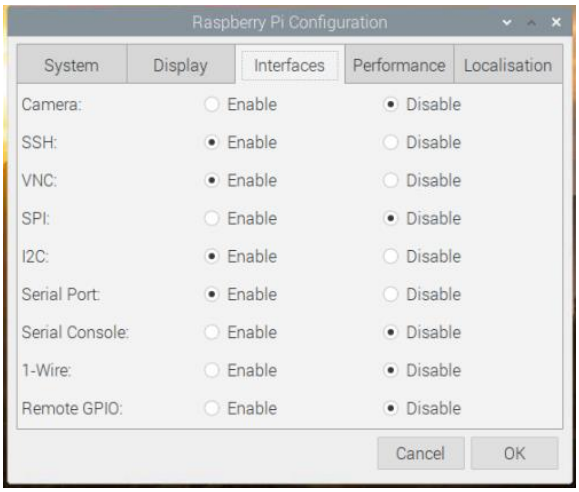
3.4 Additional dependencies Setup

The code is using some other packages. Here is a list of the packages you should install on python3:

- numpy
- serial
- smbus2
- vl53l1x
- json

Install all of these dependencies using PIP3.

Finally you should also go to the configuration tab of the pi and enable i2c, Serial port and VNC. Don't forget to reboot the pi after changing the configuration (command **sudo reboot**).



3.5 Hotspot tools V2

We also want to install hotspot tools v2 on our raspberry pi. This way we will be able to control the robot event when there is no available wifi network. The raspberry will boot in a hotspot mode and we then simply need to connect to its local network. This will also enable us to setup new wifi connections without needing to restart the other steps. To install hotspot tools v2 we simply followed the steps for installing from the project Github page[1]. There is also a video available for a more in depth tutorial.

4. Communication between balboa robot and raspberry PI

Here, we will communicate in series between the balboa and the Raspberry PI thanks to a USB cable connecting the balboa robot to one of the USB ports of the Raspberry Pi.

4.1 Serial communication

The principle of serial communication is to sequentially send one bit of data at a time from one location to another. For example, using serial data, you can send data from your balboa to a Raspberry Pi or from the Raspberry Pi to the balboa.

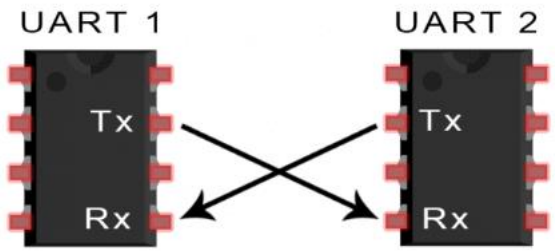
After discussing with our teacher, we chose to communicate with a USB cable because it is one of the most common ways to transmit data in serial communications hence the name of universal serial bus. So with balboa, we can easily send and receive data via a USB cable thanks to balboa's built-in serial library.

4.2 UART communication

Once the data has arrived on our balboa, it will be stored in what we call "serial reception buffer".

[1] https://github.com/km4ack/hotspot-tools2

When bits of data start flowing into the computer, a hardware device in the balboa used for serial communication between two devices called a universal asynchronous receiver/transmitter (UART) assembles each of the 8 bits into a byte. The UART then stores these bytes (up to 64 bytes, in fact) in the serial receive buffer.



One wire is for transmitting data (called the TX pin) and the other is for receiving data (called the RX pin). We can only connect two UART devices together.

Once the data is in the serial receive buffer, we can use Serial.read() to read what is in the receive buffer. However, it's important to understand that it reads in a very specific way. It'll read out the first available byte from the serial receive buffer and then remove that byte from the buffer.

In order to simplify things, we will impose a rather basic protocol :

- New messages will be read as soon as they arrive
- Messages must not exceed 12 bytes
- Each message will end with a newline character '\n' which we will call our end character

5. Communication between Raspberry pi and laptop

We are using 3 different topics for the communication.

1. raspberry/scan
2. raspberry/odom
3. raspberry/setpoint

The first and second topics are published on with the raspberry pi, and the third topic is used to control the balboa from the laptop. It is published on when you press a key controlling the robot. We are using keys "i,j,k,l" to move the robot around, key "t" to start and stop the scanning and finally you can use key "s" to save the data coming from the robot.

6. Electrical circuit

Here we will illustrate in detail the wiring of our project

6.1 Power supply of the raspberry PI

Regarding the power supply of the raspberry PI, we decided to power it directly with the balboa. The problem is that our raspberry PI has to be powered with 5 V while our balboa provides a battery voltage. To solve this problem, we decided to use a buck converter to reduce this voltage to 5 V. Here is the buck we decided to use:

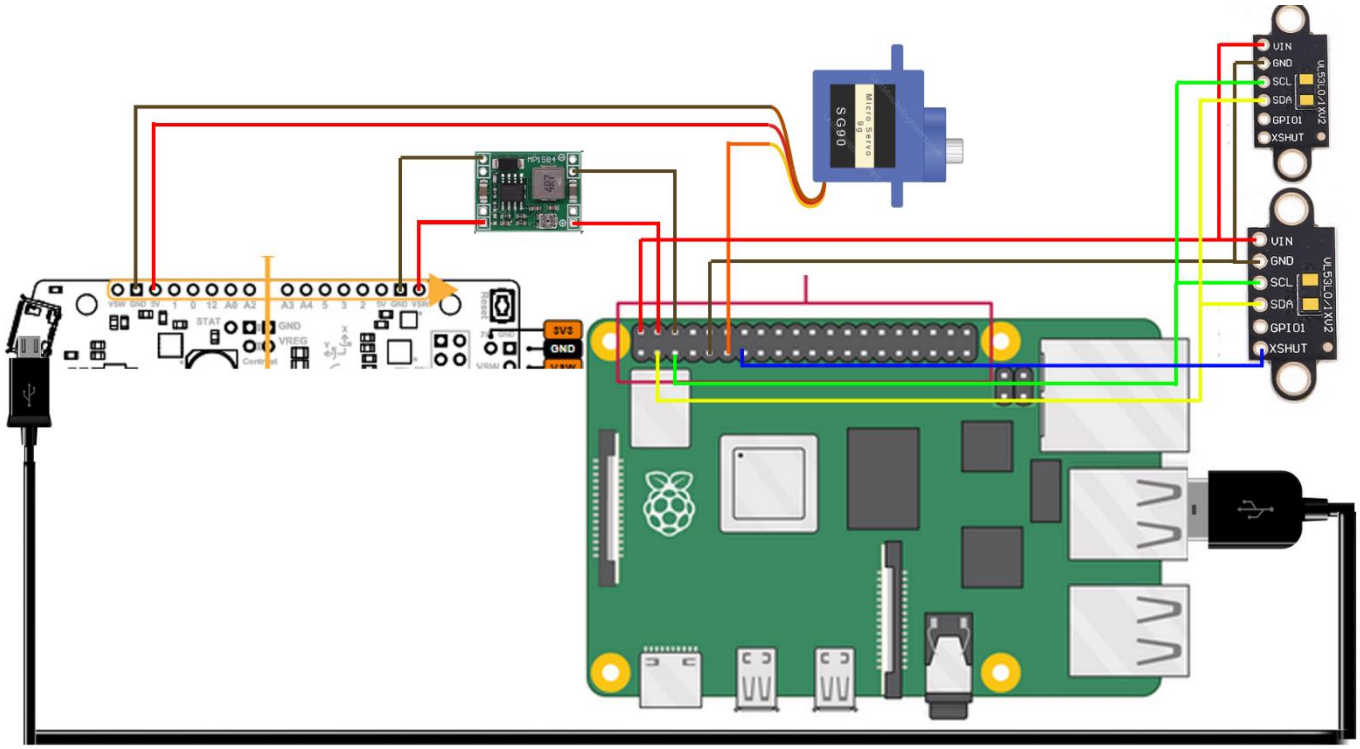
By using the buck converter, we are sure that there is enough power for the raspberry pi. Normally the balboa already has a 5v bus that we could use, but it was not powerful enough and that's why we made the choice to use a buck converter.

6.2 Servomotor

In order to have a maximum of data to better map the environment, we decided to mount our sensors on a servomotor which will turn it at a certain angle.

6.3 Wiring diagrams

Here is a complete assembly of our balboa robot with the different elements such as the buck, the servomotor, the sensors and the raspberry PI



7. Software

The software is composed of three big parts. The first part is the C++ code running on the Balboa robot itself. Basically it is in charge of the control algorithm as well as computing the dead reckoning odometry for the robot. The second part of code is the python code running directly on the raspberry pi. This part of the code is responsible for controlling the lidar scanner, and sending the data to the laptop. It is also used as middleman for controlling the balboa with the laptop. The raspberry pi gets control commands from the laptop trough MQTT, and then sends them to the balboa trough serial. The last part of code is also a python code that is running on the laptop. Its purpose is to retrieve keyboard commands and send them to the raspberry and to receive the scanned data from the raspberry pi and to display them on the screen. It is also in this code that we convert the data from polar to cartesian.

7.1 Balboa code

The control is performed by a state-space algorithm. Additionally we have added a few functions:

- ComputeOdom()
- readData()
- treatCommand()
- sendOdometry()

The ComputeOdom function is running every 10ms to evaluate the balboa robot position in a cartesian coordinate system. It is implemented following this article found online[2]. The second and third functions are running at a much higher sampling rate than the rest of the control algorithm. This ensure that a fast serial communication between the raspberry and the balboa as well as a fast handling of the commands. The last function is simply sending the odometry data trough serial when the robot reads a “O” in the serial buffer.

7.2 raspberry pi code

The code running on the raspberry pi is composed of 2 files. The Lidar file, that contains a Lidar class. This class is in charge of everything related to the laser scanners. It will connect to the two lidars, then it will start those lidars and finally it has a method to run a scan with the lidars. This class also controls the servo motor on which the two vl53l1x are mounted. The method performing the scan is working this way:

- first it moves the servo to a position
- then it asks the odometry to the Arduino
- get the distance from the two sensors
- appends the angles and distances to a list and performs everything for a new position
- finally it returns the lists when the sweep is done

[2] Wheel Odometry Model for Differential Drive Robotics

Initially we were not getting the odometry for every point in the scan but only for the last one. It had the advantage to speed up the control of the balboa because it had less communication. But because the slow nature of our scan, it is a lot better to get the odometry for every point so that we can correct the list of angle for every point individually.

The second file is the main file. This file is the one that will be started and running continuously on the raspberry pi. It has a functions that handle the MQTT communication, as well as a main loop running where it will scan around using the lidar object created at the beginning of the code. Before running this file it is important to run the command **"sudo pigpiod"** in a command terminal so that the lidar class can correctly run. This class is using hardware pwm to control the servo motor. This is so to remove the jitter that we have if we use the conventional software generated pwm.

7.3 laptop code

Finally the code running on your laptop is composed of 3 files. The roboviz.py file is a file from the roboviz library written by Simond Levy[3]. It has been slightly modified so for better result it is advised to use the file from this project github page. The lidar_to_grid_map.py file is a python file written by Erno Horvath[4]. This file has also been modified, thus the file from our github should also be used to ensure that the code will run. Before running those files you should also check that you have installed the required dependencies on your laptop. For the laptop the dependencies are:

- numpy
- matplotlib
- json
- paho mqtt
- keyboard

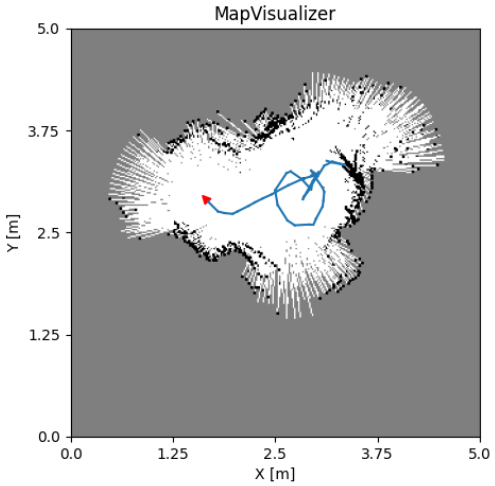
8. Mechanical assembly

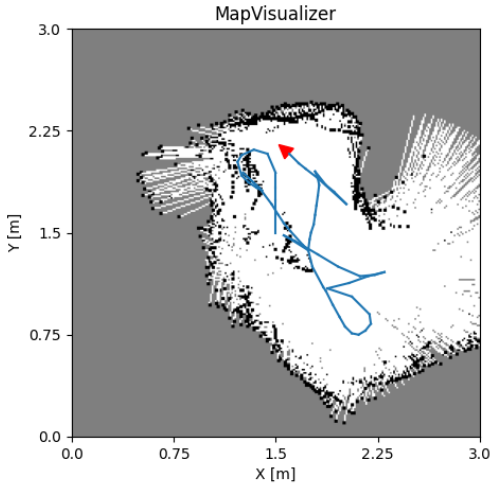
Two 3D printed parts are used to mount the raspberry pi and the servo motor on top of the balboa robot. Those 3d prints simply clip on the already existing mounts. There is also a small 3d printed support to mount the two VL53L1X on the 8g servo motor.

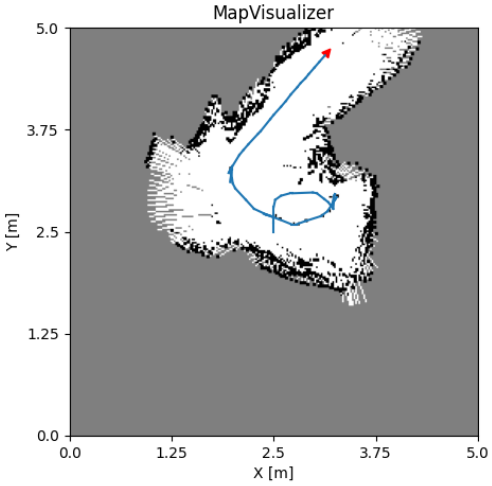
9. Results and improvements ideas

The robot is performing quiet well. When it is not scanning around it is easy to drive it around to where we want it to go. But because our scanner is slow we are controlling it with slow values. To start scanning we simply push on "t" on the keyboard. As soon as it starts scanning, there is a noticeable loss of control from the pc. The balboa is less reactive to keyboard commands. This comes from the increased communication from the odometry data. As we drive it around we can slowly see a map displaying on our screen. But because we don't run a real SLAM algorithm, the map is quickly becoming bad as a result of the odometry drift. The images below show some maps that we could draw.

[3] https://github.com/simondlevy/PyRoboViz
[4] https://github.com/AtsushiSakai/PythonRobotics/tree/master/Mapping/lidar_to_grid_map







On the maps we can see that on some places data overlaps or that there are scans of the same feature but a different places.

To improve our robot there are several ideas we could try to implement.

The first idea we have is to change the scanner with a scanner with a much higher sampling rate. Something around 5 scans/s should be enough. This improvement would help with a few problems we had. Firstly we could remove the need to correct every point of the scan with the robot odometry. This would reduce the amount of serial communication and thus the control of the robot would be the same as when we are not scanning, so a lot better. Secondly with a faster and better scanner we could try to implement a real SLAM algorithm. This was impossible with the data from our scanner. This would mostly solve the drift problem from the odometry data. If we change the scanner we would be able to really use the robot to map its environment and even try to do some path planning. A good replacement scanner that could be used as a starting point is the openSimpleLidar project from iliasem[5]. This lidar could maybe be redesigned to be smaller and lighter to work on the balboa robot.

[5] https://github.com/iliasam/OpenSimpleLidar

A second idea to improve the robot is to use the magnetometer as a reference point for the balboa heading in the odometry. We tried to use it, but the balboa is known to have hard and soft iron problems, making the data from the magnetometer hard to use. Even when performing a 3d calibration as explained here it was only giving good result for some amount of time. When using the same calibration somewhere else or later at the same location the results were not good as initially. Maybe it is possible to perform a calibration and use the raspberry pi to compute the correction coefficient at every startup of the robot.

Finally we thought about using ROS on the raspberry pi. By using micro-ROS on the balboa we should be able to use ROS to control the balboa and so every ROS feature, including the mapping and path planning algorithms could be used to improve this project.

10. Conclusion

In conclusion, we have achieved all our objectives mentioned in point 2 except the last one which is the implementation of the slam algorithm. To achieve this one we should try to improve our robot with the ideas explained before.

11. References

Configure the raspberry

https://howchoo.com/pi/install-raspberry-pi-os

https://howchoo.com/g/ndy1zte2yjn/how-to-set-up-wifi-on-your-raspberry-pi-without-ethernet

https://all3dp.com/2/find-raspberry-pi-on-network/

Communication part

Serial.read and Arduino: Here's the advanced stuff you should know (programmingelectronics.com)

How to Set Up UART Communication on the Arduino - Circuit Basics

https://www.instructables.com/Installing-MQTT-BrokerMosquitto-on-Raspberry-Pi/

Other

https://github.com/km4ack/hotspot-tools2

https://forum.pololu.com/t/correcting-the-balboa-magnetometer/14315

https://github.com/iliasam/OpenSimpleLidar

https://gpiozero.readthedocs.io/en/stable/api_pins.html#changing-pin-factory

https://gpiozero.readthedocs.io/en/stable/api_output.html?highlight=Servo#gpiozero.Servo

https://github.com/AtsushiSakai/PythonRobotics/tree/master/Mapping

https://www.domo-blog.fr/comment-creer-un-serveur-mqtt-sur-le-raspberry-pi-avec-mosquitto/

https://www.youtube.com/watch?v=_fdwE4EznYo&t=706s

https://stackoverflow.com/questions/64642122/how-to-send-real-time-sensor-data-to-pc-from-raspberry-pi-zero

https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/

https://github.com/pimoroni/vl53l1x-python