Rapport E-Puck

Group: G71
Luca Dos Santos
SCIPER: 274731
Ruben Jungius
SCIPER: 289260

May 16, 2021

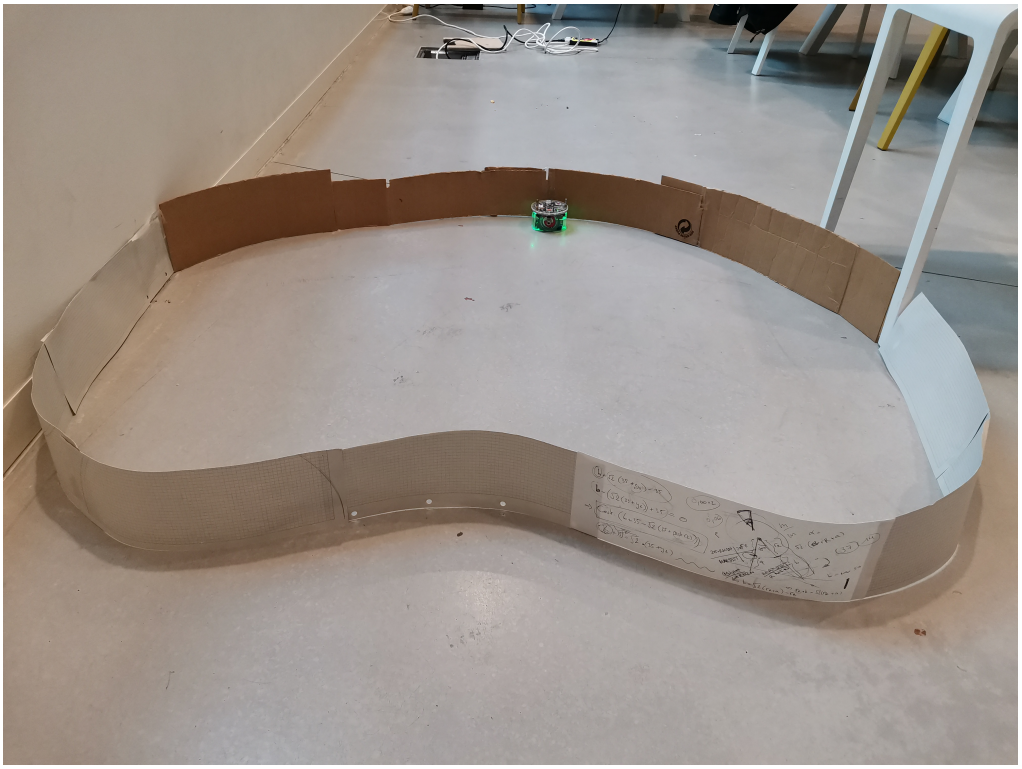

Figure 1: Picture of robot during pathing test

1 Introduction

The idea of the project is to have a wall-guided robot that can move from task to task in an imaginary production chain. After it is calibrated, an audio signal allows it to move until it reaches its next task, where it waits until another audio signal tells the robot to once again move forward.

2 Structure of the Project

2.1 Used Peripherals

The Peripherals used in this Project are the Infrared (or Proximity) sensors for Wall detection, the microphone to detect the audio signal as well as the 2 Motors for the movement of the Robot.

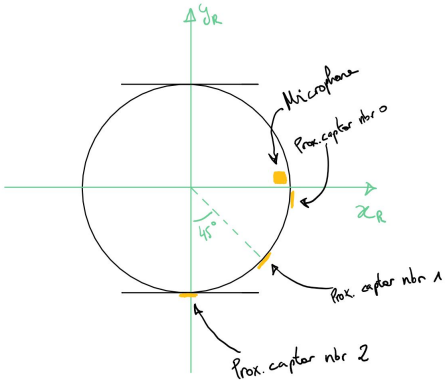

Figure 2: Robot with the used Sensors, the Front of the E-Puck is pointing towards the x_r-direction

2.2 Calibration

In the beginning of the Project we decided to use a mathematical function to convert the sensor data into distance of the robot to the wall. (Figure 3) shows the plotted data and the approximate polynomial needed. As x^4 can become very big with x hitting values over 3500, we decided to instead calibrate the robot at each launch with a table of values, that we then can use to linearly approximate the distance in between two measurement points. This allows us to use integers to convert the sensor values into distances. A further advantage of the calibration table is that the robot can work with

different wall-colors, for example a hard coded calibration table for a white wall would not work well on a black one.

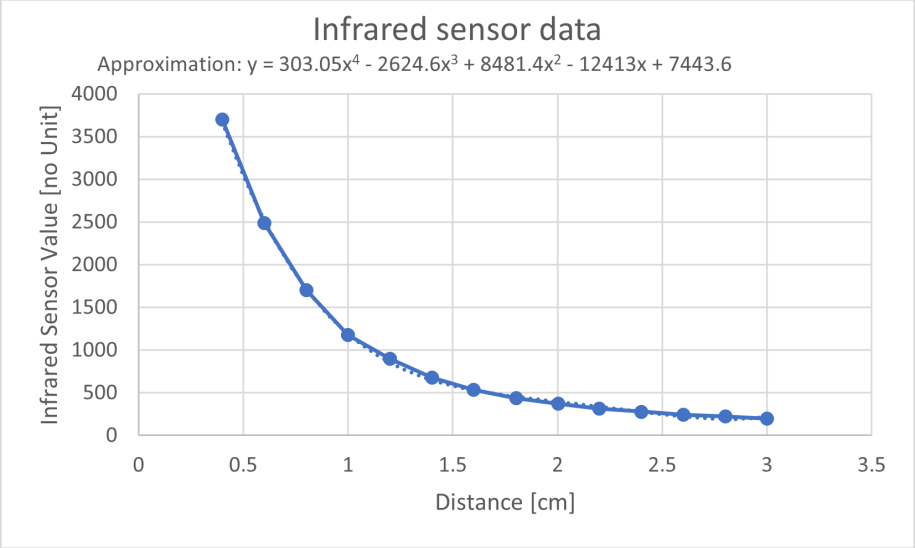

Figure 3: Excel plot of experimental sensor data. Excel approximated the curve with $y = 303.05x^4 - 2624.6x^3 + 8481.4x^2 - 12413^x + 7443.6$

The calibration table has 48 entries with 1 mm in between each entry, starting the measurement at 50 mm distance, and then approaches the wall step by step.

2.3 Launch via Audio control

After the E-puck is done calibrating, it waits for an audio signal in form of a predefined sequence. Only if this sequence is detected, the regulation and measurement threads are allowed run. For calculating the frequencies of the detected sounds, the code of the Audio TP was used and adapted to fit our purposes [1]. The sequences needed is $453Hz \triangleright 531Hz \triangleright 750Hz$. As a further condition, the sequence needs to be played within 2 seconds, or the E-puck will go back to listening for the first frequency. This has been implemented as the current 3 frequencies are all in the vocal range, resulting in lowering the risk of an accidental launch by random noises.

2.4 Measurements

To regulate our robot, two inputs are needed : an angle between the robot reference and the wall reference α and the distance between the robot and

the wall y (Figure 5). In order to find these parameters accurately, the robot needs to go straight forward during the process of measurements. Four measurements of the proximity captor '2'(Figure 2) are taken at a frequency of 80Hz and stored in a tab. A trigonometric calculation is then applied to find the angle α. To avoid any ambiguities with the reading of these values we use a mutex that locks the data during the writing.

2.5 Regulation

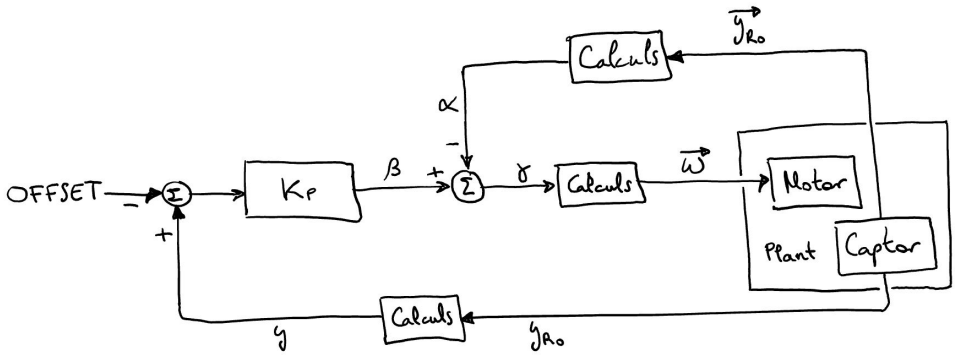

Figure 4: Scheme of the regulation steps

The goal of the regulator is to approach a wall and keeping the same OFFSET distance to it. To do that, we have to find an objective angle β that can be calculated for any given measurable distance to the wall y. This angle should allow the robot to fluently find its way to the wall. To accomplish that, we use a proportional gain K_P that multiplies the subtraction $y - OFFSET$ (y being the last of the measurements). With that it is easy to find γ, the correction angle $(\beta - \alpha)$. To correct this angle, we developed an algorithm that takes γ as an input and outputs the angular speed of each wheel that is needed to rotate γ in one period (0.1 sec). Once the rotation is done, a new cycle of measurements begins.

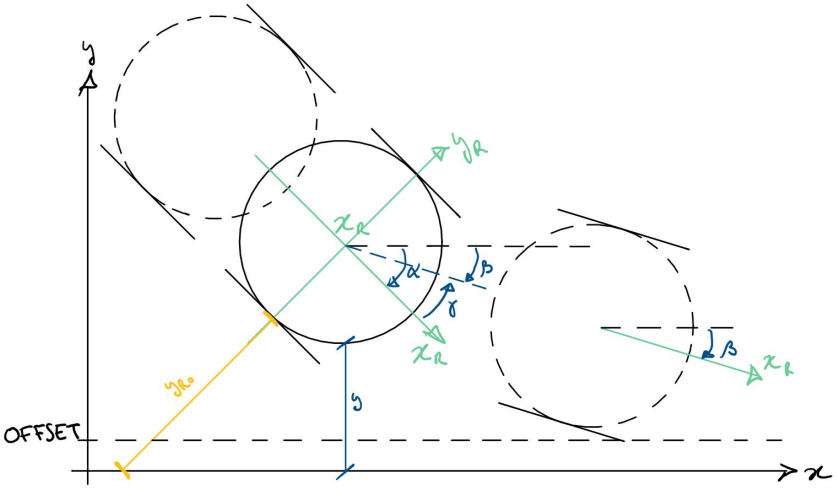

Figure 5: Dynamics of the robot. We can see both the measurement process (linear movement) and the regulation process (combination of a linear and a rotation movement).

We wrote a MATLAB script to understand the behaviour of our regulator and find the proportional gain K_P. We tried different input angles to see if the mathematical model was correctly reproducing the robot behaviour and we were pleasantly surprised by the results.

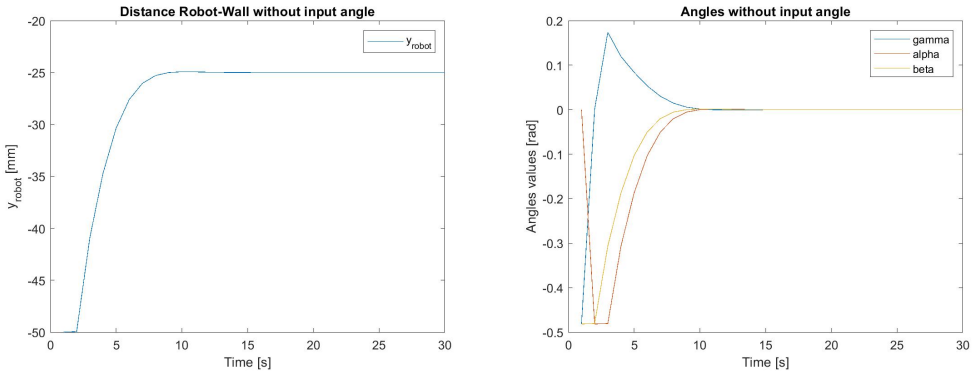

Figure 6: Behaviour of the robot when it first detects the wall with an angle α of 0°.

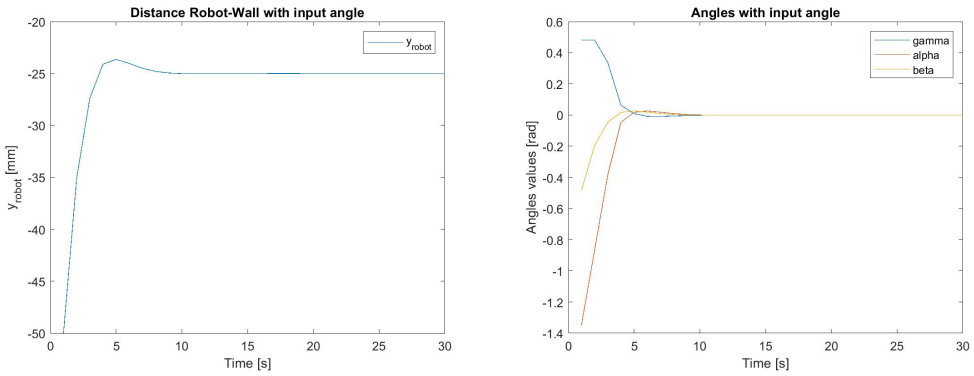

Figure 7: Behaviour of the robot when it first detects the wall with an angle α of $-77°$.

Both angles graphs show that α is trying to follow β with a delay that is slowly decreasing. Meanwhile, γ is as expected the difference between β and α.

Each iteration of the measurements part gives a new trajectory so that the regulator can easily follow all kinds of curves.

A protection has been implemented in the system to avoid a collision if a curve is to abrupt. We did that by checking if the values of the proximity captor '1' (Figure 2) (in the diagonal) are not to small. In this case the robot is stopped and only rotates during one period.

2.6 Threads

In total we have 7 threads running: Microphone, Main, Proximity-Sensors, Regulator, Measurements, Collision Protection as well as the System Idle thread. The creation of the calibration Table of the happens in the Main Thread before it enters its infinity loop and before the Measurements and Regulator thread are launched.

2.7 Flowchart

The Flowchart (Figure 8) shows the basic structure of the threads used to process the sensor data and move the robot.

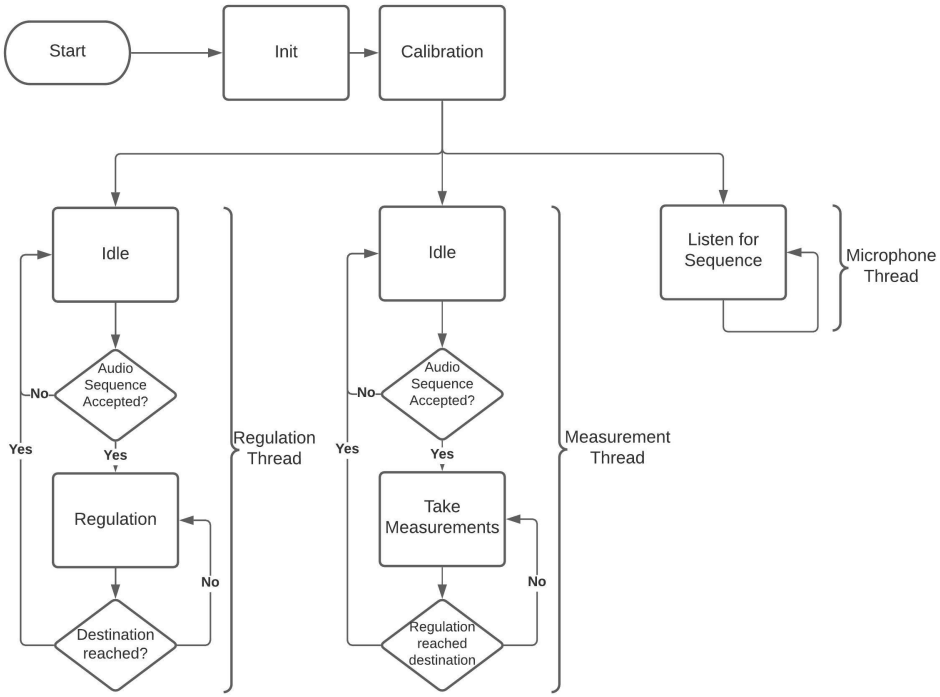

Figure 8: Flowchart of the three main threads and their basic buildup

2.8 Fixed Point

As problems with floating point operations came up, where the sum of two non-zero floats of similar value came up as zero, we looked into using fixed point operations in that case instead. The Fixed Point is defined as a 32 bit integer with a shift by 16 bits, resutling in interval range of $\pm 2^{14}$ with 16 decimals as mantisse.

The fixed point library was created with the help of [2], and the multiplication and division functions with [3].

3 Conclusion

The Project allowed us to learn about multi-threading using ChibiOS, its problems and advantages. It also allowed us to apply what we learnt last semester in the "Automatique et commande numerique" course, namely the implementation of a proportinal controller.

References

[1] Cours de Microinformatique: Practical Exercise 5: Noisy

[2] Phillip Johnston: Simple Fixed-Point Conversion in C, https://embeddedartistry.com/blog/2018/07/12/simple-fixed-point-conversion-in-c/

[3] Job Vranish: Simple Fixed-Point Math, https://spin.atomicobject.com/2012/03/15/simple-fixed-point-math/