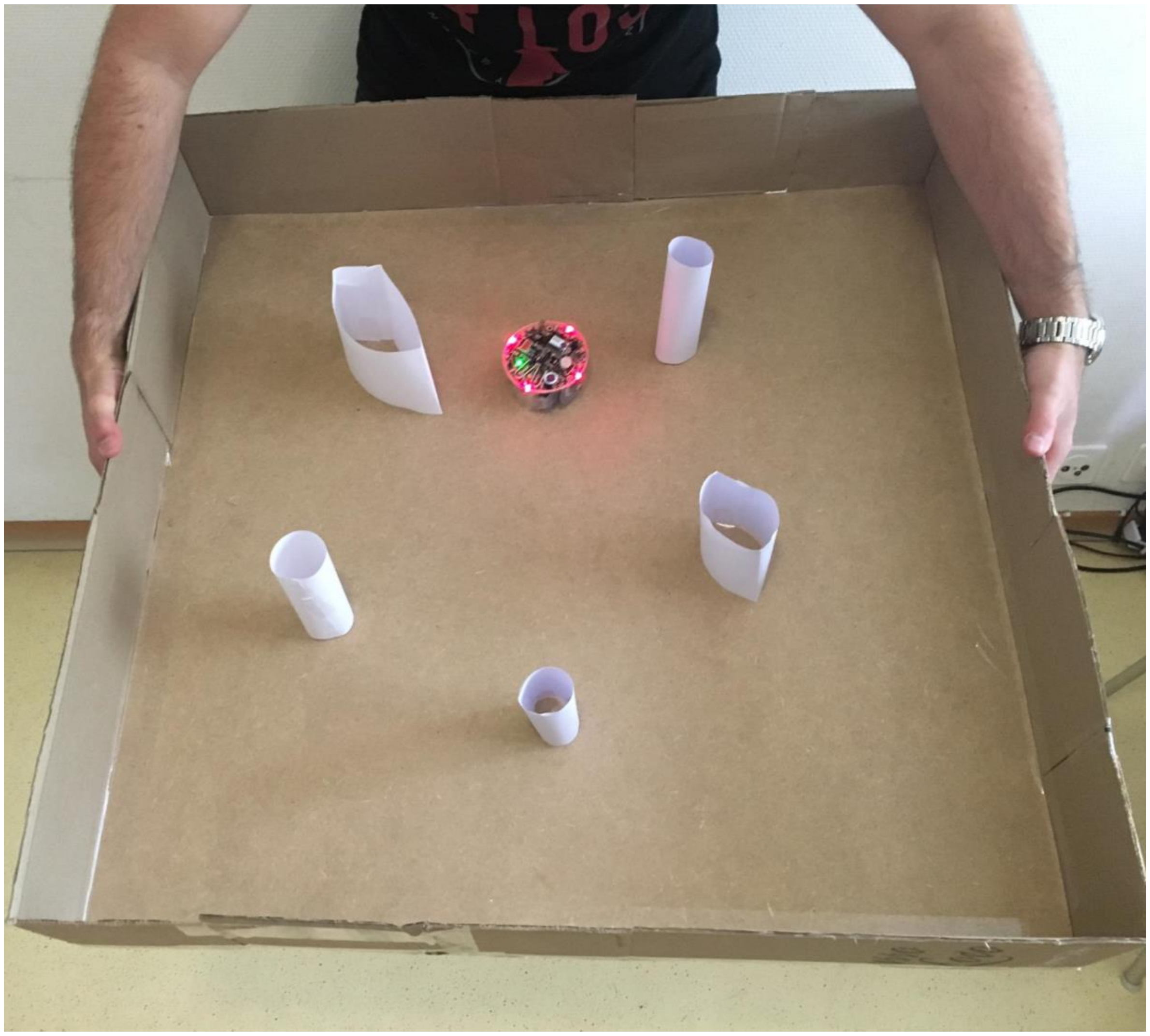
Slope controlled robot slalom



Mini project Group 31
Embedded Systems and Robotics - MT BA6

Loïc Von Deschwanden
Raphael Kohler

Loïc Von Deschwanden 283988
Raphael Kohler 284316

report group 31
miniproject

May 16, 2021

Table of Contents

1. Introduction and description of the project

The goal of this project is to program the e-puck2 such that it always moves along the path of steepest ascent. In addition, the robot constantly analyzes its environment and prevents collisions with any obstacle situated along its path.

The robot is placed in a large cardboard box that has no lid. The accelerometer of the IMU (Inertial Measurement Unit) is then used to determine the orientation of the robot by simply projecting the gravitational vector onto its reference system. The e-puck then rotates about the z-axis until its front faces along the steepest slope of the board and continues its forward movement. While doing so, ToF (Time-of-Flight) distance sensors are used to detect obstacles and a bypass movement is initiated in case the obstacle is too close. As an additional feature, one of five different preprogrammed speeds can be chosen using the selector.

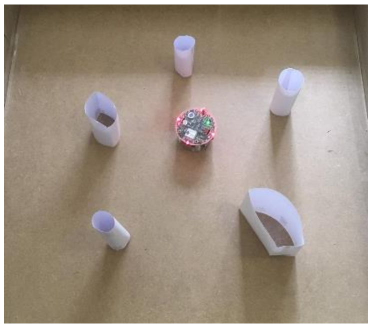


Figure 1 E-puck2 placed on the game board

The idea of this project is to place the robot onto a game board with different obstacles as indicated in figure 1. The player is then supposed to tilt the board such that the robot moves in a slalom around the obstacles. Each obstacle will have to be passed once to its right and once to its left to have the parkour completed. The goal is to use as little time as possible without getting too close to an obstacle.

2. Implementation of sensors and actuators

Proximity sensor readings, accelerometer measurements as well as motor speed updates are carried out using functions from the "e-puck2_main-processor" library. The function "chprintf()"[1], that is part of the ChibiOS, was used during the programming process to display the values measured by the sensors. The treatment of the sensor data, namely the calculation of the two motor speeds, is done in functions and threads that were created specifically for this project.

2.1. Accelerometer - Calculation of steepest ascent

Since the indicated values of acceleration depend on the position of the system of reference with respect to the gravitational field, a calibration on a horizontal surface is necessary. As the calibration process is initiated right after the startup of the robot, the e-puck2 needs to be placed on a table or on the floor when the power or reset button is pressed. Skipping this step may result in an uncoordinated movement of the robot. The axis of the frame of reference of the accelerometer of the robot are pointing in the directions as shown on figure 2[2].

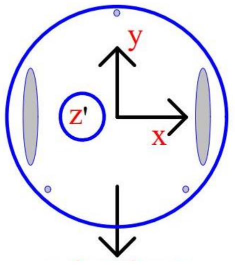


Figure 2 The coordinate system of the robot

Since the gravitational field is pointing downward, positive y-accelerometer readings correspond to the situation where the robot is facing uphill. The robot is thus programmed to rotate about the z-axis until it is facing uphill and then, the acceleration value along the x-axis is minimized.

The speed of rotation increases with an increasing error along the x-axis as a PID[3] controller is implemented. To prevent a halting rotating movement, a predefined threshold value for the x-axis is introduced. In the situation where the robot is facing downhill, the actual absolute value of the error along the x-axis decreases as well. The rotation of the robot is thus unintendedly slowed down. An additional proportional controller prevents this by increasing the speed of the rotation for negative y-values.

2.2. Proximity sensor – Obstacle detection

For the realization of this project, six of the eight TOF distance sensors are used. The four sensors situated at 17, 49 and 90 degrees to the right and left of the direction of movement respectively are used to check if an obstacle is in front of the robot. As the sensor values correspond to the intensity of the IR light that is reflected by an obstacle, larger sensor readings correspond to closer distances. To determine if an obstacle is too close, two different threshold values were defined for the different sensor locations.

In case the robot is too close to an obstacle, it will stop moving forward and rotate toward the direction where no obstacle is present. The rotation will stop as soon as there is no obstacle in front of the robot and the sensor readings of the sensors at 17, 49 and 90 degrees, with respect to the direction of movement, are lower than a threshold value. The threshold distance is about 2 cm and corresponds to a ToF sensor reading of 500. The experimentally determined sensor reading values that correspond to specific distances are represented in figure 3.

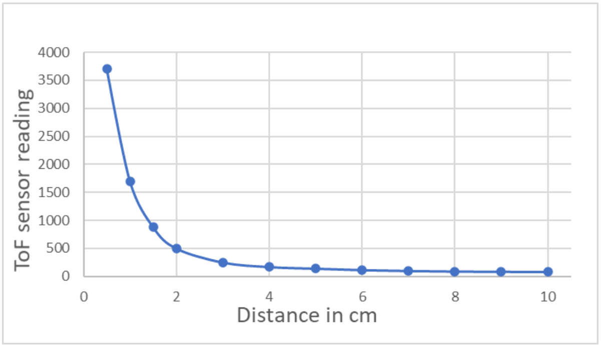


Figure 3 Proximity sensor values in function of the distance

2.3. Additional features

After having implemented the upward movement as well as the obstacle detection, the robot fulfilled its main task already. But, since the amplitude of the speed of the two motors move is constant in any situation, a rather annoying, halting movement of the robot is present. As described in section 2.1, a simple PID controller adds a motor speed correction and allows a fluid movement. In addition to the controller, instead of basing the speed calculation on a single accelerometer reading, averaging the acceleration measurements over five consecutive readings improves the fluidity of the movement even further.

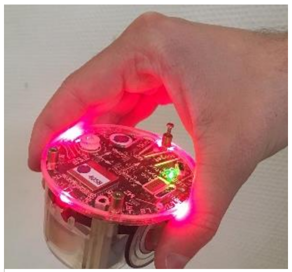


Figure 4 Default speed indication

Furthermore, to change the difficulty level of the game, the selector can be used to choose one of five different, predefined speeds. The red LEDs indicate the speed the device is set to. The default speed, 6.5 cm/s, is set when all four red LEDs are on as shown in figure 4. The slowest speed, corresponding to 2.6 cm/s, is represented by LED1, which is located above the camera. A 90-degree clockwise shift of the illuminated LED shows an additional increase in 2.6 cm/s of the speed. The fastest speed of 10.4 cm/s is thus represented by the LED situated close to the left wheel.

To make the handling of the robot easier, the motors will completely stop turning as soon as both proximity sensors at 90 degrees are covered. The threshold value to consider a sensor as covered was chosen to be 800 (about 1.5 cm). This value is chosen large enough such that accidental blocking of the motors is avoided. It is thus possible to lift the robot from the game board without it trying to move away by simply holding it on both sides and covering the sensors.

Moreover, the front LED as well as the body LED light up as soon as an obstacle is detected near the robot. The active LEDs are shown in figure 5. This flashing of the LEDs indicates that the current player did not manage to stay far enough from an obstacle.

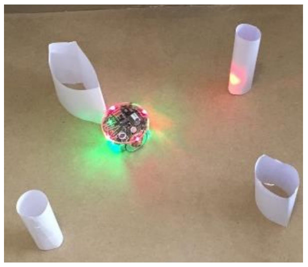


Figure 5 Robot detects obstacle

3. Code organization

3.1. General principles

The main function is used to initialize and calibrate the sensors and to start all threads. The threads used for this project are of standard priority and their scheduling is completely done by the Chibi RTOS.[4]

Flowcharts to describe the structure of each thread can be consulted in section 5. Generally, the threads called *MeanAccThd*, *FreePathThd* as well as *SpeedSelectThd* update environmental changes regularly. This actualized information is stored in static variables and can then be accessed by the thread move for further treatment.

Firstly, *MeanAccThd* calculates the average over five consecutive accelerometer readings to determine the mean orientation in space of the robot. Secondly, *FreePathThd* gets proximity sensor readings and checks for a path without an obstacle and, finally, *SpeedSelectThd* checks what speed the selector is set to and indicates it using the LEDs.

The principal program is contained in *MoveThd*. This thread uses the information gathered by the previously described threads to calculate the direction and speed of movement and actuates then the motors, accordingly, using that information.

3.2. Static and global variables

Overall, there are three static variables that are encapsuled in their own program section. These three variables are namely *mean_acc[AXIS]*, *speed* and *freePath*. The value of the latter is updated by the corresponding thread inside its own module and is accessed by *MoveThd* via a getter function. The variables *mean_acc[AXIS]* and *speed* can be directly accessed by *MoveThd* since they are defined in the same module. This thread can consequently gather their updated values whenever needed.

4. Conclusion

4.1. Limitations and possible improvements

Independent of the actual speed of the robot, only one set of PID controller parameters are used. Different factors KP, KI and KD could be experimentally determined, and implemented to get an optimal compensation effect for each selectable speed.

Instead of a simple slalom, the gameboard could be extended to a more complex obstacle course. For instance, following a wall or doing parts of the slalom all by itself could be implemented. Including other sensors such as the camera or the magnetometer would open a whole new range of tasks the robot could fulfill.

4.2. Overall results

The e-puck2 quickly finds the direction of steepest ascent, determines a free path, and moves flawlessly around obstacles. Obstacles of different appearance are placed on our game board to underline that fact. The robot reacts rapidly to changes of the environment. For example, blocking the path by placing the hand in front of it causes it to turn around immediately and to look for an alternative path.

A large gameboard is needed to install the entire setup since the obstacles need to be placed far apart. This is due to the size of the robot and the additionally needed space between it and the obstacles. Playing with such a large board is thus not very practical but, on the other hand, can make it even more fun.

The different programming tasks were divided among us, and regular meetings were organized to do experimentation as well as to discuss further improvements. By splitting the project up in sub-tasks, a well working code was implemented step by step.

Generally, the main goal that we defined at the start of the mini project was well achieved. It was interesting to experience how the original idea evolved to the final product. In addition, it is fun playing with the setup, especially since the control of the robot, always trying to move uphill, is counter intuitive.

5. Annex - Flowcharts[5] of the used threads

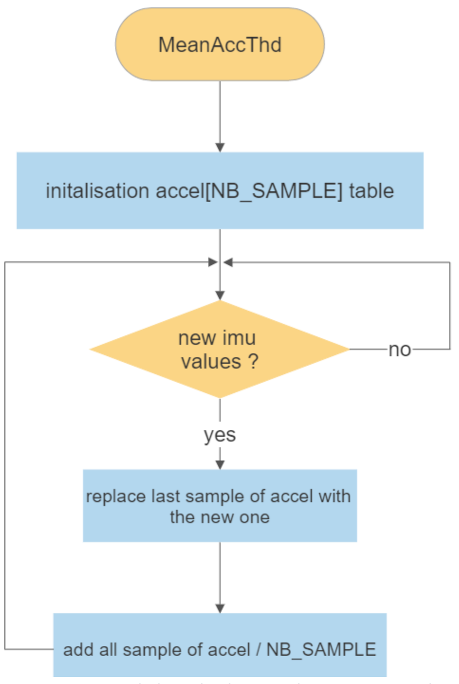

MeanAccThd
initalisation accel[NB_SAMPLE] table
new imu values ?
no
yes
replace last sample of accel with the new one
add all sample of accel / NB_SAMPLE

Figure 7 MeanAccThd calculates the mean value of the acceleration along the x-and y-axis of the last NB_SAMPLE values (=5). Its purpose is to reduce the halting movement of the robot by reducing the imprecision of the accelerometer

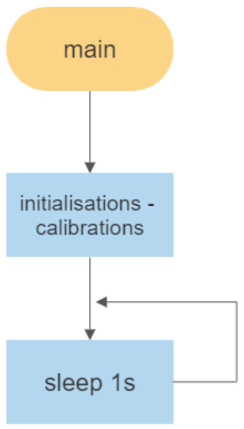


Figure 6 The main is only used to do the different initializations and calibrations needed and to start the threads (accelerometer, IR sensors, motors).

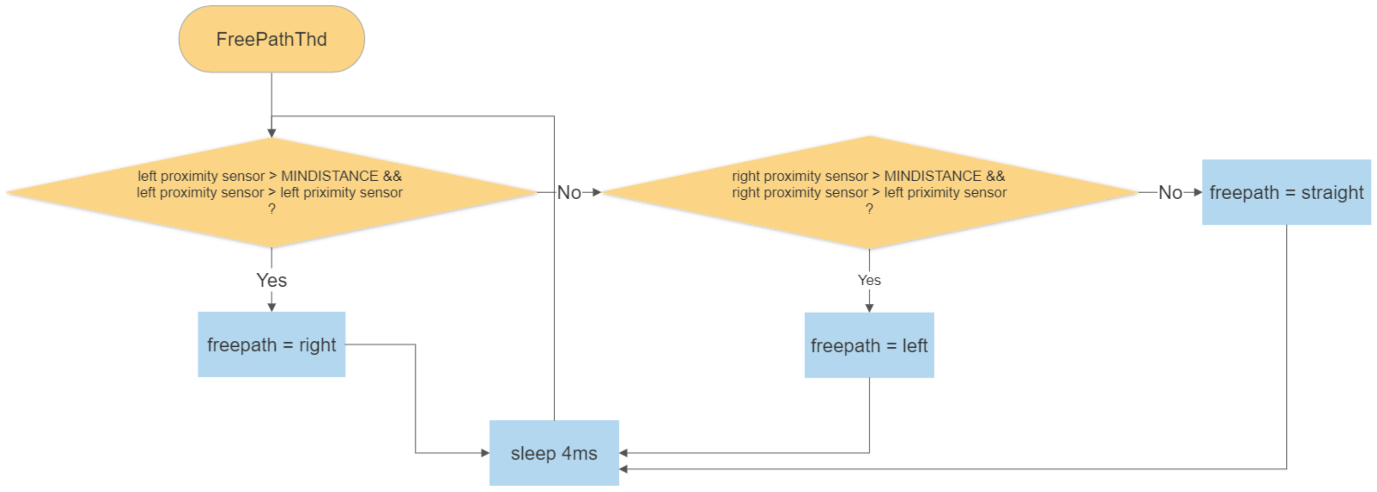

FreePathThd
left proximity sensor > MINDISTANCE && left proximity sensor > left prixmity sensor ?
No
right proximity sensor > MINDISTANCE && right proximity sensor > left prixmity sensor ?
No
freepath = straight
Yes
freepath = right
Yes
freepath = left
sleep 4ms

Figure 8 The thread FreePathThd reads the IR sensors to determine if there is an obstacle. If there is one in the direction of movement, the robot has to turn.

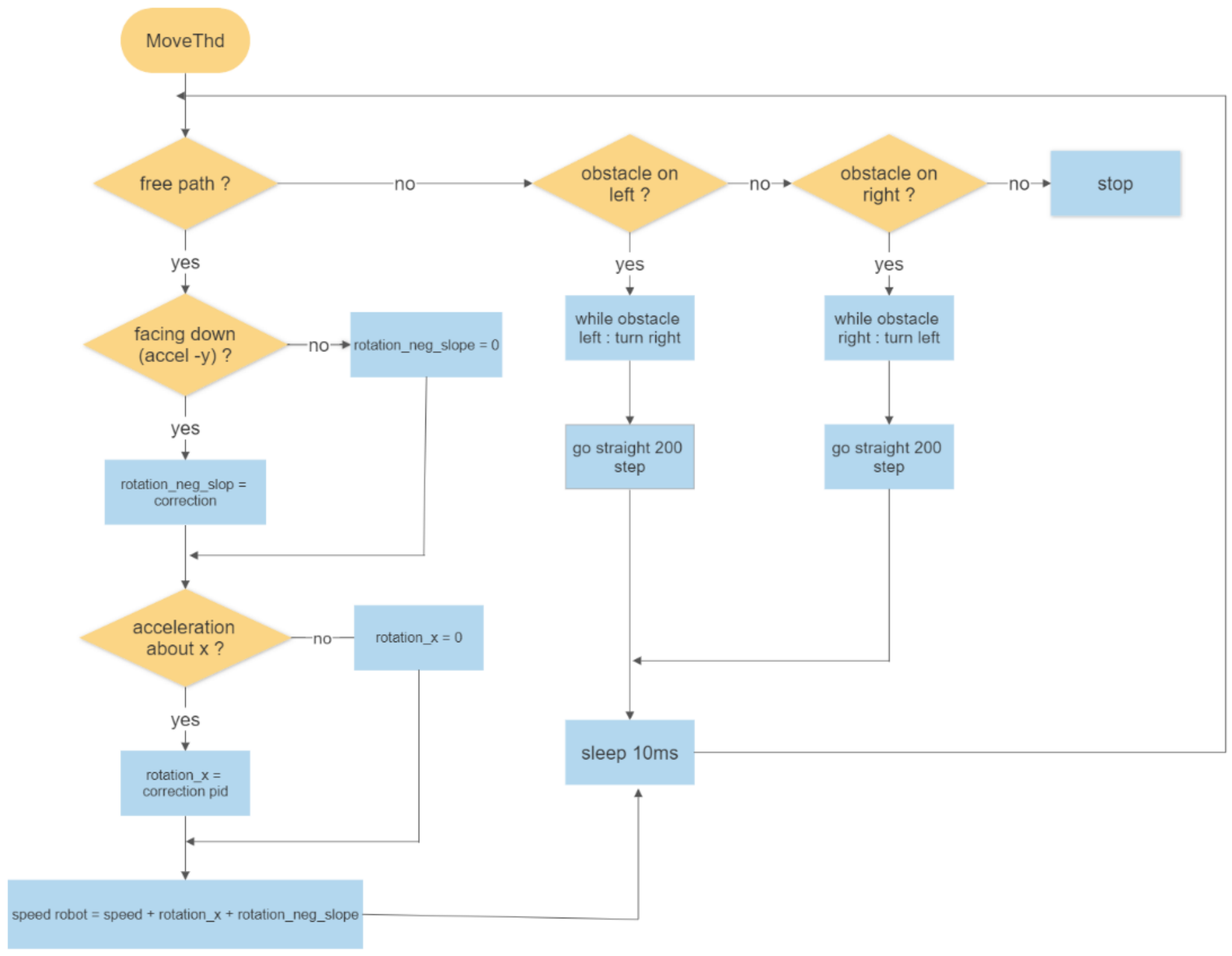


Figure 9 MoveThd is the principal thread that uses the different values that are determined by the other threads/functions of the program to assign the correct motor speed in function of the slope of the board and of the location of obstacles.

6. Sources and references

[1] Doxygen (30. Arp 2020), ChibiOS/HAL,
http://chibiforge.org/doc/19.1/hal/group___h_a_l___c_h_p_r_i_n_t_f.html, consulted: 22.4.2021

[2] Mondada, F. (2019), Gctronics. Dossier électronique de l'Epuck,
https://moodle.epfl.ch/pluginfile.php/2002535/mod_resource/content/9/e-puck2_F4-SCH-BOM-IMP.PDF, consulted: 15.4.2021

[3] Cours Microinformatique (Printemps 2021), Practical Exercise 4 : CamReg,
https://moodle.epfl.ch/pluginfile.php/22861/mod_resource/content/9/TP4Corrections.pdf, consulted: 1.5.2021

[4] Giovanni Di Sirio, ChibiOs EmbeddedWare, RT Threading,
https://www.chibios.org/dokuwiki/doku.php?id=chibios:%20|documentation:books:rt:kernel_threading , consulted: 5.5.2021

[5] SmartDraw, LCC (1994-2021), Flowchart maker, https://www.smartdraw.com/flowchart/flowchart-maker.htm , consulted: 13.5.2021

EPFL logo: https://www.epfl.ch/about/overview/identity/, consulted: 11.5.2021