EPFL

Embedded Systems and robotics (MICRO-315)

Mappuck Project Report

Authors:
Fabian Haller
Lukas Stuber

May 14, 2021

https://github.com/Fabiano197/Mappuck

Contents

1 Problem Statement

The goal of our project is the mapping of an unknown area with a robot called e-puck2. In literature, this problem is commonly referred to as SLAM (Simultaneous Localization and Mapping). In particular, the goal of the project is to map a continuous wall of an unknown area with correct dimensions and to represent those results on a computer screen. Moreover, the e-puck should be able to map the z position of the travelled course to give an idea of the surface topology. The focus of this project lies on the precision of the measured data. Thus, the shape and dimension of the mapped data should reflect reality as good as possible.

2 File and Thread Structure

In this section, a short overview of the file and thread structure of the project is given. The project consists of seven .c/.h files and four threads. A general overview can be seen in figure 1. For simplicity, .h files are not shown in the diagram. In addition to custom modules, the e-puck2_main_processor library was used [2].

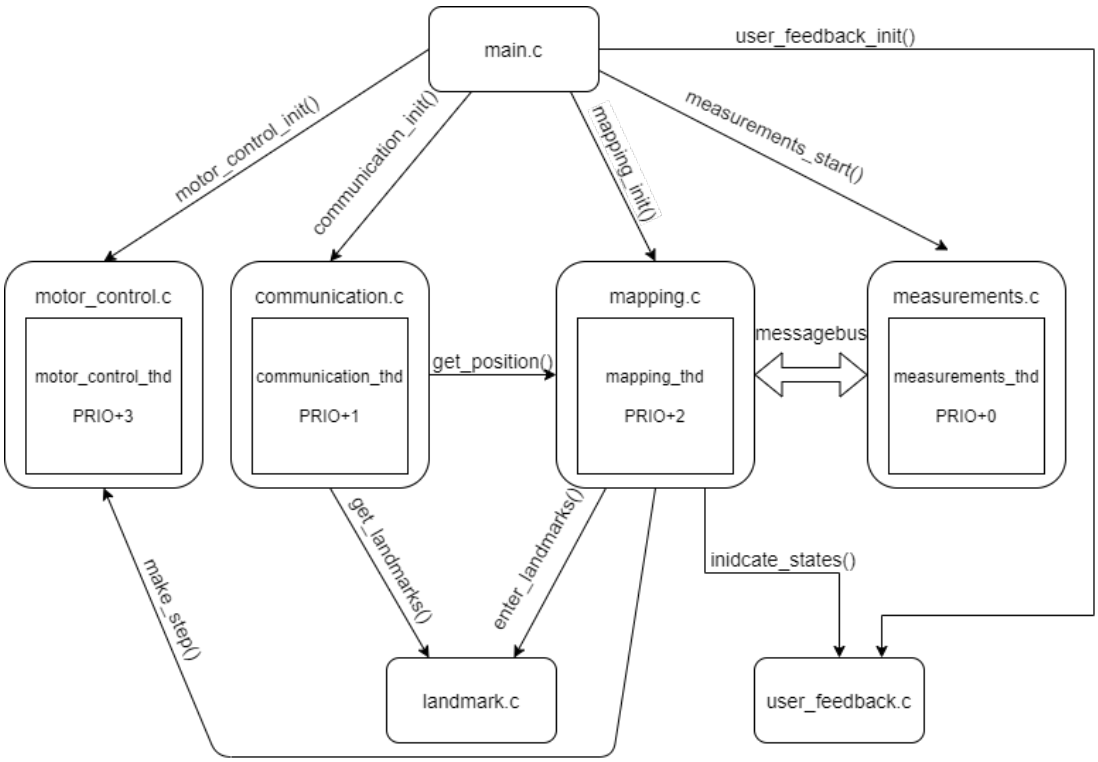


Figure 1: File and thread intercommunication diagram

2.1 main.c/main.h

The sole purpose of the main module is the initialization of the hardware abstraction layer (HAL), of ChibiOS and the other custom modules for the robot control. Thus, it only starts all necessary modules after a start/reset of the robot and is idle afterwards.

2.2 motor_control.c/motor_control.h

The motor control module is responsible for all motor related tasks. Since our project requires precise movement with as little error as possible, it is absolutely indispensable that the thread of this module has highest priority such that movements are timed as precisely as possible. The interface of this module can take an arbitrary rotation angle and step distance, which the robot will perform.

2.3 communication.c/communication.h

The communication module is responsible for sending position and map information to the external computer by Bluetooth. This enables a real-time monitoring of the robot by the computer. The priority of the communication thread is below average since small delays in sending information to the computer do not cause any problems. The thread is configured to send new information every second.

2.4 mapping.c/mapping.h

The mapping module is the core module of the project. It processes the sensor information collected by the robot and reacts with feedback (e.g. to not collide with a wall) and calculates the motor control command, which is sent to the motor control module. It also stores relevant environmental information, namely the landmarks discussed in subsection 2.6, for mapping purposes. The thread priority of this module is above average because fast data treatment is essential for fast movement.

2.5 measurements.c/measurements.h

The responsibility of the measurements module is the collection and conversion of sensor data. For this project, data of infrared proximity sensors, time-of-flight sensor and inertial measurement unit are collected. All collected data needs to be processed in order to remove possible offsets and/or to convert measured units to units which are relevant to our project (e.g. reflected light intensity measured by proximity sensors need to be converted to distance). The measurements thread has lowest priority to give priority to other modules and to only collect data if no other thread needs to perform calculations. If new sensor data is required by the mapping module, priority is assured by the thread structure (all other modules would wait in that case).

2.6 landmarks.c/landmarks.h

The landmarks module is responsible for storing and processing collected mapping data. It is equipped with a line-fitting algorithm to calculate a wall polygon out of the collected sensor data. In contrast to the other modules mentioned above, this module does not possess an own thread. More information about the functional principle is given in subsection 5.3.1.

2.7 user_feedback.c/user_feedback.h

This module serves to give visual and acoustic feedback to the user. It is entirely optional and does not contribute in any way to the correct running of the other modules. It gives feedback each time the mapping module starts a new mapping stage (cf. section 5).

3 Initial Tests

The accuracy of all the sensors had to be tested before they could be used effectively. The tested sensors are the IR proximity sensor, the time-of-flight distance sensor and the accelerometer.
For the IR sensor, an inverse quadratic relationship between measured intensity and distance was adequate (figure 2). The proportionality constant connecting the two values had to be determined

empirically (cf. equation 1).
The TOF sensor was already provided with a function returning the measured distance. However, that function had an offset of 52mm (cf. equation 2). That value was found empirically as well, using a linear regression line on GeoGebra.

$$d_{IR} = \frac{c_{prop}}{\sqrt{I_{measured}}} \tag{1}$$

$$d_{TOF} = d_{measured} - d_{offset} \tag{2}$$

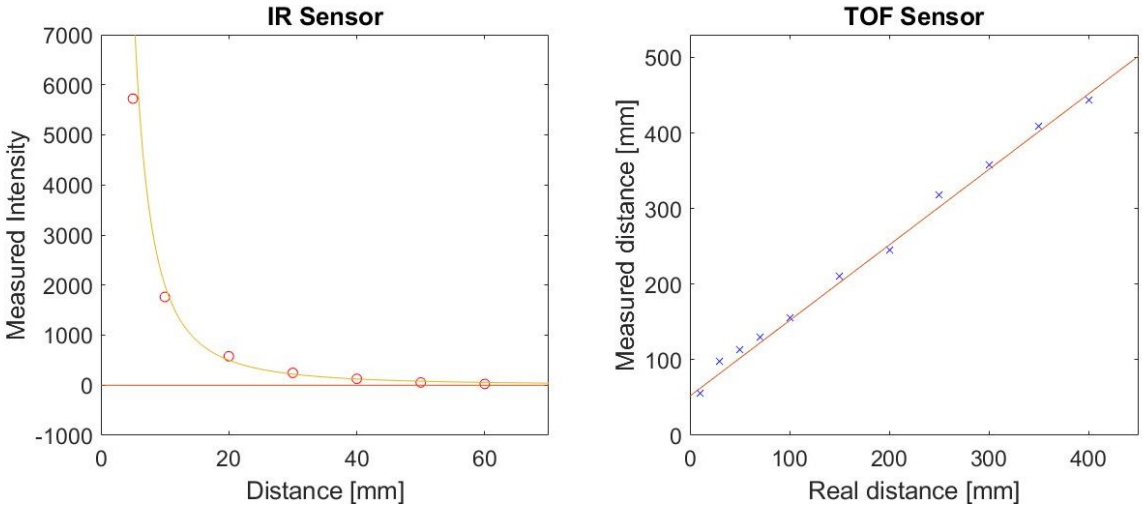


Figure 2: Initial tests of the sensors

The IMU module of the e-puck2_main_processor library is provided with a function that returns the acceleration of a given axis. With that the inclination angle could be easily calculated. But on flat ground the angle was constantly about 1° negative. So an offset of 1° was added to the inclination angle. That value was determined on a closed loop. The angular offset was adjusted such that the height was always the same at the start point after one round. Since a full round is composed of several hundred steps the offset could be calibrated very precisely. That guaranteed as well for an accurate height calculation on inclined ground.

4 Development Steps

1. Definition of problem statement (cf. section 1)
2. Establishment of the file and thread structure (cf. section 2)
3. Sensor testing (cf. section 3)
4. Implementation of wall tracking
   In the simplest form, the tracking of a wall can be done with only one proximity sensor at the side, in our case the right side. But this setup is not very stable and cannot track sharp edges of the wall. Instead, we chose to use a linear combination of the signals from the right and front-right sensor to calculate the motor control signal. The weights of these two signals had to be found experimentally. Additionally, the TOF sensor turns the robot counterclockwise if it detects a wall closely ahead of itself.

5. Construction of a test area
   The test area is made out of a wooden ground plate. Wooden posts are screwed to the ground plate and form the structure of the walls. The walls are made of carton that is nailed to the structure posts. The shape of the contour has both convex and concave curves and corners.
   Since the TOF sensor does not point straight forward but a little up, the walls need a certain height. Otherwise, the laser would sometimes pass over the walls and the data would become unreliable. The test area can be seen in figure 3.

6. Implementation of the landmarks
   After every centimeter, the robot saves its position and the position of the right wall as landmarks. These are sent out by Bluetooth. In figure 4, the measured data is shown.

7. Mapping with Python
   The external computer is running a python script with a thread that constantly reads the data from the Bluetooth port. The data gets plotted on a graph. Whenever a new data package is received, the plot gets updated.

8. Processing of wall landmarks
   As a last step, all wall landmarks are processed. To do so, the microcontroller of the robot calculates a line-fitting polygon of the wall landmarks. Subsection 5.3.1 gives more information about the line-fitting algorithm.

Figure 3: Top view of test area

5 Mapping Stages and Technical Implementations

This section explains the different stages of the mapping algorithm and gives a brief overview of their implementation.

5.1 Stage 0: Startup

During the startup stage, all peripherals are initialized and time is given to the user to place the e-puck to its start location. This stage lasts for five seconds. It is crucial to neither move nor interfere with

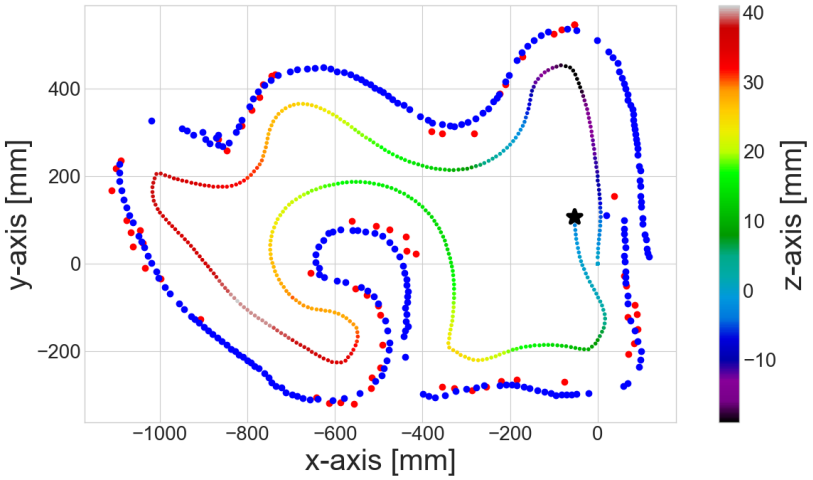


Figure 4: Plot with all measurements of robot, IR proximity in blue, TOF in red and estimated position in colours of colour bar

the robot after its startup stage because all collected data is stored relative to its starting position and the results would become imprecise otherwise. A visual and audible startup countdown is signaled to the user to facilitate the timing.

5.2 Stage 1: Find Borders

For the actual wall mapping stage (stage 2), it is necessary that a wall is situated close to the right side of the robot (at 90° clockwise from TOF sensor and camera). Therefore, stage 1 consists of creating the desired conditions. To do so, the robot will rotate in fractions of a full revolution and measure the distance to the next wall at each position. After one full revolution, it will rotate again in order to face the direction of the smallest measured distance. The smallest distance is chosen in order to guarantee that the wall is (locally) perpendicular to the direction faced by the robot. It will then move to the wall until the desired wall distance is reached and then turn 90° counterclockwise which will guarantee that the wall is on the right of the robot (since the wall was perpendicular to the moving direction before). A short visual and audible feedback is given to the user to indicate the completion of this stage.

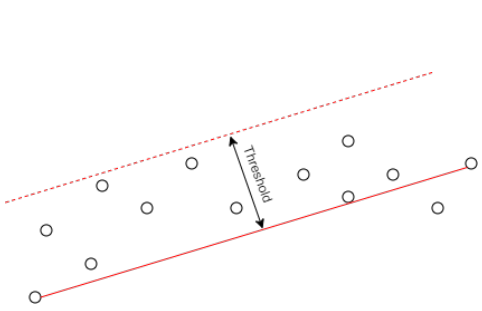
5.3 Stage 2: Follow Borders

During this stage, the robot is following the border of the unknown area. It is controlled such that the distance to the wall is kept constant by use of feedback from IR proximity and TOF sensors. Feedback is calculated in steps and each step is performed in the following order: collecting sensor data, calculating feedback based on the sensor data, change direction based on feedback, step forward for a predefined distance (defined it to be $1cm$ for this project). This step is repeated as many times as it takes to get back to the initial position (i.e. the whole wall contour has been mapped).

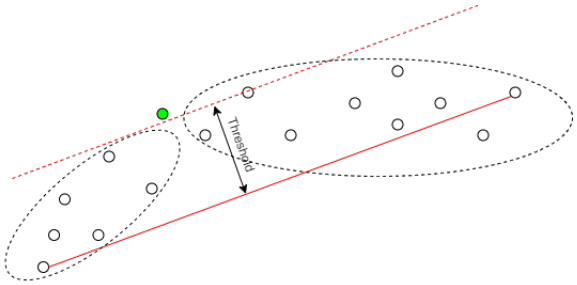
Before performing the first step, a coordinate system with the origin set to the position where the robot starts following the walls is created. After that, it updates its position relative to the initial position after every step. In addition to that, based on its position, IR sensor and IMU, the robot stores the location of the wall (later referenced as wall landmark) and its 3 dimensional position (later referenced as surface landmark) where the z-axis is obtained by integrating the slope calculated by the IMU. The raw collected data can be seen in figure 4. The wall landmarks are then processed by a line-fitting algorithm described in the next subsection (5.3.1).

5.3.1 Wall Landmark Processing

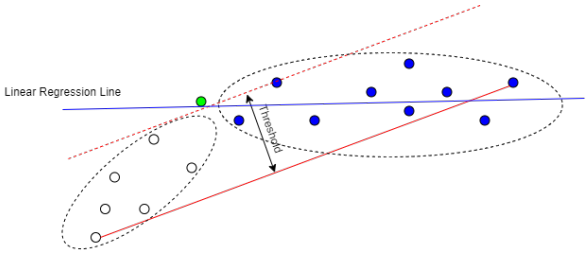
The purpose of this subsection consists of giving a short explanation on how the line-fitting algorithm to process wall landmarks works. For more detailed information, please refer to section 3.2 in [3].



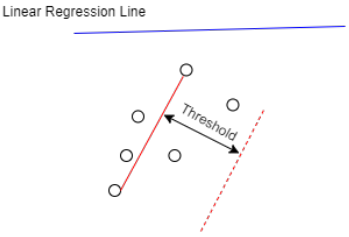
(a) No wall landmark exceeds threshold



(b) One wall landmark exceeds threshold, points get split in two subgroups



(c) Group of old wall landmarks are fitted with linear regression



(d) Fitted wall landmarks get deleted

Figure 5: Step for step illustration of the line-fitting algorithm

In figure 5 the different steps of the line-fitting algorithm are presented. In a first step, the algorithm draws a line between the oldest and newest wall landmark and calculates the maximal distance between this line and all other wall landmarks stored at this moment (cf. figure 5a).
The execution of the rest of the algorithm depends on whether the previously calculated distance exceeds a predefined threshold value. If that is not the case, the algorithm stops and will be called again after a new wall landmark has been added. However, in case the distance of one wall landmark to the line exceeds the threshold, the wall landmarks are split into two groups (cf. figure 5b, in green the wall landmark which exceeds threshold and in dashed circles the two groups).
The group where the oldest landmark is located is then fitted by linear regression while the group with the newest landmark rests untouched (cf. figure 5c, in blue the linear regression line).
In a last step, all wall landmarks in the old group get deleted and only the regression line is stored (cf. figure 5d). This algorithm is re-executed each time after a new wall landmark has been entered into the system. As soon as two regression lines have been calculated, the intersection of those is calculated, stored and the older regression line deleted. During the course of the mapping, many of those intersections are calculated and they will form the corners of the polygon, which fits the walls. The complete polygon of test data acquired by the e-puck can be seen in figure 7.
Finally, a short remark to the choice of the threshold value used by this algorithm: If it is chosen too small, an overfitting occurs and measurement deviations are not smoothed out which would deteriorate the main purpose of this algorithm. In contrast, if the threshold is chosen too large, smaller features are not mapped correctly and the resolution suffers. Thus, the choice of the threshold value mainly determines the performance of this algorithm.

5.4 Stage 3: Loop Closure

The mapping of the wall stops as soon as the robot returns to its initial position. Since the estimated position of the robot is affected by measurement deviations, the probability that the estimated position perfectly fits the actual position is extremely low. In order to correctly detect the returning to the initial position, a so-called close-loop radius is defined. The mapping will immediately stop as soon as the robot's estimated position lies within the close-loop radius of the initial position. It will then stop moving and collecting data. A melody is played and the LED's blink in many different colours to indicate the successful completion.

6 Results

Figure 7 shows the result obtained by the robot on the test area. The robot has arrived inside the predefined close-loop radius of $100mm$ around the initial position, which closed the loop. The gap between the robots position and the initial position (origin of the coordinate system) is the close-loop radius. The threshold distance of the line-fitting algorithm (refer to subsection 5.3.1) is chosen to be $25mm$. The defined speed of the two wheels is $26mm \cdot s^{-1}$.
The overall shape of the polygon matches the real shape approximately. Figure 8 shows the comparison between the fitted polygon and the walls. The start and end positions of the robot are off by less than $10mm$ in distance and less than $5°$ in angle. However, it is possible that there is a shift in the x-direction that cannot be seen on the graph. The length of the polygon in figure 7 (point $(-1100mm, 300mm)$ to point $(150mm, 500mm)$) is $1220mm$. The real distance measured on the test area is $1160mm$. That is an error of 5.2% of the x-dimension of the plane. The width of the plot is $815mm$ (point $(0mm, -100mm)$ to point $(-60mm, 730mm)$. The real width is $780mm$. The error of the y-dimension is 4.5%.
The span of the z-coordinate is $112mm$ which corresponds to the height of the DVD case tower we used as support structure for the test area. Figure 6 shows the height of the highest corner of the test area (corresponds to the point $(-1100mm, 350mm)$ in the graph of figure 7). From that height of $113mm$ we need to subtract $19mm$ when we compare it to the span of the z-coordinate in the figure because the robot does not drive to the very edge of the board and thus covers a smaller height. That gives an error of 19.1% of the z-dimension.
A video of the mapping is available on GitHub (link provided on the title page).

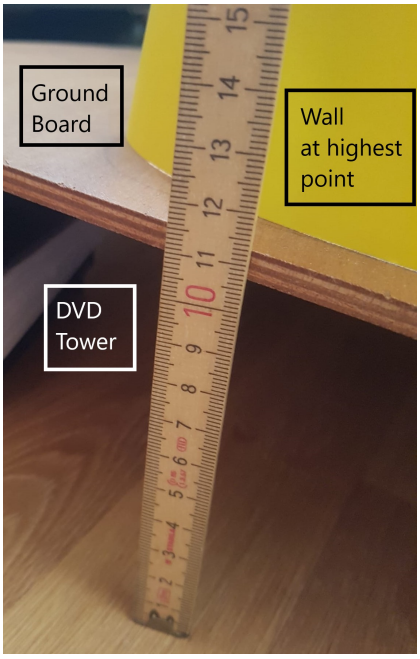


Figure 6: Side view showing the support structure

7 Conclusion

7.1 Achieved Features

The robot can be placed in an unknown environment with walls. It finds the closest wall and drives towards that point. Then it follows the wall with a satisfying accuracy. While tracking the wall it measures and records its own position and those of the walls that it has already passed by. The

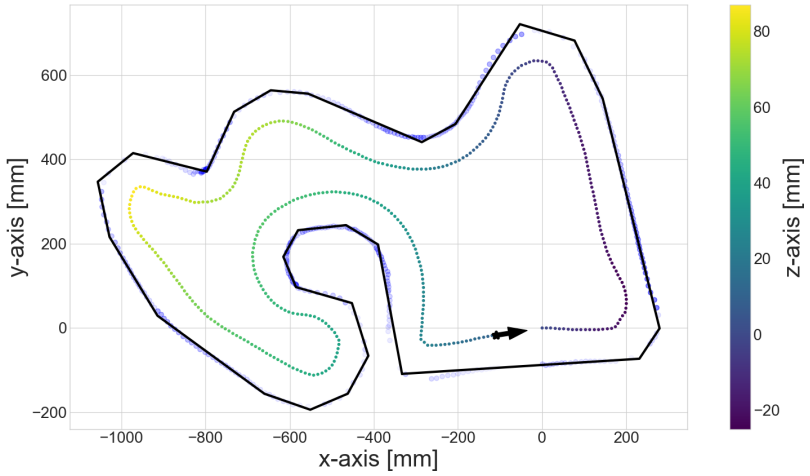


Figure 7: Complete plot of one turn around the test area with a maximum elevation of 10cm

Figure 8: Shape comparison of the polygon and the test area. Note that the result in figure 7 is from a different run

program runs a line-fitting algorithm to transform the wall measurements into a polygon. The surface landmarks, the wall landmarks and the polygon corners are sent by Bluetooth to a computer. On the computer a program is running that receives, decodes and plots the data sent by the robot.
All these functions work within our expectations.

7.2 Discussion of Results

The result shown in figure 7 have satisfying quality. The errors of the length and width of the plot are within reason. In contrary, the error in the z-axis is quite high. While the relative slope measured by the robot fits the actual test area very well, the absolute numbers have a fairly big error. This could have been corrected by rescaling the measured z positions but has not been done since it was

not observed until this point.
The shape of the polygon fits on the real walls in most points. That indicates that the threshold distance of the line-fitting algorithm is well chosen, such that there is neither overfitting nor underfitting. The offset of the estimated final position compared to the actual position is very small for the y- and z-axis and the angle. In the x-direction it is visible that the last line of the polygon in figure 8 is much longer than the actual length of the corresponding wall which shows an offset between the estimated and actual x-position at the end of the mapping.

7.3 Limitations and Possible Improvements for the Future

7.3.1 Accuracy of the Sensors

All the sensors have a limited resolution and contain noise. The IR sensor is only effective in a range up to $7cm$ and is very sensitive to different light conditions. To improve the measurements, it would be beneficial to use a TOF sensor, instead of a proximity sensor, to monitor the walls on the right side of the robot. With more precise measurement data, the resolution of the fitted polygon could be improved since one could choose a lower threshold value for the line-fitting (cf. section 5.3.1).

7.3.2 Precision of the Position and Landmarks

The precision of the position calculation is good and the error very small. However, small errors are unavoidable and are caused by many error sources such as non-perfect friction of the wheels on the ground. Thus, the imprecision of the estimated position with respect to the origin gets bigger and bigger during the mapping procedure due to error propagation.
The only solution to diminish the uncertainty of the position is a feedback system. In order to implement such a feedback system, the robot must be able to uniquely recognize previously mapped features. These features are usually referred to as landmarks in SLAM algorithms. There are multiple of those feedback systems such as the Extended Kalman Filter or Fast SLAM. The problem in case of our project is, that the unique identification of mapped features turns out to be extremely challenging given only the data of the IR proximity and TOF sensors. An implementation with only TOF sensors has already been performed by using 90° corners as landmarks [3]. Unfortunately, our map consists of smooth walls and no easily recognizable features exist, which could be used as landmark. With more advanced data processing, it might be possible to uniquely identify features, which could be used as landmarks but due to a lack of time and computational resources, it was not implemented. Nonetheless, the mapping performance could have been improved with such a feedback system.

7.3.3 Interior Mapping and Island Detection

The robot never maps the interior of the test area. Thus the program at this state is suited for the mapping of an inclined plane but not for a non-linear topology with hills and valleys. Moreover, the robot cannot detect obstacles in the interior of the unknown area, which are not connected to the walls. In order to map the interior, it would be indispensable to have a feedback system since the position incertitude would grow too much otherwise. One could then calculate an inset polygon (out of the wall polygon) and follow the lines of the inset polygon [1]. The TOF sensor could then be used as a feedback to estimate the position.

7.3.4 Steps, Holes and Gaps

The robot is extremely susceptible to uneven terrain. It is incapable of moving over small steps and the wheels lose contact very easily to the ground. This limits the usage of the robot to areas with very smooth surfaces. A more robust four wheel system would greatly improve the range of use.

References

[1] Fernando Cacciola. *CGAL 5.2.1 - 2D Straight Skeleton and Polygon Offsetting*. 2021. URL: https://doc.cgal.org/latest/Straight_skeleton_2/index.html#:~:text=An%5C%20offset%5C%20polygon%5C%20can%5C%20have,the%5C%20distance%5C%20is%5C%20sufficiently%5C%20large). (visited on 05/13/2021).

[2] Eliot Ferragni, Daniel Burnier, and GCtronic. *(GNU) e-puck2_main-processor*. 2021. URL: https://github.com/e-puck2/e-puck2_main-processor (visited on 05/14/2021).

[3] Manigandan Nagarajan Santhanakrishnan, John Bosco Balaguru Rayappan, and Ramkumar Kannan. "Implementation of extended Kalman filter-based simultaneous localization and mapping: a point feature approach". In: *Sādhanā* 42.9 (Sept. 2017), pp. 1495–1504. ISSN: 0973-7677. DOI: 10.1007/s12046-017-0692-y. URL: https://doi.org/10.1007/s12046-017-0692-y.