Basic Navigation Strategies for Mobile Robots - 2nd Lab Assignment

Manuel Madeira (83836), Maria Teresa Parreira (83844)

June 12, 2019

1 INTRODUCTION

The present Laboratory Assignment concerns the Pioneer P3-DX robot and it aimed at achieving the navigation of the robot around the North tower 5th floor (see Fig. 1.1), starting from inside the lab and returning to the starting point. Furthermore, the robot should detect doors and determine if they were fully open, half open, or closed; upon detection, it should turn with front pointing towards the door and issue a sound according to the door status.

In order to achieve the objectives set for this assignment, we relied on the software *MATLAB*.

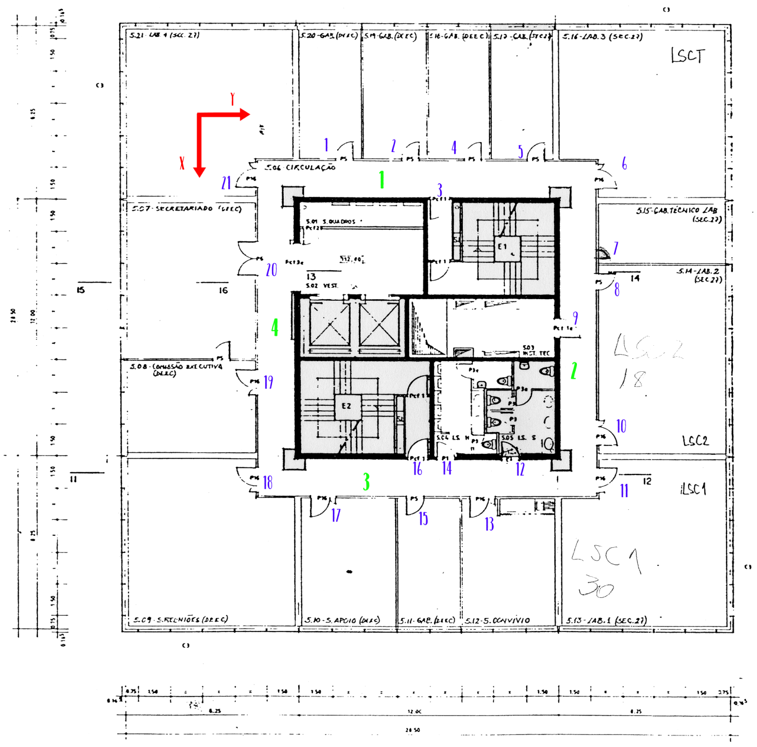

Figure 1.1: Floor 5 of North Tower. In red, the global ("world") frame. In blue, the number attributed to each door on the floor. In green, the numbering of each corridor, in regards to the order with which they were completed.

2 SENSORS

In order to provide positioning references for the robot, the information provided from three different sensors was used: the robot's own odometry, a Lidar and the robot's sonars.

2.1 Odometry

The robot has an internal sensor for position, calculated from the rotation of each of its wheels. This will provide (x_G, y_G, θ_G), *i.e.* the robot's position and orientation in regards to the global frame, whose origin and orientation of axis are defined by the robot's local frame when it is initialized (at $t = 0$). Note that the local frame is, at each moment, defined by (e_{x_L}, e_{y_L}), with e_{x_L} pointing forth and e_{y_L} pointing to the robot's left.

This is, however, not a reliable measurement, as it only accounts for the activation of the motors and rotation of wheels but has no external feedback to check whether or not the commands are being properly followed. For this reason, it is very susceptible to the surroundings irregularities (inclination of the floor, for instance) and to the robot's own physical conditions (if the wheels have different radius, or dirt on them, etc.). Using robot's odometry as a localization strategy, by itself, is therefore a huge risk and insufficiently robust approach.

2.2 Lidar

A Lidar is a surveying method that measures distance by illuminating the targets with pulsed laser light and measuring the reflected pulses with a sensor. The Lidar used scanned 682 points, returning a distance (in mm) associated with each of them in a maximum range varying from 4-5+ meters. It covered a range of 240° and is associated with a 1 second delay comprising the scan acquisition and transmission to the computer. Although it is a quite robust source of information on the surroundings of the robot, some limitations must be pointed out: first, upon collecting scans, there were some outliers, that is, points whose distance measure was not consistent with the neighboring points; this would happen in particular for longer distances. In order to deal with this, a preprocessing stage was required after scan acquisition but before analysis (see Section 2.2.1 for more information).

The amount of time required for the scanning process to take place constitutes another major limitation of using this sensor as the only source of orientation for the robot, as it would result in a significant interval between position updates. In order to ensure that each scan being processed is an updated one, a timer can be used .

2.2.1 Scan Preprocessing

Each Lidar scan is composed of an array with 682 elements, each containing the distance measured. This can easily be plotted in a 2D x-y "map" by first translating each point to an angle ($\gamma \in [-120, 120]°$, clockwise) and then making use of trigonometry (Eq.2.1, where k_i is the k^{th} element of the scan array).

$$x_i = k_i \times cos(\gamma); y_i = k_i \times sin(\gamma) \tag{2.1}$$

Additionally, each point with a distance above 4000mm or below 10mm was excluded from the preprocessed scan - the final array is then of variable length. This preprocessing can be consulted on file *LidarCorrection.m.*

2.3 Sonars

The sonars embedded in the robot can also be used for localization and scanning of the surroundings. In total, there are eight sonars, with sensitivity ranges from 10 centimeters to over 4 meters, depending on the ranging rate. The function *pioneer_read_sonars.m* provides an array of eight distances measured, in mm, each with respect to one sonar (Fig. 2.1 illustrates numbering and placement of sonars) However, this method for orientation is associated with a number of limitations: first, we noticed upon taking measurements that distances under 200mm were received as being 5000mm; then, sonars are extremely sensitive to reflections in the materials being scanned - making the measurements unstable and less reliable, especially when compared to Lidar readings. Finally, we should mention that the sonar is subjected to a time interval between acquisitions of around 300 miliseconds.

In light of these facts, we decided to rely on the Odometry for general localization, on the Lidar for correction of position, orientation and door status detection and the sonars, being the least robust sensor, for detection of objects on the path (these strategies will be detailed below).

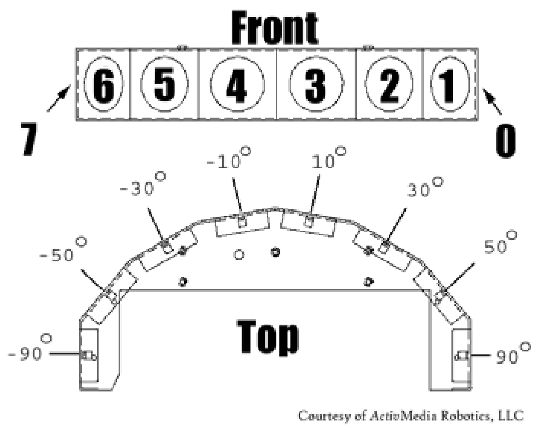

Figure 2.1: Pioneer's embedded sonars (8 in total) and their orientations with respect to the robot's front.

3 Kinematics

Autonomous motion planning and control of wheeled mobile robots is a field of much research interest in the field of robotics. The robot model used in this assignment is a two wheeled differential drive robot, where each wheel is driven independently, which provides good manoeuvring; mobile robots with such drive systems are a typical example of non-holonomic mechanisms due to the perfect rolling constraints on a wheel motion (no longitudinal or lateral slipping).

Forward motion is achieved when both wheels are driven at the same rate, turning right is achieved by driving the left wheel at a higher rate than the right wheel and vice-versa for turning left. The Non Holonomic constraint expresses the assumption that the wheels do not slide and can be expressed by Eq. 3.1.

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{3.1}$$

With respect to the angular velocity of each wheel, linear and angular velocities of the robot can be calculated as (Eqs. 3.2,3.3):

$$v = (v_{\text{right}} + v_{\text{left}})/2 = (\omega_{\text{right}} + \omega_{\text{left}})r/2 \tag{3.2}$$

$$\omega = (\omega_{\text{right}} - \omega_{\text{left}})r/(2d) \tag{3.3}$$

where r is the radius of the wheel and d is the axial distance between wheels.

3.1 Controller design

The controller used for the robot was based on Malu,Majumdar (2014) [1] and it is a Point Tracking controller, *i.e.* the robot is oriented towards previously defined points in space, rather than being controlled by means of a trajectory. Hence, using the robot's current position and orientation, (x_c, y_c, θ_c) and the next desired position (x_d, y_d), the distance and rotation of the robot are defined in Eqs. 3.4 and illustrated in Fig. 3.1.

$$\begin{aligned}
&\rho = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}\,,\ \dot{\rho} = -v\cos\alpha \\
&\beta = \operatorname{atan2}(y_d - y_c, x_d - x_c)\,,\ \dot{\beta} = v\sin\alpha/\rho \\
&\alpha = \beta - \theta\,,\ \dot{\alpha} = -\dot{\theta} + v\sin\alpha/\rho
\end{aligned} \tag{3.4}$$

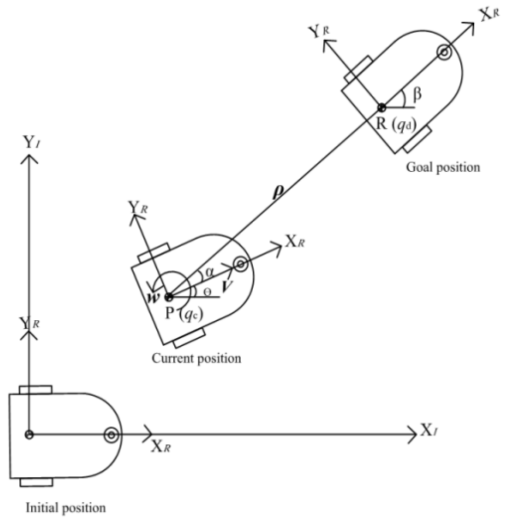

Figure 3.1: Error vectors of robot's position and orientation [1].

Note that there kinematics equations, based on polar coordinates are valid when $\rho \neq 0$, which is not relevant if a tolerance radius around the desired point is defined (in this case, it corresponded to 50mm).

3.2 Proof of Lyapunov Stability

From this point, a linear and angular velocities (v, ω) must be defined in order to minimize ρ and lead the robot to the desired position *in a finite interval of time.*

In order to study the asymptotic behaviour of the controller, by determining which $(v, \omega) = f(\rho, \alpha, \beta)$ guarantee that the robot will be driven to $[0, 0, \beta]$ in finite time, one can make use of the Lyapunov Stability Theory. If we consider a positive definite quadratic form of the Lyapunov function, $V = v_1 + v_2$, where v_1 is one half of the squared weighted norm of ρ, the distance error vector, and v_2 is one half of the squared weighted norm of α, the orientation error vector, we get:

$$\begin{aligned} V &= \frac{1}{2}\rho^2 + \frac{1}{2}\alpha^2 \\ \dot{V} &= \dot{\rho}\rho + \alpha\dot{\alpha} \\ \dot{V} &= \rho(-v\cos\alpha) + \alpha(-\dot{\theta} + v\sin\alpha/\rho) \end{aligned} \tag{3.5}$$

where we use the kinematics equations derived in 3.4. If the linear and angular velocities are defined as:

$$\begin{aligned} v &= K_\rho \rho \cos\alpha \ (K_\rho > 0) \\ \omega &= K_\rho \sin\alpha \cos\alpha + K_\alpha \alpha \ (K_\rho, K_\alpha > 0) \end{aligned} \tag{3.6}$$

then $\dot{V}_1$ and $\dot{V}_2$ can be made non-positive:

$$\begin{aligned} \dot{V}_1 &= \rho(-K_\rho \rho \cos^2\alpha) = (-K_\rho \rho^2 \cos^2\alpha) \leq 0 \\ \dot{V}_2 &= \alpha(-K_\rho \sin\alpha \cos\alpha - K_\alpha \alpha + \frac{K_\rho \sin\alpha \cos\alpha}{\rho}) = -K_\alpha \alpha^2 \leq 0 \end{aligned} \tag{3.7}$$

This means that both V_1 and V_2 are non-increasing in time and will asymptotically converge to a non-negative finite limit. Therefore, we get $\dot{V}$ in a negative semi-definite form which, by applying Barbalat's Lemma - if $f(t)$ has a finite limit as t$\rightarrow \infty$ and if $\dot{f}$ is uniformly continuous (or$\ddot{f}$ is bounded), then $\dot{f}(t) \rightarrow 0$ as $t \rightarrow \infty$ - we get that a controller whose velocities (v, ω) are determined by Eqs. 3.6 will conceive the robot motion behaviour as smooth and stable, given the right calibration.

3.3 Simulation

A simulation of the robot's motion over a set of desired positions was computed (file *test_SIM.m*), providing as goal points the corners of each corridor, whose length was defined roughly as 16m. The controller defined the velocities as they are presented in the previous section, with K_ρ, K_α being empirically determined so as to achieve smooth transitions in the velocities profile over time. The best results are presented in Fig. 3.2, for $K_\rho = 1, K_\alpha = 0.5$ and a radius of tolerance of 50mm. In order to prevent the robot from hitting walls on the corners, the strategy delineated involved first a cycle where the linear velocity is set to 0 and the robot rotates only, until it is properly oriented to the next desired position, and then a second cycle where the robot can both move and rotate, approximating itself to the point.

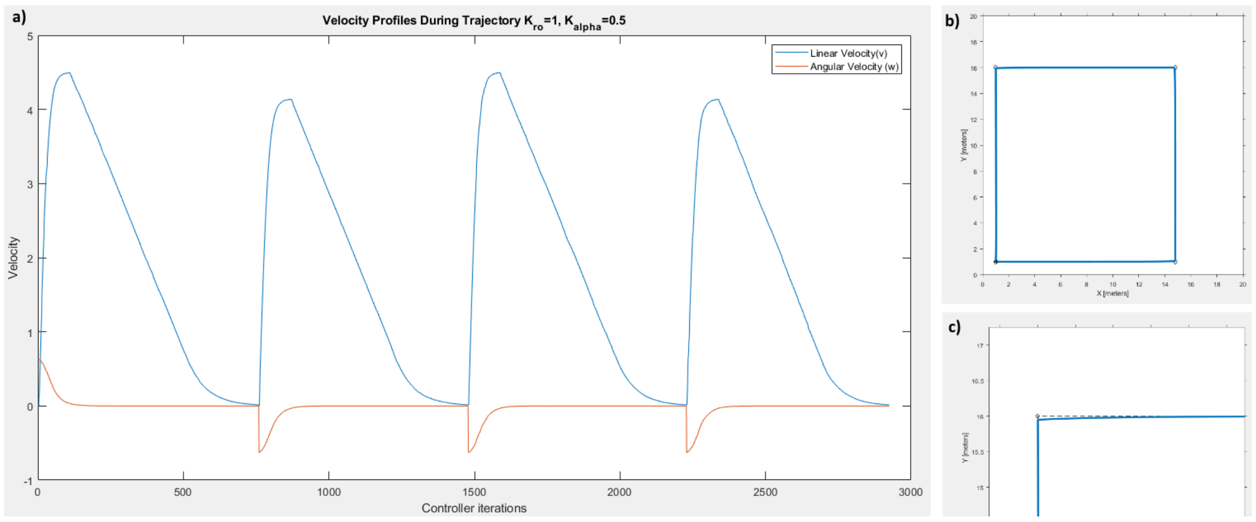

Figure 3.2: Results of simulation: (a) velocities profile (blue - v, orange - ω) over controller iterations; (b) map of positions of robot over the duration of the simulation; (c) detail of map on top left corner of trajectory. The desired point and straight line between point are displayed and a small deviation of the robot can be seen, which is a consequence of the tolerance radius defined.

4 Tasks and Methods

4.1 Desired Capabilities

In line with the objectives for this assignment, the robot's was set to fulfill a number of tasks in the most robust way possible:

- achieve automated navigation through the corridors, not hitting any fixed obstacle on the way (walls, benches);
- update its position in accordance to the surroundings, in order to overcome the already mentioned limitations of the odometry-based localization;
- stop when there is a door (in a total of 21 doors on the path), turn its front to the door and determine the door status (closed, half-open or open);
- be able to detect when objects are close to its front and stop if the distance to the object is small enough, avoiding collision.

The way to achieve this can be seen in Fig. 4.1 and will be detailed in the following subsections.

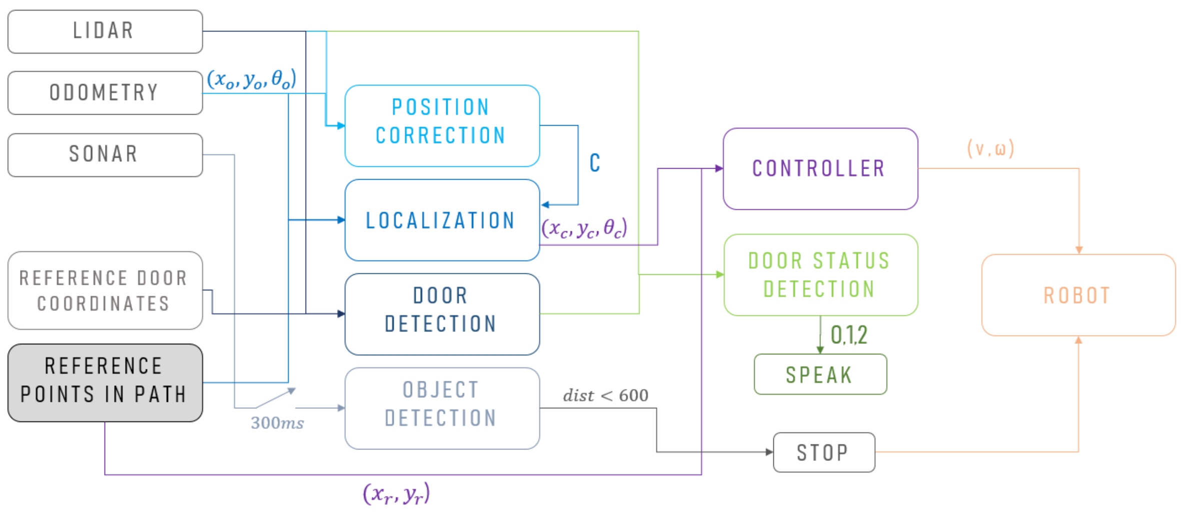

Figure 4.1: Block diagram representing the overall strategy for robot locomotion according to the objectives of the assignment.

4.2 Orientation in space

The robot starts its path inside the lab and, upon entering the corridor's area, moves forth towards door 1 (refer to Fig. 1.1). The desired positions in the robot's path were determined as the corners of each corridor and initially placed 600mm from the wall on the left and 600mm before the wall in front.

Two controllers were used: one for rotations (*controller_orientation.m*), where linear velocity is set to 0 and which is used in moments when precise rotation of the robot, while maintaining its position, is necessary (*e.g.* when it turns 90°to face a door); this controller was necessary since, when moving towards a point (for which the *controller_translation.m* was used), we cannot define the orientation at the final position.

Additionally, two versions of each controller were available, due to limitations of MATLAB's function *atan2*, used for calculation of the β value as defined in Eq. 3.4: this function outputs an angle from $[-\pi, \pi]$ which, for situations where the robot is moving with an orientation oscillating in values close to π, makes a small variation in the angle of orientation to be calculated as a large difference in orientation. This is a source of instability for the controller, which is why, for corridors 3 and 4 (doors 12 to 21 in Fig. 1.1), where either the path or the rotation towards a door impose rotations around π, the controllers have embedded a threshold that will guarantee that no sudden changes in the angle calculated occur.

4.3 Localization Correction

The current position of the robot was determined using the position and orientation provided by the odometry but correcting these measurements with a correction matrix. The underlying principles for this strategy were based on the fact that, by correcting either x,y or θ in regards to the odometry, we are translating and/or rotating the global frame and, therefore, all the new (corrected) coordinates must be transformed accordingly. The transformation matrix for a translation of (x_k, y_k) and a rotation of θ_k is:

$$T_k = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x_k \\ \sin\theta & \cos\theta & 0 & y_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

File *get_T.m* serves this purpose. From this point on, given a proper correction matrix C, the new (corrected) position can be calculated:

$$\begin{aligned} T_{\text{corrected}} &= C \times T_{\text{odometry}} \\ C &= T_{\text{corrected}} \times T_{\text{odometry}}^{-1} \end{aligned} \tag{4.2}$$

where the corrected position and orientation, (x_c, y_c, θ_c), can be calculated from indexes (1,4),(2,4) and acos of index (1,1) of $T_{\text{corrected}}$, respectively. This is achieved using file *odo_get_localization.m* whenever a position update is required and the calculation of a new C, upon correction of measurements, is accomplished by calling *get_C.m.* Matrix C is updated in 2 different types of occasions along the path:

1. The robot's distance to the wall, as well as orientation in regards to the orientation of the wall, are used to correct θ_G and either x_G or y_G, depending on the present corridor (*e.g.* in corridor 1 the corrected coordinate is y_G but in corridor 2 it is x_G).

2. The robot's distance to a door whose coordinates are known corrects the remaining coordinate. The global coordinate corrected is consistent with the local frame, that is, we correct the global coordinate whose axis is pointing forwards or backwards in the local frame (*e.g.* in corridor 1 the corrected coordinate is x_G but in corridor 2 it is y_G).

This way, all the 3 parameters used to defined the robot's localization in each corridor are updated.

4.3.1 Correction of y_L and θ_L

In order to obtain the real coordinates corresponding to y_L and θ_L, four Lidar scans were used (this amount was scans was set as a threshold for consistency - to decrease the influence of outlier points in scans - and could be altered as it was one of the arguments of the function). After pre-processing the scans, using the already mentioned *LidarCorrection.m* file, an harmonic mean was carried out so that any existing outliers could be smoothed out (see file *HarmScans.m*). Based on that mean, the algorithm would only use the data coming from one of the robot's sides (defined by the variable *int_ori*: 1 to use the left side, 0 to use the right side), considering only angles included in the interval $\pi/4 \leq \theta \leq 3 \times \pi/4$ (or the symmetric interval, depending again on the value of *int_ori* used). After the transformation to the x-y space, the corresponding wall would be approximated through a linear regression in order to increase the consistency of the different measurements. Through the slope of that regression in the robot's frame, m_{wall}, the coordinate θ_L could be obtained through the expression: $\theta_L = -\arctan(m_{wall})$.

Regarding the y_L coordinate, it was obtained through the minimum distance to the corresponding wall. In fact, the coordinates y_L and θ_L are calculated in relation to the center of the corridor and to the axis along the axial direction of that corridor, respectively (see file *LidarCorrection.m*).

After this, a transformation from y_L to x_G or y_G and from θ_L to θ_G is carried out, taking into account the corridor in which the robot is. The corrected measurements would only be used for a new calculation of the C matrix if the difference between the then current y or$x$ and θ and the new values was large enough (1° for θ_G and 30mm for x_G or y_G.

Some limitations of this method are the requirement of a flat wall in the interval of angles considered, in order to ensure that the linear regression has a physical significance, and the time spent on this operation - 1 second for each Lidar measurement, plus the time spent on pre-processing those scans and carrying out the regression.

4.3.2 Correction of x_L

The correction of x_L was included in the door status process, being described in subsection 4.4.

4.4 Door Status Determination

The door status determination was carried out after the robot has turned itself to the assessing door, but not necessarily at the center of it. The code included the coordinates of all the doors in the global frame and, even though the goal point in each corridor was at the end of it, the robot was set to stop approximately in the middle of each door, after which it would turn its front to it, assess door status, and then turn back until it was agin oriented with the final point.

As an input for the determination, a Lidar scan, the door width, an integer that determines if the classification of the door should be done considering both sides of the wall surrounding the door or only one of them, the maximum distance parallel to the wall to be considered and finally a boolean that determines if a correction on the coordinates along the axis parallel to the wall (*i.e.*, x_L) should be made - because it was not in our interest to have it corrected on every door.

The approach used starts with a pre-processing stage according to the arguments used, including the transformation to the x-y space already described in previous Sections. After this step, the door is detected through peaks on the derivative of the data along the wall. After detecting the door, the set of points representing the wall or walls and the set of points representing the door are put apart. The points included in the wall set are subjected to a linear regression, which will be subtracted to all the points (both wall and door point sets), lining up the wall set points around zero. After that deduction, the maximum distance in the perpendicular direction to the wall is measured from the points in the wall set and is used to determine if the wall is open, semi-open or closed: if that distance is smaller than a threshold ("ombreira"), it is considered to be closed; if that distance is bigger than that value but smaller than "ombreira" + "door_width"/2 $\times \sin\alpha$, where α is a threshold angle (defined empirically as 45^o); for distances bigger than this value it is considered open.

Some results obtained using the function *LidarFrontProcessing.m* (the function responsible for all the processing just mentioned) are shown on Fig. 4.2.

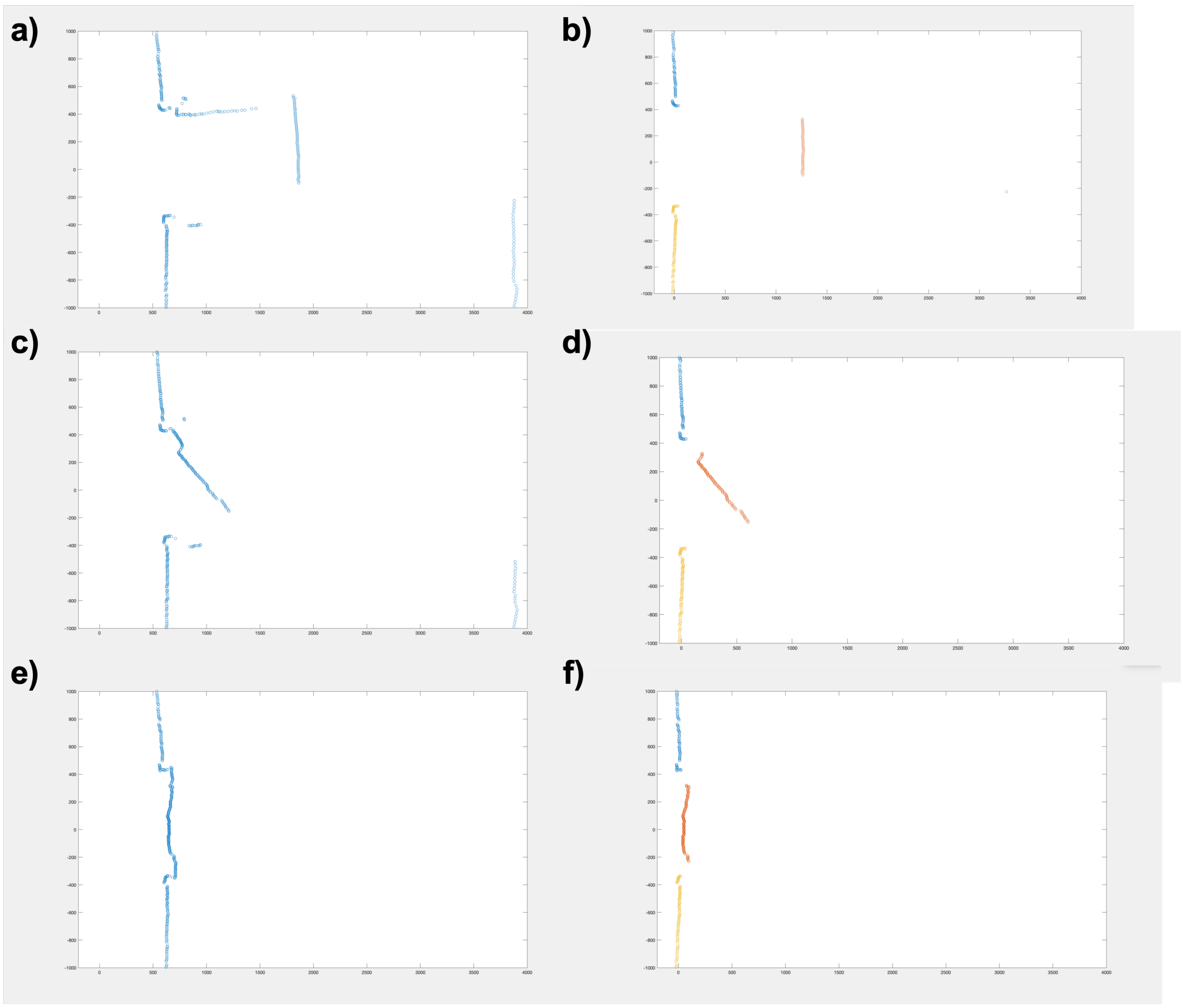

Figure 4.2: Results of LidarFrontProcessing: (a) Lidar scan obtained through the transformation to the x-y space for an open door; (b) Lidar scan obtained after processing the scan in (a), outputting an "open door" message; (c) Lidar scan obtained through the transformation to the x-y space for a semi-open door; (d) Lidar scan obtained after processing the scan in (c), outputting a "semi-open door" message; (e) Lidar scan obtained through the transformation to the x-y space for an closed door; (f) Lidar scan obtained after processing the scan in (e), outputting a "closed door" message; all the scans were obtained considering both the left and right wall, but as was mentioned we could have selected only one side to be processed.

Simultaneously, through the transformation to the x-y space, it is possible to determine the position (along the direction of the wall) in which the robot is in relation to the door. Therefore, it is possible to check and correct the coordinates of the robot along that direction (changes from corridor to corridor), taking into account the door coordinates. Similarly to what was described above, this correction was only applied if the difference calculated between current position and corrected position was "bigger" than 250mm (that is, if the robot was more advanced on the corridor than what was being measured, which means a difference positive or negative, depending on the corridor). Due to stability issues, however, these corrections were only carried out in doors 1, 7 and 19.

For more details about the process described in this section, see *LidarFrontProcessing.m.*

4.5 Object Detection

In order to detect approximation to an object, either moving or standing still, the robot's sonars were used. A timer was set to guarantee that the detection would not take place in intervals smaller than 300ms - otherwise we would compromise the performance of this function. When *sonar_detect.m* is called, it uses *pioneer_read_sonars.m* to get readings from the 8 sonars available. It selects the two sonars in the front of the robot (sonars 3 and 4) and, if the distance to an object on either of the sonars is detected as being smaller than 600mm, it stops the robot (the threshold was set to this value in order to ensure that the robot stopped before its distance to the object was 200mm, which would result in a reading of 5000mm of distance). Then, an audio file is played ("Sai da frente Guedes!"), signaling that an obstruction on the path was detected. The robot will not move until both sonars display distances bigger than 600mm.

5 Results and Discussion

The demonstration was carried out with robot No. 4 with the battery at the yellow/orange level. The results were the following:

- **1st attempt** - The robot performed well, even though it went too close to the left wall in corridor 2. However, while entering the laboratory room it deviated too much to the left again, hitting the cabinet.
- **2nd attempt** - The robot hit the left wall in corridor 2, having to be corrected manually. This time, the entrance on the laboratory was well performed, as well as all the other corridors.
- **3rd attempt** - The robot hit the left wall in corridor 2 and the cabinet on the entrance, having to be corrected manually in both cases. In the other corridors, there were no other relevant problems to report.

5.1 Debugging Performance

After the unexpected behaviours shown by the robot during the demonstration, namely the performance shown in corridor 2 and in the entrance, the workspace on MATLAB of the third attempt was saved, so that an *a posteriori* analysis could be carried out. Some of the results obtained on that run are shown on Fig. 5.1.

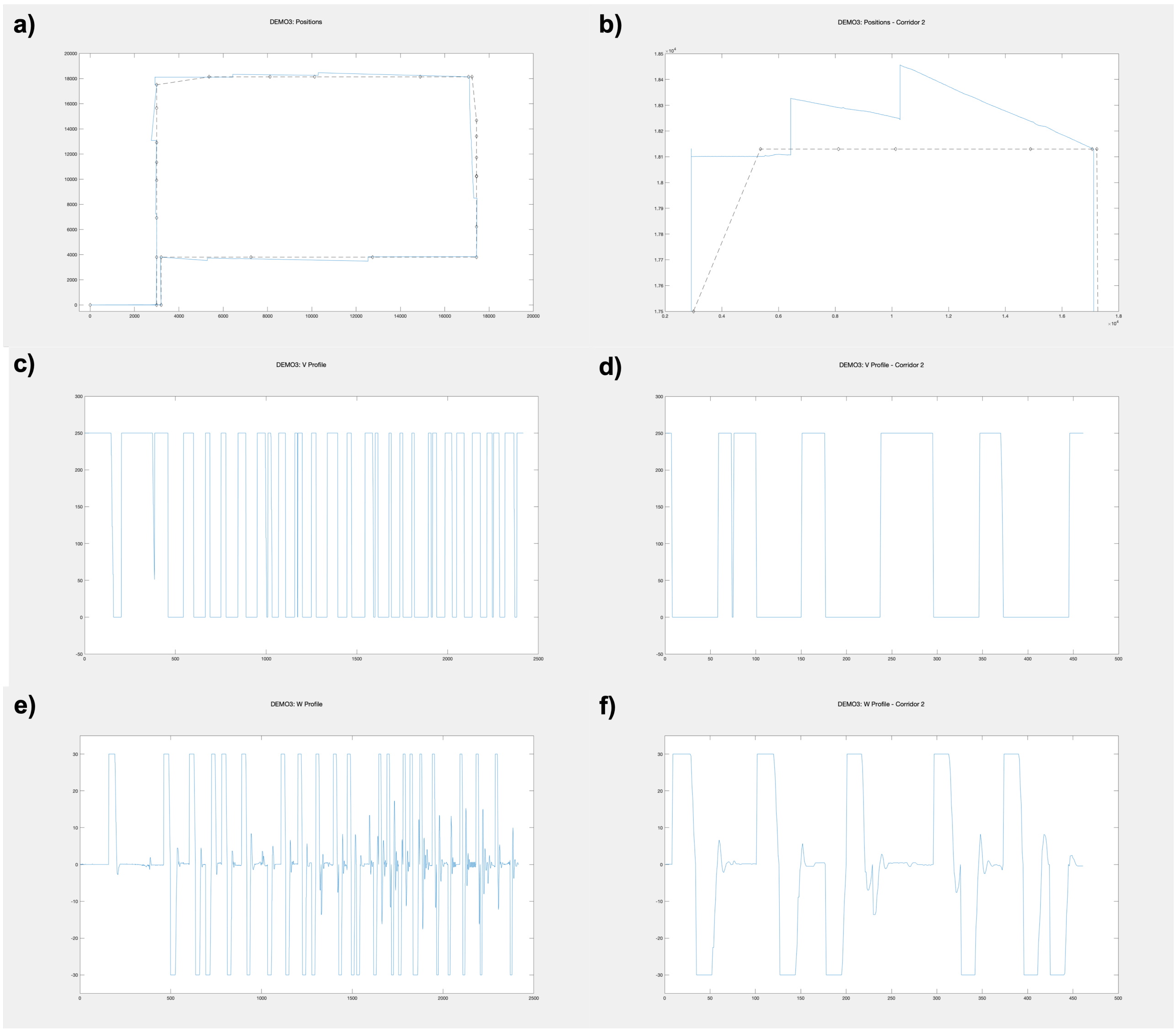

Figure 5.1: Results of the 3rd attempt: (a) positions occupied by the robot (blue line) and reference points (rhombus connected with dashed lines); (b) a) zoomed-in corridor 2; (c) profile of the linear velocity during the whole run; (d) profile of the linear velocity only in corridor 2; (e) profile of the angular velocity during the whole run; (f) profile of the angular velocity only in corridor 2.

In order to collect a video recording of the performance, the run was carried out with the same robot, No. 4, but with the battery at the green level. The results obtained are shown on Fig.5.2.

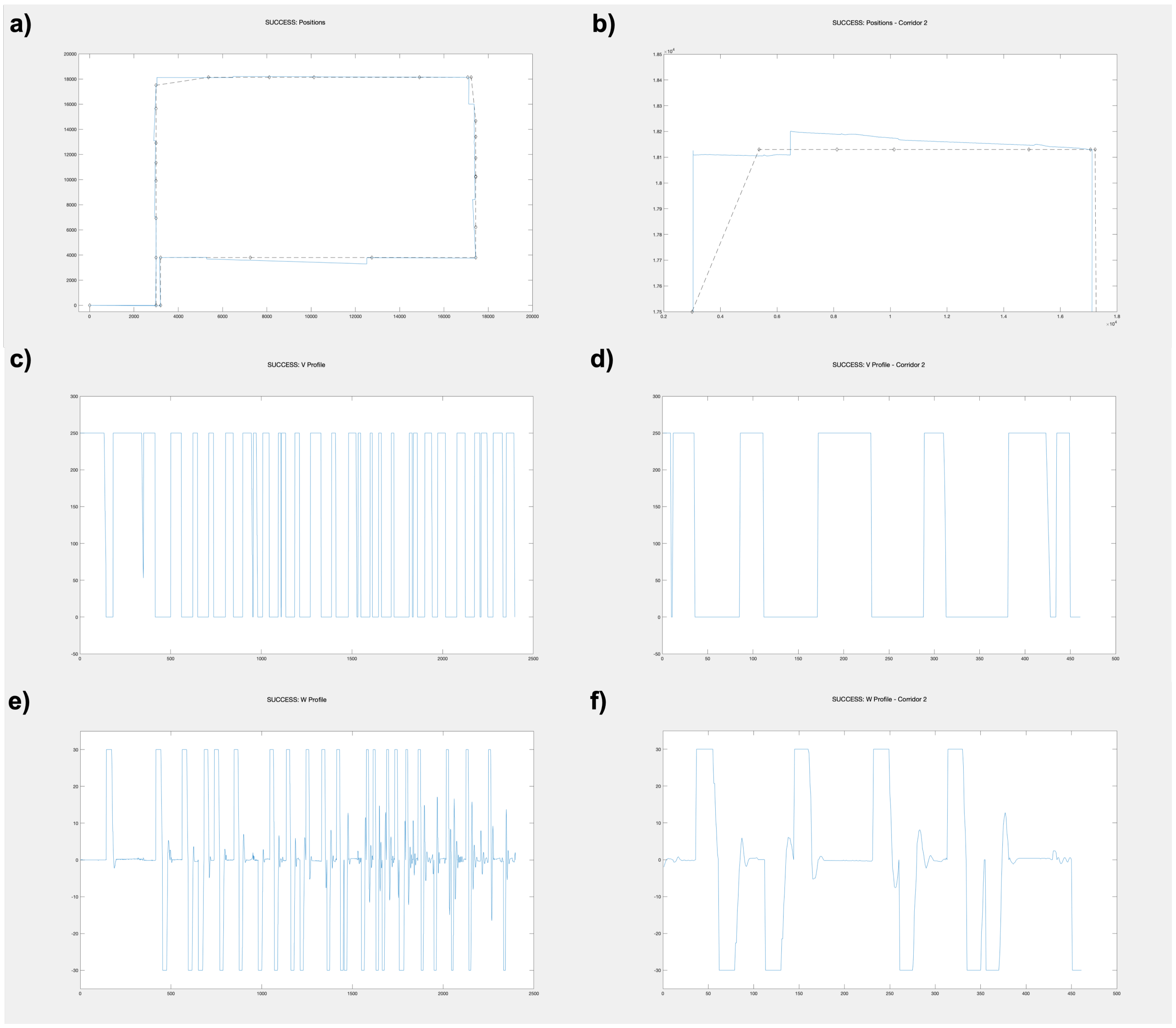

Figure 5.2: Results obtained during the video recording: (a) positions occupied by the robot (blue line) and reference points (rhombus connected with dashed lines); (b) a) zoomed-in corridor 2; (c) profile of the angular velocity during the whole run; (d) profile of the angular velocity only in corridor 2.

Before proceeding to the analysis of those results, three observations should be made: first, the sudden discontinuities that can be seen in the positions plots correspond to the points where corrections to the robot localization were carried out; then, the maximum angular velocity verified is a consequence of a limiter that was implemented in the controller; finally, in the transition from corridor 1 to corridor 2, the robot doesn't go directly to the next point, since that way it would collide with the benches present on that corridor. Thus, it goes to a point with a y_G similar to the one in the end of the corridor (goal point) and it only turns after reaching it.

Looking to the angular velocities profile of both Figures (5.1.e) and 5.2.e)), it's clear that the time that the robot takes until it stabilizes itself (*i.e*, has an angular velocity near or even zero) increases with the duration of the run, as well as the oscillations during those periods. This behaviour is a consequence of the increasingly slower velocity at which the computer is performing the different tasks as the run is getting heavier and heavier in memory resources, resulting in a smaller frequency of commands (namely, due to the plot obtained in Fig. 5.3, which was drawn in real time).

Regarding the performance during the 3rd run of the demonstration, as it was mentioned before, the performance of our robot was completely unrecognizable and we couldn't find any satisfactory reason to explain that behaviour. On Fig. 5.1.b), one can observe that the robot has a position correction on the y coordinate in the beginning of the corridor (6000 < x < 8000) and the robot finds that its coordinate is quite wrong. Consequently, it should behave like in the other corridors: after its correction, it adopts a new trajectory based on that new position and follows the correct path based on the correction of the odometry. However, in that run, in the next position correction (10000<x<12000), we understand that the robot didn't correct itself on the previous correction, since the deviation from the intended path is even bigger. Thus, the collision against the wall was shortly after this (non-)correction, constituting an extra evidence of the odd behaviour of the robot.

In comparison with the run of the video (Fig. 5.2), where the robot performed perfectly in terms of path, there's no significant difference, except in corridor 2. On that corridor, in the video run, after detecting its wrong y coordinate, the robot lines itself with the goal point (at the end of the corridor) and in the second correction there's almost no shift in that coordinate after the correction, indicating that the correction was correctly carried out.

An hypothesis for this odd behaviour in the 3rd run of the demonstration (that would also explain the odd behaviour on the previous runs of the demonstration) was that some coding error was leading to this different behaviour in corridor 2, but it wasn't observed neither on trials before the demonstration, nor in the trials after it (even when we tried to reproduce the low battery level). We point out that the code wasn't changed after the demonstration, and so, the version sent to the Professor immediately after it was exactly the same that we used for the trials represented on Fig. 5.2. Further hypotheses for the odd behaviour of the robot are extensively explored in Section 5.2.

5.2 Data collection

In order to try to replicate the behaviour observed during the demonstrations, as well as to collect more data that could help explain the performances observed, over the course of a day the robot followed its path, with decreasing amount of battery. The odometry and corrected position plot of 4 runs can be seen in Fig. 5.3.

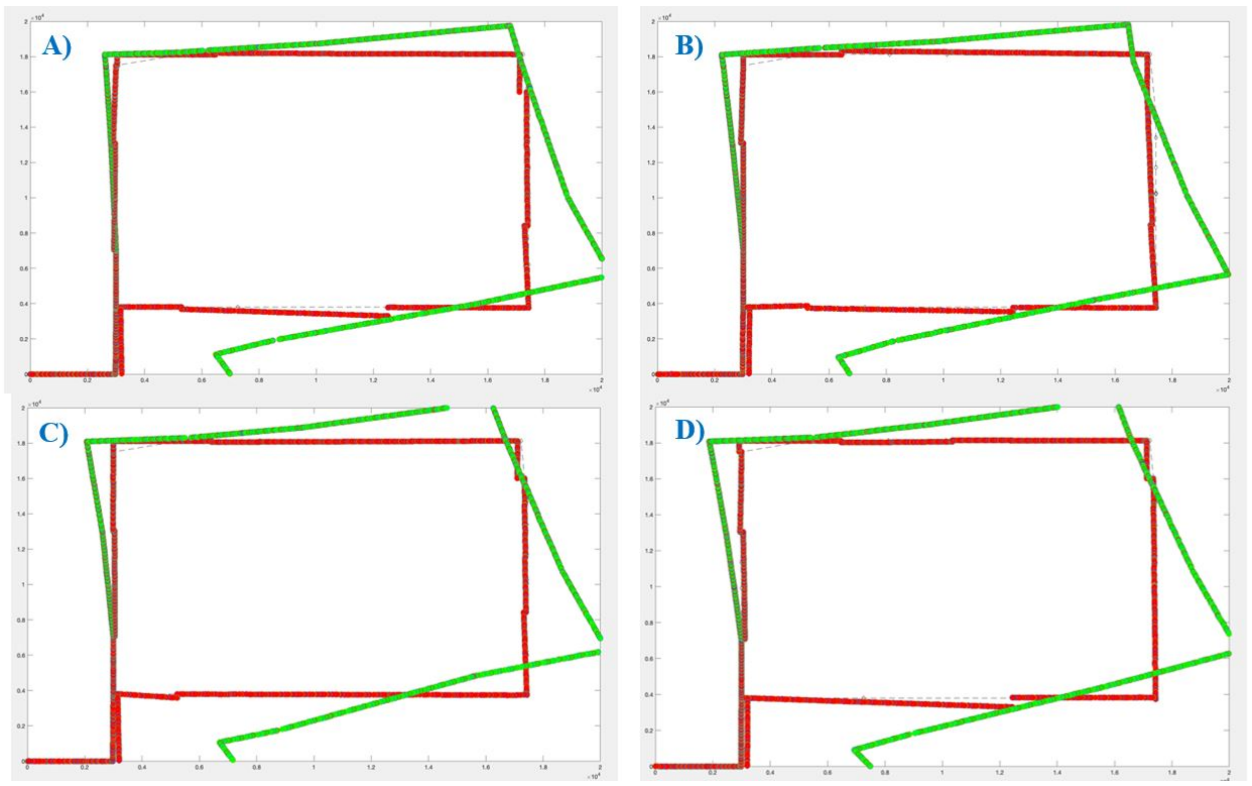

Figure 5.3: Map of path. Green points represent raw odometry readings. Red points are corresponding corrected positions of the robot. **A)**,**B)**, **C)** and **D)** are represented by order of of decreasing battery.

None of the runs represented resulted in crash of the robot on the 2nd corridor, but in the last (**D)** plot in Fig. 5.3) the robot crashed when entering the laboratory room again (cannot be seen due to axis limits). Furthermore, a progressive deviation of the odometry readings from the real positions can be observed (particularly obvious in corridor 1). Note that the strategy used for correction of position assumes that the error in the measurement is constant in between new calculations of the C matrix, which is not true; however, corrections were made with a frequency high enough for that assumption to not have a great influence in the performance of this localization correction strategy. The deviation starts from the moments when the first correction is applied (on the corridor 1, in front of the first door). The discontinuities in the red data points mark places where correction in the local y-axis occur.

Another phenomenon observed along the runs was the delay between sending the command via MATLAB and the execution of the respective control. For instance, we noticed that, when the robot was ordered to rotate only (*i.e.* linear velocity of 0, angular velocity $\neq 0$ but a very small value), a command to stop this rotation would not be executed immediately, with an increasing delay (consistent with the increasing oscillations observed in Figs.5.2 and 5.1). In the cases where a precise rotation was desired, *e.g.* a $(\pi/2)$rad rotation, the robot would actually rotate more than that. This could have a cumulative effect in deviation from odometry localization, but the fact that this behaviour was not always observed makes this hypothesis insufficient to justify the large scale deviations observed. Moreover, the local θ correction is able to solve this cumulative effect.

Even though it is also not a sufficient explanation by itself, the decrease of battery is evidently shown here to have effects either in odometry measurements or in the amount of lateral offset observed in a forward motion (in this case, the robot very clear deviates to the left). Exactly how that should affect performance, is not clear; the first hypothesis, that of incorrect odometry readings, seems unlikely since it relies on optical encoders which calculates position from effective rotation of wheels and even considering the limitations on these reading already mentioned (different ratio of wheels, etc.) there is no apparent reason as to why lack of battery would progressively worsen this measurement. On the other hand, the second explanation, that of there being, effectively, bigger deviations as the battery level goes down, seems more reasonable: there is less power conveyed to each of the motors, which means that the controls sent to the robot would not be executed fully (for instance, the angular velocity set is not achieved). This could explain the deviation if the motors of each wheel are differently affected by the lack of battery, but also if they are equally affected but somehow the decrease in power enhances inherent sources of deviation (such as different ratio in wheels, etc).

Furthermore, this last hypothesis would justify the reason as to why the robot, during the demonstrations, crashed against the walls - the commands sent to correct the calculated deviations were not successfully executed by the robot, or not fast enough, which would be sufficient to cause the crash of a robot whose cumulative effect of deviations caused it to move very close to the wall. However, given that a crash on corridor 2 was not observed in any of the runs executed after the demonstrations, even given a very low amount of battery left, the only factor left that could back up this hypothesis would be the constant utilization to which the robot was subjected on the day of the demonstration, since it had been running for many hours straight.

The hypotheses raised in this Section cannot, however, be considered comprehensive enough to explain the many differences observed in between runs. Overall, any run is subjected to large uncertainty, which a robust control computation can minimize to a certain point; even the position in which the robot is initiated, which defines the origin and orientation of the global frame, can induce a different behaviour (our code, however, corrects all the three variables, thus reducing the influence of this factor). In any case, we must unfortunately leave the behaviour observed during the demonstrations partially unexplained.

5.3 Limitations and Improvements

The execution of this assignment carried limitations inherent to the conditions provided for execution. The large number of groups and reduced number of robots (some of them with very bad performance) and Lidars was definitely a cause for delayed progress, which lead to a less precise refinement of certain aspects of this assignment. Moreover, the constraints relatively to the schedule during which the Lidars were available caused an extremely high affluence on those periods and severely hindered the already mentioned progress.

Furthermore, variations in the behaviour of the robot within each repetition of the path conditioned how certain tasks were computed: for instance, initially the proximity of the robot to each door would be subjected to great variations, which made it hard to compute a function that could properly display the door status in spite of the robot's localization. This, however, lead to progressively more robust methods for classification.

Regarding the potential improvements that we could consider to our approach, we can point out the time that our robot takes to complete a full run (10 min). Even though it doesn't have particularly slow linear or angular velocities along the path, our approach spends lots of time on y_l and θ_l correction (waiting for different Lidar scans, processing that information, etc.). This point could be significantly improved by allowing the robot to do this correction without stopping its motion and relying on less Lidar scans, though this option opens the door for a less conservative strategy and, consequently, more risks.

Besides, despite already having a quite robust door detection algorithm, it could be enhanced by using a bit more complex steps than linear regressions. Hough Transform seems like the perfect applicant for this task.

6 Conclusion

The present assignment serves as testament to the gap between predicted and/or theoretical response of a system and that in a real world context; in particular in the Robotics field, where the main sources of uncertainty are not only the surroundings and the way they interact with the robotic device but the robot itself. Therefore, the strategy presented here suffered many modifications, with the purpose of increasing not only the performance of the robot but also its robustness. The reformulations of methods used, as well as the proper integration of each component and tack into the main code which would control the robot's behaviour, were of high pedagogical value.

The approach presented has its limitations, and the formulation is rather complex - each corridor was very personalized, resulting in an extensive code. However, and even though a more simple approach could have been adopted, there could be a trade-off between simplicity and robustness, thus making performance rely too much on "luck". In order to reduce this risk, we opted with covering more possible outcomes in the code, thus trying to decrease instability. Furthermore and almost paradoxically, we noticed that more information doesn't necessarily mean a better performance, since it can lead to increasing instability due to contradictory measurements.

Apart from the demonstrations, which have been discussed, with the current code the robot exhibited a good performance, with automated motion through all the corridors as well as during the entrance and exit of the laboratory room. Trying to troubleshoot the robot's bad performances is not an easy task but it did get us to deepen some of the concepts and knowledge that had previously not been as explored.

Automated motion remains a challenge due to the unpredictable nature of the conditions on which the robots are dependent; however, with the increase in computational resources, data handling mechanisms and machine learning, automation will certainly play a big role in shaping the technology of the future.

References

[1] J. M. Sandeep Kumar Malu, "Kinematics, localization and control of differential drive mobile robot," *Global Journal of Research In Engineering*, 2014.