Instituto Superior Técnico

Masters in
Electrical and Computer Engineering

Robotics

2nd Semester 2018/2019

Basic Navigation Strategies for Mobile Robots - Pioneer DT

84037 – Eduardo Costa
84038 – Eduardo Melo
84087 – João Sebastião

Docente: João Sequeira

June 7th 2019

Contents

1 Program Structure

The project is named 'ProjectPioneer', and is divided into several components with specific and smaller objectives that, when combined, lead to the execution of the main objective: make the robot navigate around the north tower 5th floor, detect doors and determine whether they are fully open, closed or half open. The structure of the program is presented below in figure 1.

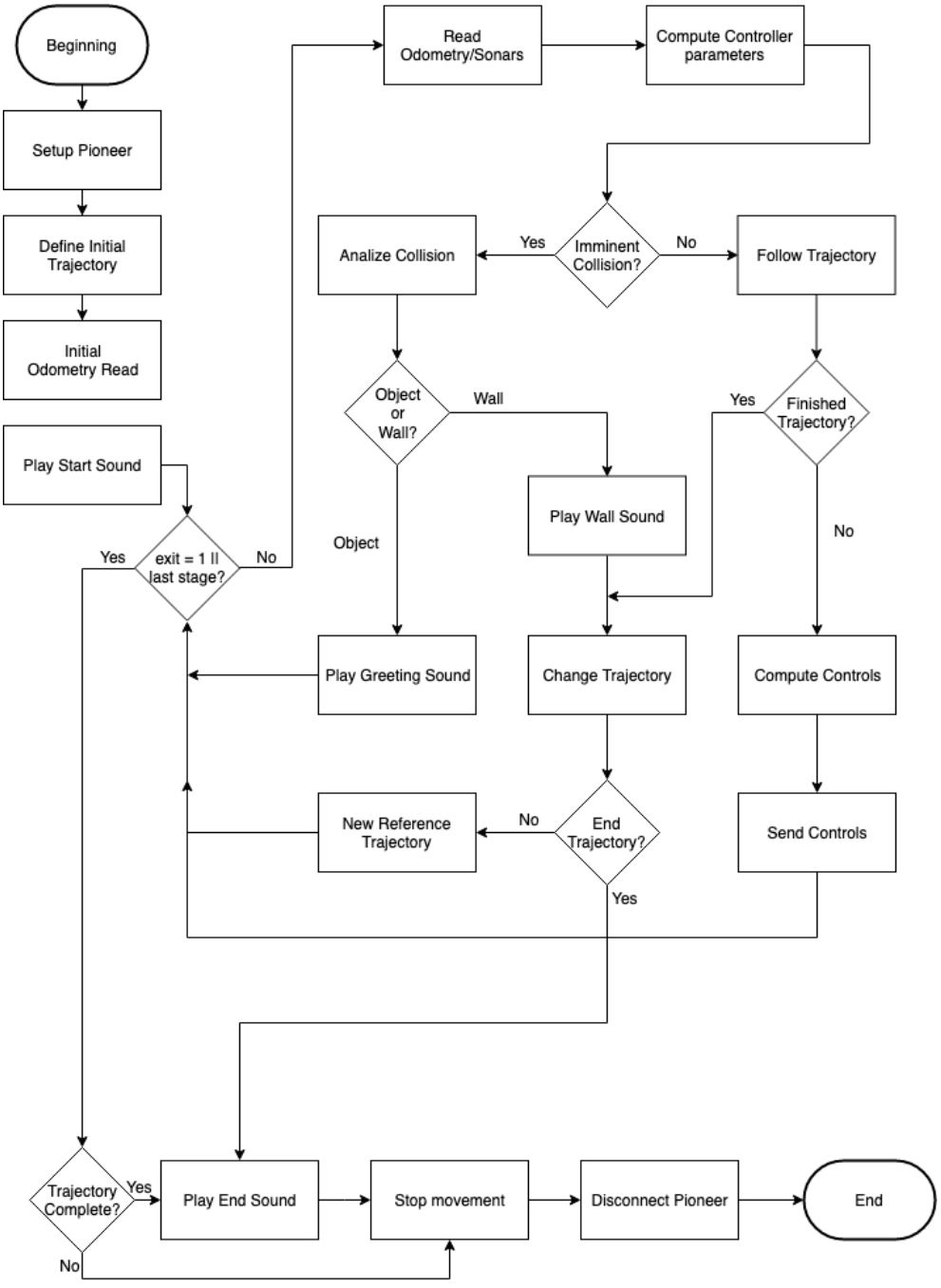

Figure 1: Structure of the program.

2 Navigation

In order to make the robot navigate around the 5th floor, a combination of odometry, the sonars and the laser was used. Under the assumption that not all robots are equal, both the odometry data and the sonar data will be different in each robot and, therefore, in order to maintain consistency and robustness, it is not possible to rely entirely on one or the other.

For this reason, it was decided to use the combination of all 3. The odometry was used to calculate the parameters to track the position of the robot and plot the trajectory that it must follow. The sonars were used to detect obstacles that may appear in the middle of the trajectory of the robot. The laser was used to support the trajectory by aligning the robot's trajectory with the walls, support the sonars in identifying obstacles that can be either people walking or objects placed in front of it or walls and, at last, determine the doors' state, whether they're fully open, closed or half open.

The entire route is divided into 6 sections. Each section is divided into several stages. These stages correspond to trajectories between doors or between doors and checkpoints. Checkpoints are defined points in the trajectory in which the robot corrects it's alignment. Every trajectory can end in one of 3 possible scenarios: a checkpoint, a door or a wall. In figure 2 there is a representation of the 5th floor including the 6 different sections, the desired route, every door and the checkpoints marked with an X.

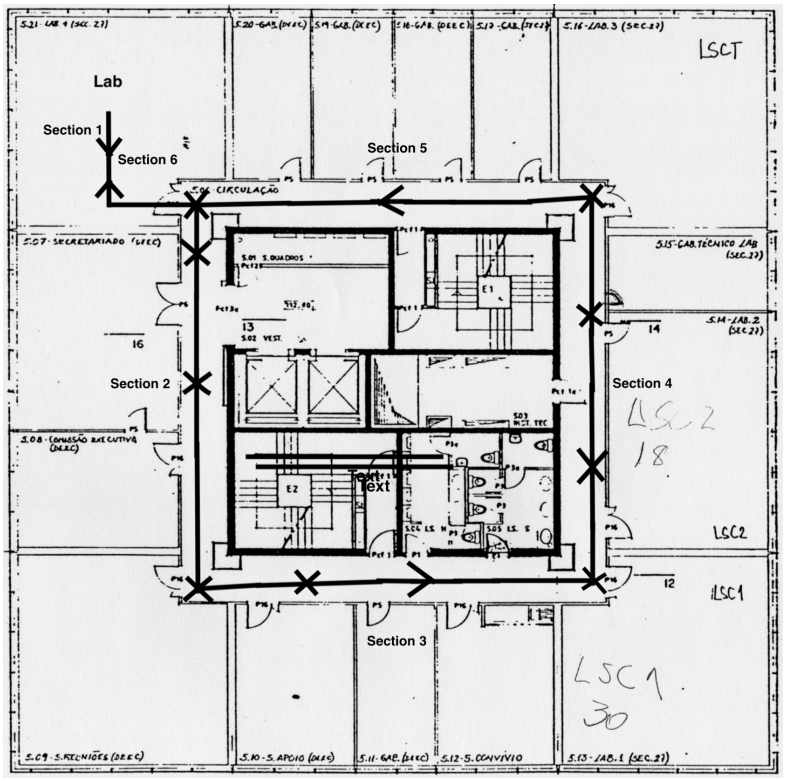

Figure 2: Map of 5th floor with checkpoints.

When the current trajectory ends in a checkpoint, there is always at least one wall, either left or right wall, to provide alignment. This alignment takes into consideration the distance to the next reference point and the distance to the wall that it's being used.

When the current trajectory ends in a door, after the procedure to analyze the door is finished, if there is a clear wall on the other side, the same procedure as before is executed. If there isn't, in the case there are benches, a checkpoint is located a small distance away where that procedure can be applied.

Finally, when the current trajectory ends in a wall, the current section ends and a new begins (except for the first and final section that involve exiting and entering the lab, respectively). Before starting the new section, 3 steps must be taken. The first step is to fix the robot's alignment with the support wall in order to be approximately parallel to the wall. The second step is to measure the distance of the robot to the wall of the previous section. Since the distance between every door and checkpoint is fixed but the point at which the robot starts the new section isn't, it is necessary to correct the distance to the first stopping point to make sure that the following stopping points are correct. The third and final step is to measure the distance to the supporting wall of the current section to maintain the intended distance to that wall throughout the new section.

3 Control Strategy

To track the position of the robot, we used the geometric representation presented in figure 3

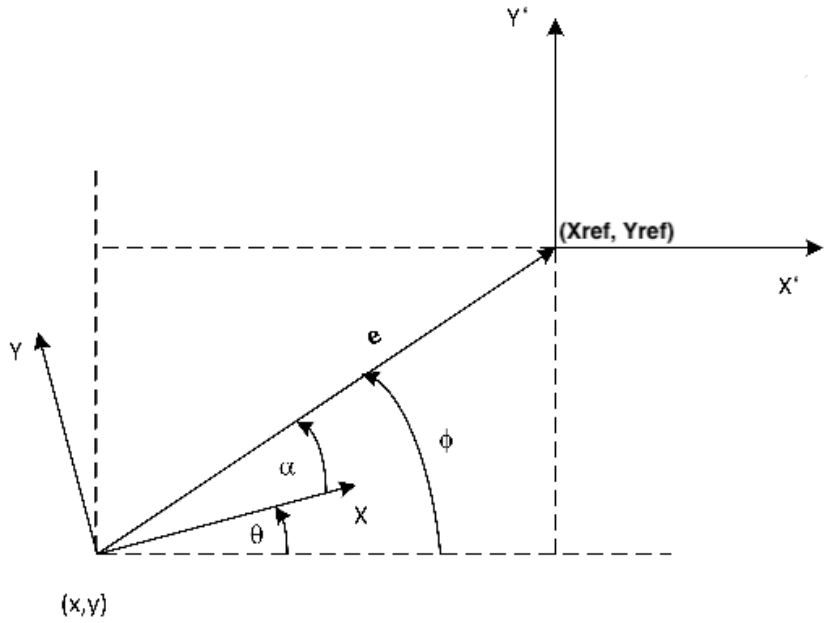

Figure 3: Geometric Representation.

where

$$\begin{aligned} e &= \|\mathbf{e}\| = \sqrt{(X_{ref} - x)^2 + (Y_{ref} - y)^2} \\ \phi &= \tan^{-1}((Y_{ref} - y)/(X_{ref} - x)) \\ \alpha &= \phi - \theta \end{aligned} \quad . \tag{1}$$

The control law used was the following:

$$\begin{cases} v = v_{\max} \tanh\left(K_1 e\right) & (2) \\ \omega = v_{\max} \left(\left(1 + K_2 \dfrac{\phi}{\alpha}\right) \dfrac{\tanh\left(K_1 e\right)}{e} \sin(\alpha) + K_3 \tanh(\alpha) \right), K_1, K_3 > 0, & (3) \end{cases}$$

with $K_1 = 2, K_2 = 10, K_3 = 40$. Using the Lyapunov function

$$V \equiv \frac{1}{2} e^2 + \frac{1}{2} \left(K_2 \phi^2 + \alpha^2\right), \text{ with } K_2 > 0, \tag{4}$$

it is assymptotically stable at $e = 0$.

4 Door Detection

When the trajectory ends in front of a door, the robot rotates towards it and then evaluates the door's state. To do that, the laser scans the door region and the 150 points that are closer to the front of the robot are evaluated.

The points are sequentially separated in groups of 3 and the groups are sorted. This gives us 50 groups, and we will only use the median value of each group forwards, so that outlier points are discarded. Then, the distance of all the medians will be measured.

The decision of whether the door is closed or half open is based on two distinct thresholds. The distance to all of the points is computed and if it is below the minimum distance or above the maximum distance, other two pre-defined thresholds, a counter variable increments. If the counter is below 3, then the door is considered closed. If the counter is between 3 and 24, the door is considered half-open, otherwise it is considered fully open.

These thresholds were chosen after several experiments of testing and could be further tweaked, depending on the level of precision it is required but, for this purpose, we considered the results obtained to be more than satisfactory, as shown in figures 4, 5 and 6.

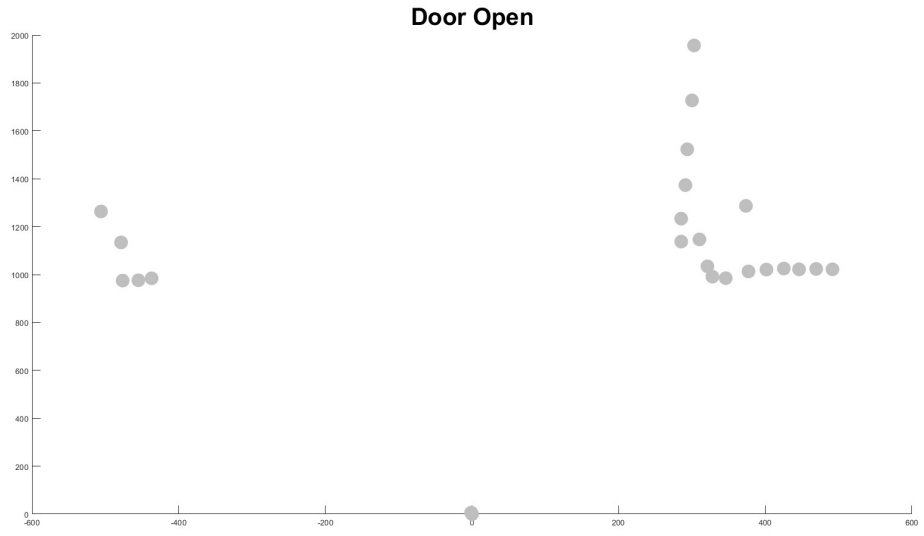

Figure 4: Laser reading of an open door.

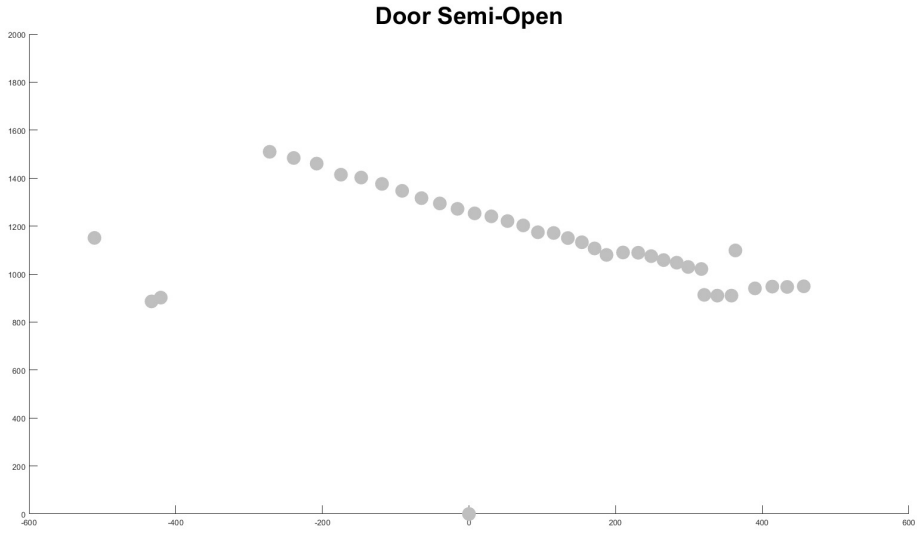

Figure 5: Laser reading of a semi-open door.

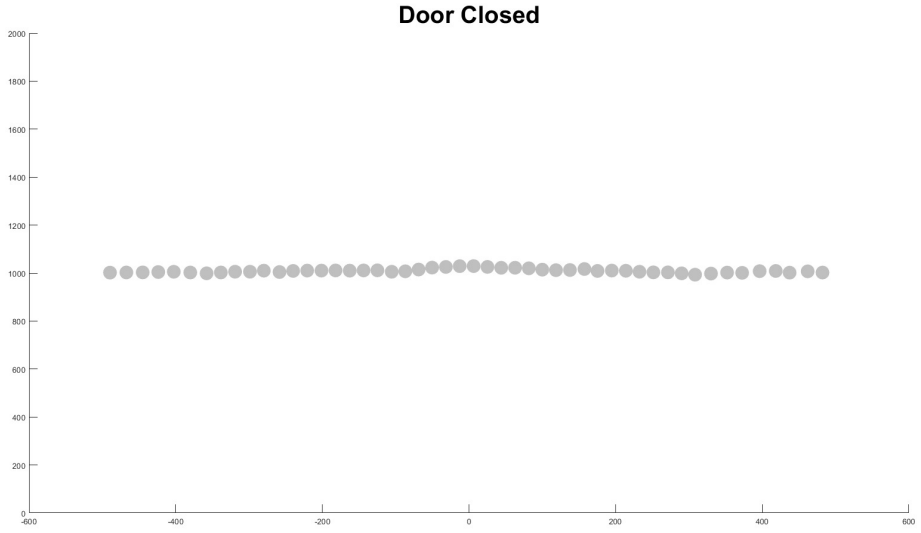

Figure 6: Laser reading of a closed door.

As shown in figure 4, some of the points aren't plotted, and the function classifies the door status as 'open'.

In figure 5 some points aren't plotted, but not as much as in the previous case, and because of that, the door status was classified as 'semiopen'.

Finally, in figure 6, all of the points are plotted, meaning that the door is classified as 'closed'.

5 Collision Detection

Collision detection is performed by the front sonars of the robot and is complemented by the laser to determine what type of collision is detected. Collision is only performed while the running flag is set to 1 (that is if the robot movement is not halted by the user) and is detected if the sonars provide a measured distance that is smaller than a pre-defined collision range and different than 0, to prevent bad readings from the sonars that can affect the program execution.

The types of collision that may be detected depend on the section, and stage, in which the robot is located. Some trajectories between stages don't include walls in their straight path and, therefore, any collision that is detected is due to an obstacle passing and/or placed in front of the robot. In this case, the robot's movement is halted and a greeting sound is played and the program cycle continues.

When the trajectories do include walls, it is necessary to determine whether the collision detected is an object or a wall, which, as shown in figure 1, has different procedures to execute. Following the flow chart, after the detection by the sonars, the laser readings are used to analyze the collision. The laser readings go through some data processing to try to minimize the possibility of errors, which will be explained at a latter stage.

In case it is an object, like in the situation before, the robot's movement is halted and a greeting sound is played and the program cycle continues. If it is a wall however, a sound is played indicating the presence of a wall and, since each corridor path ends in a wall, the trajectory is changed and a new section begins.

To evaluate how well the laser would perform in obstacle detection, 3 tests were conducted. The blue X marked in each figure represents the location of the laser, and consequently the robot. First, we wanted to see how well the laser detected walls. The result obtained is shown in figure 7.

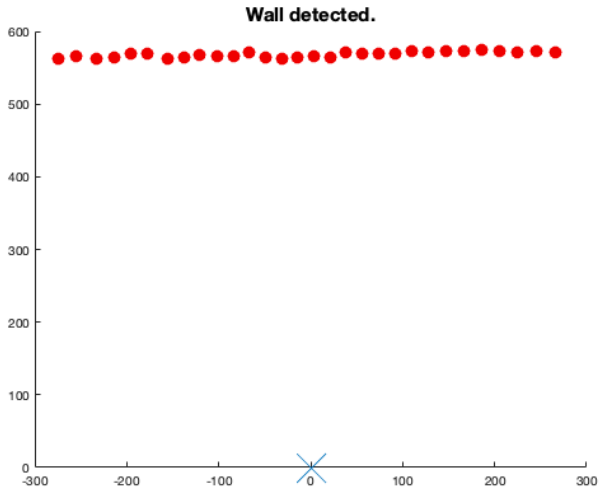

Figure 7: Detection of a wall.

By observation, the group of points shown indicates that there is a "flat" surface at approximately 589.43 mm, which indicates the presence of a wall. It is important to note that the points are not exactly at that distance, some are closer and some are further away, indicating that although reasonably accurate and even assuming that the wall is a perfectly flat surface, there are some errors to take into consideration when defining thresholds for object detection.

The next test was to evaluate how well the laser would detect a person with the wall as background. The result obtained is shown in figure 8.

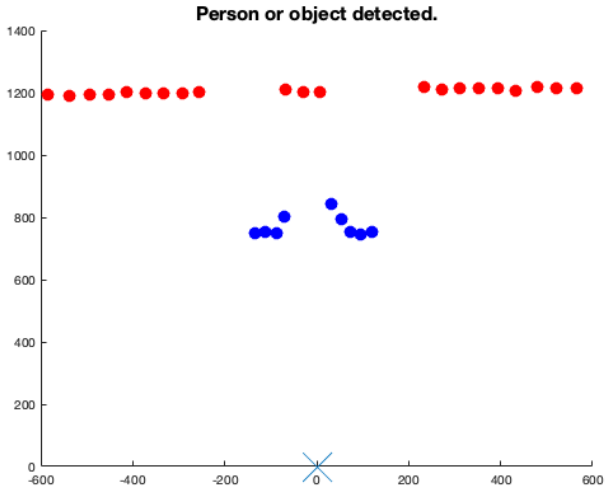

Figure 8: Detection of a person in front of a wall.

As shown, the laser is able to detect the person's presence extremely well, differentiating it from the wall. This test is particularly important because, as shown in figure 1, when a wall is detected, the trajectory is changed and if the person is not detected properly, the robot would change trajectory too soon and would most likely compromise the following trajectories.

The final test was to evaluate how well the laser would detect a person somewhere along the corridor. This situation is the most likely to happen considering the size of the corridors and the number of doors in the floor from which people can enter/exit and it is also the most important test, considering how the program was designed, shown in figure 1. The result obtained is shown in figure 9.

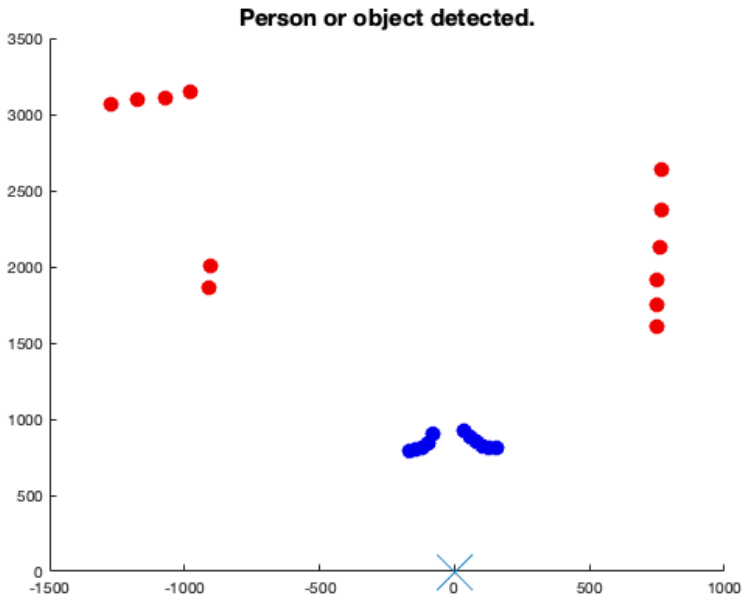

Figure 9: Detection of a person in the middle of the corridor.

By direct analysis, the laser is again able to detect the person's presence extremely well while detecting a wall on the right and 2 open doors on the left separated by a small fraction of wall between them. The first door does not appear because the door opens to the left side and to perform this test, we didn't consider the entire visibility of the laser, and the second door appears to be fully open as the "imaginary line" connecting the left points is almost perpendicular to the right wall.

From these tests, we conclude that the laser provides a good and clear way to identify the different types of objects that can cause collisions. It also shows that it can be used to align the robot's trajectory with the wall as the wall points can define a reasonably well defined line of support for orientation. Finally, it also showed good indications of being able to determine the door's state fairly well. For these reasons, the laser was used in these 3 situations, adding more robustness and error resistance to the program execution.

6 Software Structure

This section is dedicated to the functions created and used throughout the program, a brief explanation of what their purpose is and how they achieve that purpose, and also some variables that play an important role in the program.

6.1 Variables

Some variables were created that are used throughout the program. Some are fixed variables, which means they are not changed since the moment they are created, some are program flags that are activated/deactivated and others are just "regular" variables. Some of these were made global to simplify their usage throughout the program.

6.1.1 Fixed Variables

First, the fixed variables that consist of either maximum/minimum values or predefined distances. These are:

- V_{max}: Maximum linear velocity;
- V_{min}: Minimum linear velocity;
- W_{max}: Maximum angular velocity;
- $DefaultRange$: Default range for collision detection;
- $CollisionRange$: Current range for collision detection ($\leq DefaultRange$);
- $leftDistance$: Distance that the robot should maintain from the left wall;
- $rightDistance$: Distance that the robot should maintain from the right wall;

6.1.2 Program Flags

Next, the program flags that either control the program execution by user commands, allow sounds to be played or participate in the collision detection. These are:

- $RunningFlag$: Determines whether the program is running (1) or paused (0);
- $SoundFlag$: Determines whether a sound can be played (1) or not (0);
- $ContinueFlag$: Activated with collision detection, program executes the next loop iteration without sending prior movement commands;

- $WallFlag$: Also used with collision detection, has 3 possible values:
  - 0: There is no wall in the current stage so all collisions are objects;
  - 1: There is a wall in the current stage, required to identify type of collision;
  - 2: When an obstacle/wall is detected, decrease to minimum velocity;

6.1.3 "Regular" Variables

Finally, the "regular" variables are related to velocities, angles, system description or trajectories. These are:

- K_1, K_2, K_3: Controller gains;
- ϵ: Error tolerance between robot's position and reference point;
- *stage*: Stage of the track in which the robot is located;
- $FinishTraj$: Determines if the current trajectory is finished (1) or not (0);
- V: Robot's current linear velocity in $meter/second$;
- W: Robot's current angular velocity in $degree/second$;
- $intV$: Convert linear velocity to 32-bit value to send to the robot;
- $intW$: Convert angular velocity to 32-bit value to send to the robot;
- $fixAngle$: Angle used to adjust trajectories;

6.2 Functions

6.2.1 PlotInit

This function creates a new figure that will plot the trajectory performed by the robot and that includes two buttons, identified by *Exit* and *Pause/Resume*, that when pressed call a specific function to execute (callback function).

6.2.2 ExitProg

This callback function stops the robot movement, closes the connection with the robot and sets the exit flag to 1 to exit the program.

6.2.3 PauseProg

This callback function analyzes the running flag and if it's set to 1, it play a "paused" sound, stop the robot movement and set the flag to 0. If the running flag is set to 0, it will again play the "paused" sound and send the movement controls to the robot to proceed.

6.2.4 Change Trajectory

This function is responsible for the change in the trajectory for the robot to follow. As shown in figure 2, the entire route is divided into 6 sections, which are:

- Section 1: Trajectory from inside to outside the lab;
- Section 2: First corridor on the right of the lab;
- Section 3: Second corridor;
- Section 4: Third corridor;
- Section 5: Forth and final corridor;
- Section 6: Trajectory from outside to the starting point inside the lab.

Door detection is also performed in this function to analyze whether doors are fully open, half open or closed.

Depending on the section where the robot is located, different functions can be called. These functions are named *DefineTraj*, *Rotation*, *GetOrientation*, *FixOrientation*, *getAngle*, *getDist* and *GetDoorStatus*.

6.2.5 DefineTraj

This function receives the destination point and creates the trajectory from it's current position (position read by the odometry) to that point and prints the trajectory onto the figure created initially.

6.2.6 Rotation

This function receives the direction of rotation, reads the odometry and sends control to the robot to rotate in that direction until the full rotation is performed. This rotation has no linear velocity, only angular velocity.

6.2.7 GetOrientation

This function receives the side (left or right wall) that will be used to fix the robot's orientation. With the use of the *Lidar*, a collection of points is selected from the readings of the laser. Using this set of points, the distance to the wall is retrieved by ordering the array of points and choosing the median value, and then the linear

regression of that set of points is computed, providing the angle between the robot and the wall.

6.2.8 FixOrientation

Using the information provided by the *GetOrientation* function, this function is used to send the robot controls to performs a rotation to compensate the angle between the robot and the wall until it's aproximately parallel with the wall.

6.2.9 getAngle

This function receives the direction, in degrees, and computes the angle between the odometry read and that direction.

6.2.10 getDist

This function computes the distance from the robot to the surface in front of it using the laser readings.

6.2.11 GetDoorStatus

This function analyzes the laser readings and determines whether the door is fully open, semi-open or closed. This determination is explained in section 4, and when it's done, the corresponding sound is played, and the door state is printed on top of the figure.

6.2.12 IdEvent

This function uses the laser readings to determine if the collision detected is an object or a wall. For this, the distance points provided by the laser are grouped into groups of 5 and then sorted in an array. If the discrepancy between the measured minimum value and the median value is above a certain threshold, it is considered to be an object, otherwise it is considered to be a wall.

7 Performance Analysis

The main priority was to make the robot follow the trajectory correctly while maintaining a good distance from the walls and detecting doors and collisions correctly. To achieve this, we set the maximum velocity to 20 mm/sec and defined checkpoints between doors to fix the robot's orientation. Thus, by compromising on speed and efficiency, we improved reliability and good trajectory planning.

However, our solution can have some problems. By having more stopping points, the trajectories will be smaller and the robot might not be able to completely fix the trajectory on time, missing the destination point. Also, under the assumption that the walls are completely flat surfaces, when analyzing a more irregular surface with the laser, the combination of the irregularities and possible errors from the laser readings can hinder the robot's orientation and trajectory, possibly causing it to crash against objects.

Regarding the demonstration, some conclusions can be drawn. The first and second runs were unsuccessful but the third and last run was successful. In the first and second run, the robot crashed once against the wall and then against one of the benches but in the same corridor, corresponding to section 4 shown in figure 2, which we considered the hardest section of the entire route.

From what we could observe, the cause that led to this unsuccessful run happened between sections 3 and 4 and we believe the possible causes for this are either one of the mentioned above or the combination of the two. For example, by having a bad reading of the wall, the trajectory for the robot to follow wasn't ideal and by having multiple stopping points, the robot wasn't able to adjust it's trajectory properly, causing the collision.

It is important to note that, despite the unsuccessful runs, the robot properly handled obstacles, by ceasing movement until the obstacle was removed, and also detected all doors and correctly analyzed the doors' state until the stopping point of the unsuccessful runs and until the end of the route in the successful run.