TÉCNICO LISBOA

Robotics 2014/201

.

2nd *Assignment*

Strategies for planning and following of paths by mobile robots

Alexandre Laborde
79448

Pedro Cristóvão
70273

March 3, 2018

1 Problem Description

The goal of this assignment is to create an algorithm that allows a Pioneer P3-DX research robot to leave a room passing trough a door, move around the 5^{th} floor of Instituto Superior Técnico's North Tower following a given trajectory and return to the same room. In this assignment, the robot must be able to correct its behavior autonomously in order to respond to unmodeled difficulties such as receiving wrong odometry data or noisy sonar information.

The remainder of this document is divided as follows. In section 2 we start by reviewing all concepts required to implement our proposed solution and define the notation used throughout the document. After, in Section 3 we review the basic ideas and algorithm behind our solution. In Section 4 we discuss how the robot's trajectory was generated from the building's blueprints. The details of our controller's design and implementation are explained in Section 5. After, in Section 6 we detail how our algorithm coupes with erroneous information in order to improve its performance. The results of our algorithm are shown in Section 7, and finally we conclude this document in Section 8 by reviewing all lessons learned during the development of this assignment.

2 Conventions

In this assignment, we used Pioneer P3-DX research robot whom dimensions where obtained from the manufacturer's web page. In order to remove complexity from our simulation we simplified the robot's shape based on the original robot. We maintained the same dimensions and measured point in which the robot's rotates around (see Figure 1).

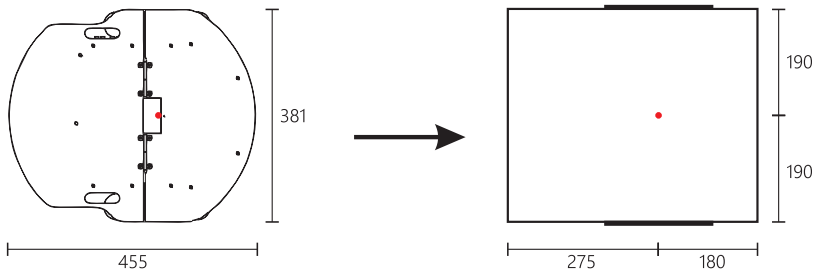

Figure 1: Simplification of the robot. The red dot represents the robot's rotation point.

This robot has two frontal parallel driven wheels and a free wheel to keep balance, it cannot do pure sideways translations thus being a non-holonomic robot. This type of robot architecture is called unicycle robot.

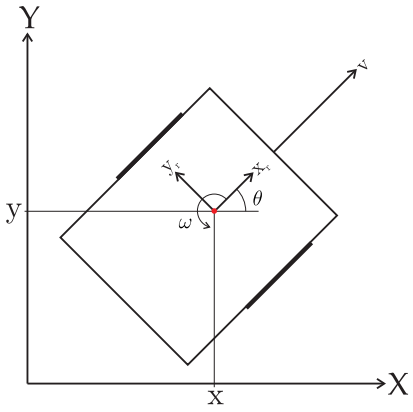

Figure 2: Unicycle's dynamic model. x_r and y_r represent the robot's coordinate frame.

In the unicycle robot dynamic model (see Figure 2), the control variable $\mathbf{u}$ is a 2×1 vector is composed by a linear velocity v and angular velocity ω. The control $\mathbf{u}$ relates to the robot's state $\mathbf{q}$ by,

$$\dot{\mathbf{q}} = M(\mathbf{q})\mathbf{u} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1}$$

In this assignment, the robot has to maneuver inside a building. Based on the building's blueprints we created a partial model of the 5^{th} floor. Its dimensions in meters can be found in Figure 3 where the central area represents the elevator shaft thus being impassable terrain.

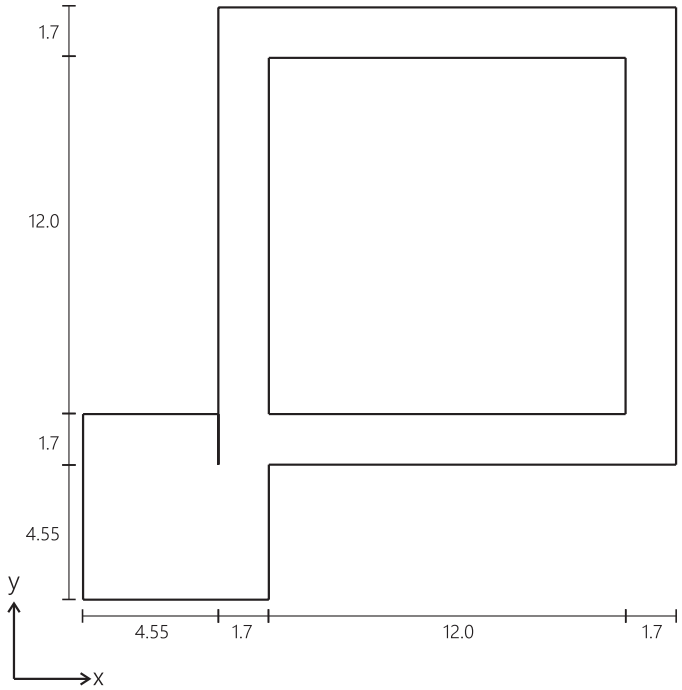

Figure 3: Partial map from the North Tower's 5th floor. Dimensions in meters.

Since most images in this document where obtained from our simulator, we should also define the meaning of the representation used in our simulation environment. As is shown in Figure 4, the red rectangle represents the robot, the green and cyan lines the robot's x and y axis respectively and the black line represents the distance the a specific sonar is returning. The trajectory related symbols are a blue circle for the current point the robot is trying to approach, the blue circumferences the programed points in the trajectory and the generated trajectory is represented by a blue line. The magenta dotted line represents the robot's path so far.

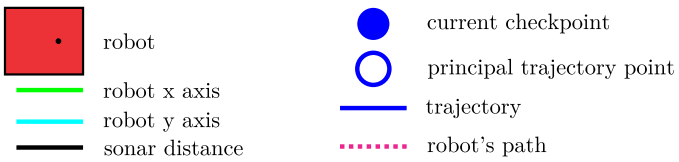

Figure 4: Representation used in the simulation environment.

3 Solution

In our algorithm we approach trajectory following by dividing the trajectory in several checkpoints and then the robot converges to each one of these points sequentially. The algorithm updates the current checkpoint to the next one when the robot is close enough to the current checkpoint or the controller as converged.

Our algorithm, can be divided in four general steps. Every iteration starts by reading the information from the robot's odometry and sonars. Then, using correction techniques it establishes the robot's current attitude in the

simulation referential. After that, it checks if the current checkpoint needs to be updated and updates it if necessary. From this attitude and checkpoint, it calculates the appropriate control commands to send to the robot and the dispatches them. It does until this the robot arrives at the last checkpoint. Note that we didn't use any timer.

4 Trajectory Generation

In this work, we created an algorithm that automatically generates the points that serve has the trajectory's build blocks. Our algorithm takes advantage of the building's geometry to generate the points that will connect in a curve around the building's inner corners without intersecting the walls.

Let w_{in} be the point where two inner walls intersect, w_{out} the intersection between two outer walls and a a vector that goes halfway from w_{in} to w_{out} (see Figure 5). Lets define p_1 and p_2 as points before and after a given turn, equidistant to w_{in} and centered in the hallway. We want p_1 and p_2 to be in the center of the respective hallway to ensure that the robot is not dangerously close to any wall. Defining a line l that passes through w_{in} and is perpendicular to a, its possible to obtain p_1 and p_2 by moving $w_{in} \pm \|a\|$ along l.

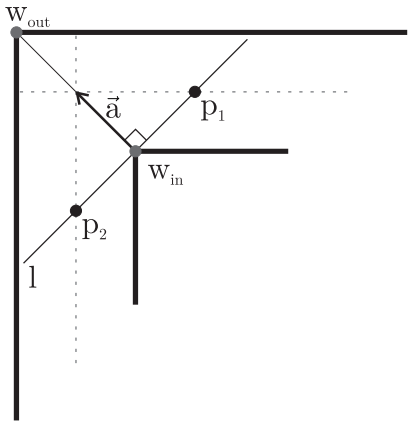

Figure 5: Point geometry in the corners. The dotted lines represent the center of the respective hallways.

After these points have been calculated, the algorithm adds another point between every corner's p_2 and p_1 of the next corner to ensure that the trajectory maintains a straight line in the hallways.

Although this technique can generate all points necessary for the robot to travel in the hallways, it cannot be used to navigate inside the room since there are several obstacles in it such as furniture that our map does not take into consideration. Therefore the points necessary to navigate inside the room were obtained experimentally so that the trajectory makes a smooth curve, going from the starting position towards the door. After these points have been determined, we obtain the final set of points that can be seen in Figure 6 where the blue points are those generated automatically, the red points are the ones experimentally determined and the start/finish point is marked in green.

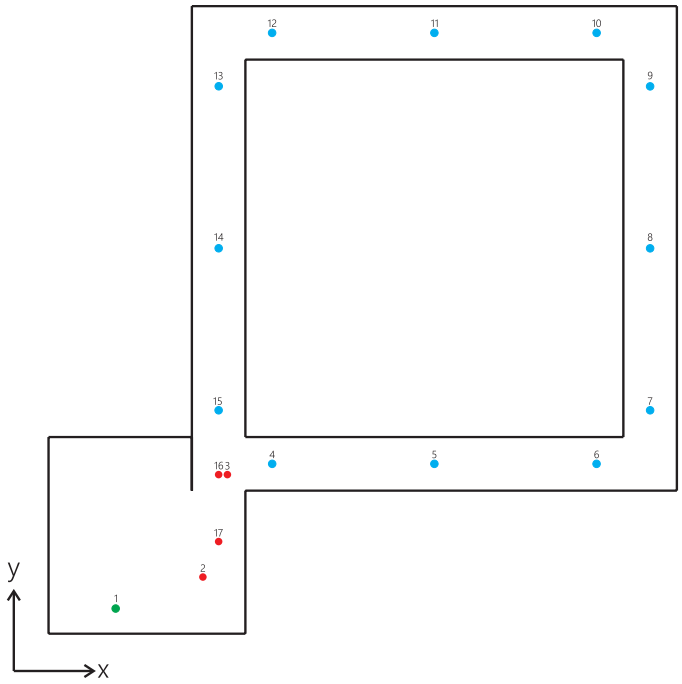

Figure 6: Principal points of the trajectory.

Since the robot follows a trajectory by reaching several checkpoints, there is a need to generate the set of checkpoints from the set of principal points obtained previously. A checkpoint is a reference state that the robot must reach. We can define checkpoints as a function of time as follows, let $\mathbf{c} : [0, 1] \to \mathbb{R}^3$

$$\mathbf{c}(t) = \begin{bmatrix} \mathbf{p}(t) \\ \theta_{\text{ref}}(t) \end{bmatrix} \tag{2}$$

where $\mathbf{p}(t) : [0, 1] \to \mathbb{R}^2$ is the spline curve that passes through the above reference points, and θ_{ref} is the angle of reference for the robot. Due to some specifications of our controller we needed to find θ_{ref} with the following constraints:

- the range of θ_{ref} must be $\mathbb{R}$.
- θ_{ref} should be continuous and differentiable.
- θ_{ref} must be the angle between the x-axis (from the frame of reference) and $\dot{\mathbf{p}}(t)$.

Let ϕ be the angle between the x-axis (from the frame of reference) and $\dot{\mathbf{p}}(t)$, then

$$\theta_{\text{ref}}(t) = \phi_0 + \int_0^t \frac{d\phi(\tau)}{d\tau} d\tau \tag{3}$$

where $\phi_0 = \text{atan2}(\dot{y}_{\text{ref}}(0), \dot{x}_{\text{ref}}(0))$ and

$$\frac{d\phi}{dt} = \frac{\langle R\dot{\mathbf{p}}, \ddot{\mathbf{p}} \rangle}{\|\dot{\mathbf{p}}\|^2}$$

with

$$R = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

The derivation of $\frac{d\phi}{dt}$ can be found in (1), note that this definition of $\theta_{\text{ref}}(t)$ satisfies all the constraints, it is C^1 since $\frac{d\phi}{dt}$ is a product and a division of C^1 functions, its range is in $\mathbb{R}$ since it is the integral of $\frac{d\phi}{dt}$, meaning that a value is being accumulated over time, hence there is no bound on this accumulated value, and the final constraint is satisfied because $\theta_{\text{ref}}(t) = \phi(t)$. The integration was calculated using the rectangle rule of numerical integration.

For practical reasons $\mathbf{c}$ is discretized as follows:

$$\mathbf{c}_i = \mathbf{c}\left(\frac{i}{N-1}\right) \ \forall i \in \{0, \dots, N-1\}$$

Where N is the number of samples.

5 Controller Design

At each point in time a checkpoint is selected, each checkpoint specifies a reference state that the robot should approximate. Basically our controller must solve a regulation task, meaning that $|\mathbf{q}(t) - \mathbf{c}_i| = 0$ as $t \to \infty$, for some selected $i \in \{0, \dots, N-1\}$.

To solve this problem, inspired by model predictive control we state the regulation problem as the following optimization problem.

$$\min_{\mathbf{u}} \ \|\mathbf{q}_{\text{ref}} - \mathbf{q}(\mathbf{u}, t)\|^2 \tag{4}$$

From second order Taylor expansion we can approximate $\mathbf{q}(\mathbf{u}, t)$ as

$$\mathbf{q}(\mathbf{u}, t_0 + h) \simeq \mathbf{q}(t_0) + h\dot{\mathbf{q}}(\mathbf{u}, t_0) + \frac{h^2}{2}\ddot{\mathbf{q}}(\mathbf{u}, t_0)$$

$$\Leftrightarrow \mathbf{q}(\mathbf{u}, t_0 + h) \simeq \mathbf{q}(t_0) + h \begin{bmatrix} c_\theta v \\ s_\theta v \\ \omega \end{bmatrix} + \frac{h^2}{2} \begin{bmatrix} -s_\theta \omega v \\ c_\theta \omega v \\ 0 \end{bmatrix} \tag{5}$$

(5) is a second order prediction of $\mathbf{q}$ after time h, which in our implementation is $h = 0.1$, some later experiments pointed to a less magical value for h such as the inverse of the frequency of the robot.

Now the optimization problem becomes

$$\min_{\mathbf{u}} \left| \mathbf{q}_{\text{ref}} - \mathbf{q}(t_0) - h \begin{bmatrix} c_\theta v \\ s_\theta v \\ \omega \end{bmatrix} - \frac{h^2}{2} \begin{bmatrix} -s_\theta \omega v \\ c_\theta \omega v \\ 0 \end{bmatrix} \right|^2 \tag{6}$$

Remember that $\mathbf{u} = [v, \omega]^T$. To solve the optimization problem we used the gradient descend algorithm as follows :

Let $\mathcal{L}(\mathbf{u})$ be the cost function defined in (6).

The initial guess for $\mathbf{u}$ is the solution of

$$\min_{\mathbf{u}} |\mathbf{q}_{\text{ref}} - \mathbf{q}(t_0) - hM(\mathbf{q})\mathbf{u}|^2$$

Algorithm 1 Gradient Descend

```
1: procedure GRADIENTDESCEND(q_REF, q, h)
2:     maxIte ← 100
3:     ε ← 1E − 10
4:     ite ← 0
5:     u ← M(q)^T (q_ref − q)
6:     do
7:         u ← u − h∇L(u)
8:         ite ← ite + 1
9:     while ||∇L(u)||^2 > ε and ite < maxIte
10:    return u
```

Which is $\mathbf{u}^* = M(\mathbf{q})^T(\mathbf{q}_{\text{ref}} - \mathbf{q})$.

This initial guess was not used as the final answer because it can be shown that if the robot as reached the reference orientation and it is parallel to the line generated by the reference point and unitary vector with the reference orientation then the robot will get stuck in this position, therefore it is not useful to solve a regulation problem.

The gradient is calculated using the centered differences of the cost function $\mathcal{L}(\mathbf{u})$. Also for security reasons we bound control output at 30cm/s.

A demo of our controller is available on-line at (2). This controller is not perfect in the way that $|\mathbf{q}(t) - \mathbf{c}_i| = \varepsilon, \quad \varepsilon > 0$ as $t \to \infty$, but it successfully converges to a curve by solving various regulation problems for each checkpoint. Tthe results from our controller performance in a simulation environment can be found in Figure 7. This results shows that our controller can perform this task with almost never leaving the programed path. When applied directly to our robot, we found that this controller failed since using the real robot induced errors in the system that came from wrong odometry information. This information would lead our simulation to believe that the robot was at the center of the hallways when in the reality it was close to the walls. This type of errors can happen when the robot's wheels turn at slightly different speeds from each other for instance.

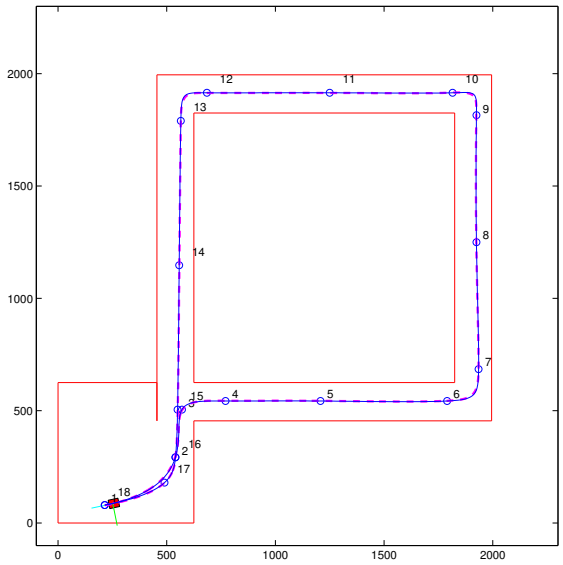

Figure 7: Controller's performance in the simulation environment.

6 Error Correction

In order to compensate the errors in the robot's position caused by the erroneous odometry information we use the on board sonars.

This robot has eight sonars spread around the robot,s front,they are 20 degrees apart from each other and return the distance to an object by measuring the time a sound wave takes to leave the sonar, bounce at an object and return to the sonar receiver. In this specific robot, the sonars have a working range between 300 and 5000 millimeters. The way sonars operate makes them susceptible to errors and even if the robot does not move, between to measures the values can vary significantly. To smooth out some of the this noise, we applied a mean filter over a window of 10 measures.

In order to correct the odometry, we estimate the robot's position using information from the sonars. We estimated the position of the robot by solving the following problem:

If $\mathbf{r}$ is the robot's position, d_i the distance signal given by each sonar $i \in \{1, \ldots, 8\}$ and dref_i is the distance to the closest wall to the robot based on the current estimate r.

Assuming for now that d_i has no noise for every i, then d_i is the distance of the robot to a wall. Using the difference between dref_i and d_i we can estimate the real position of the robot. dref_i can be calculated by computing the intersection of the line corresponding to the closest wall (l_1) and a line that passes through the robot and the sonar i (l_2) (see Figure 8).

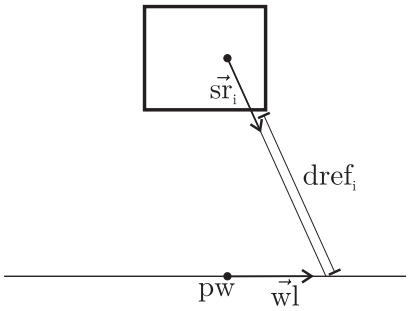

Figure 8: Representation used in the simulation environment.

This line intersection problem turns out to be a solution to a simple linear equation:

$$\mathbf{r} + t\mathbf{sr}_i = \mathbf{pw} + s\mathbf{wl}$$

$$\Leftrightarrow \begin{bmatrix} \mathbf{sr}_i & -\mathbf{wl} \end{bmatrix} \begin{bmatrix} t \\ s \end{bmatrix} = \mathbf{pw} - \mathbf{r}. \tag{7}$$

Because $|||\mathbf{sr}_i||| = 1$, $\text{dref}_i = t - k$, where k is the distance between $\mathbf{r}$ and the sonar i, in our implementation this is just a constant.

A final position is obtained from the sonar information using the following formula:

$$\mathbf{r} + \frac{1}{8}\sum_{i=1}^{8}(\text{dref}_i - d_i)\mathbf{sr}_i \tag{8}$$

Using equation (8) we can obtain a better estimate of the real robot position, but now a new problem appears since we need to integrate this information with the data that comes from the odometry. To solve this issue we accumulated the differences in the odometry while adding position corrections, this translates in the following equation:

$$\mathbf{r}_k = \mathbf{r}_{k-1} + (\delta_k - \delta_{k-1}) + \frac{1}{8}\sum_{i=1}^{8}(\text{dref}_i - d_i)\mathbf{sr}_i \tag{9}$$

where $\mathbf{r}_k$ is the estimated position at time k, δ_k is the odometry signal at time k. We do not correct the odometry's information regarding θ since it does not vary that much from the real value and our controller is able to coupe with such error.

Although this technique works with information from all sonars, we found that using only the two side sonars would give better results since the other sonars tend to project the robot to a point behind where it really is. We also limited the range in which the sonars actuate to 1500mm so that it can be tolerant to big openings such as the entrance for the elevators or open doors. We choose this maximum length based on the dimensions of the hallways.

7 Results

The final results of our work are presented in this section.

Comparing the control signals of both simulation (see Figure 9) and in the real robot (see Figure **??**) for the same task, we can clearly observe the increase of noise in both v and ω in the robot, this can be explained by the errors on the actuators and sensors, of the real world.

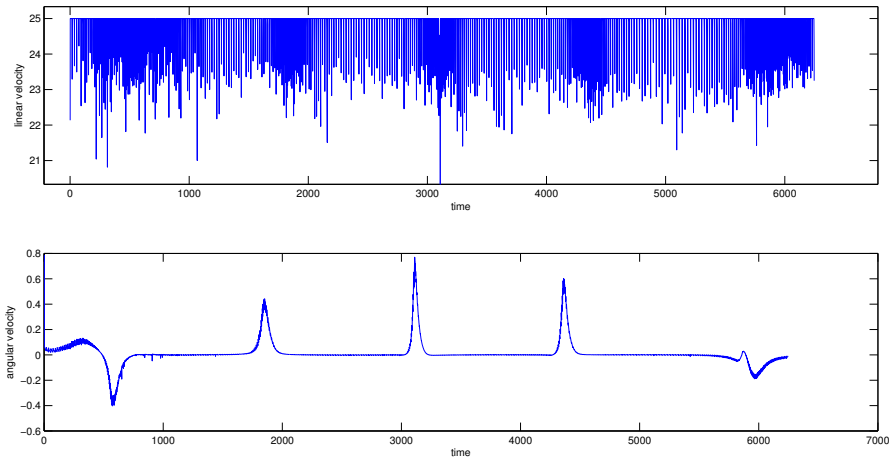

Figure 9: Control signal in simulation with loss linear velocity bound. Linear velocity signal is has so much noise because the robot is always stopping at each checkpoint by design

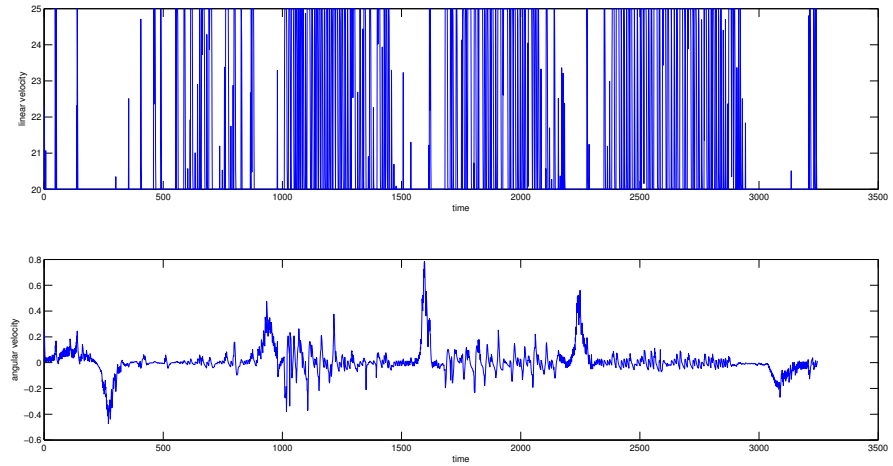

Figure 10: Control signal in simulation with tight linear velocity bound. Robot doesn't stop at each checkpoint because it has a high lower bound in its linear velocity.

We also collected the information from the sonars while the robot was moving. This allowed us to approximately reconstruct the buildings blueprint. The results of this experiment are shown in Figure 11. In it we can see that our map is actually very accurate to the reality. Since the sonars were bounded to a maximum distance that was related to the dimensions of the hallway, we can see the access to the open areas of the floor such as open doors to other rooms, the entrance to the elevators and service stairs. Inside the starting room, on the other hand the data revealed mostly noise caused by people moving near the robot and glass furniture nearby.

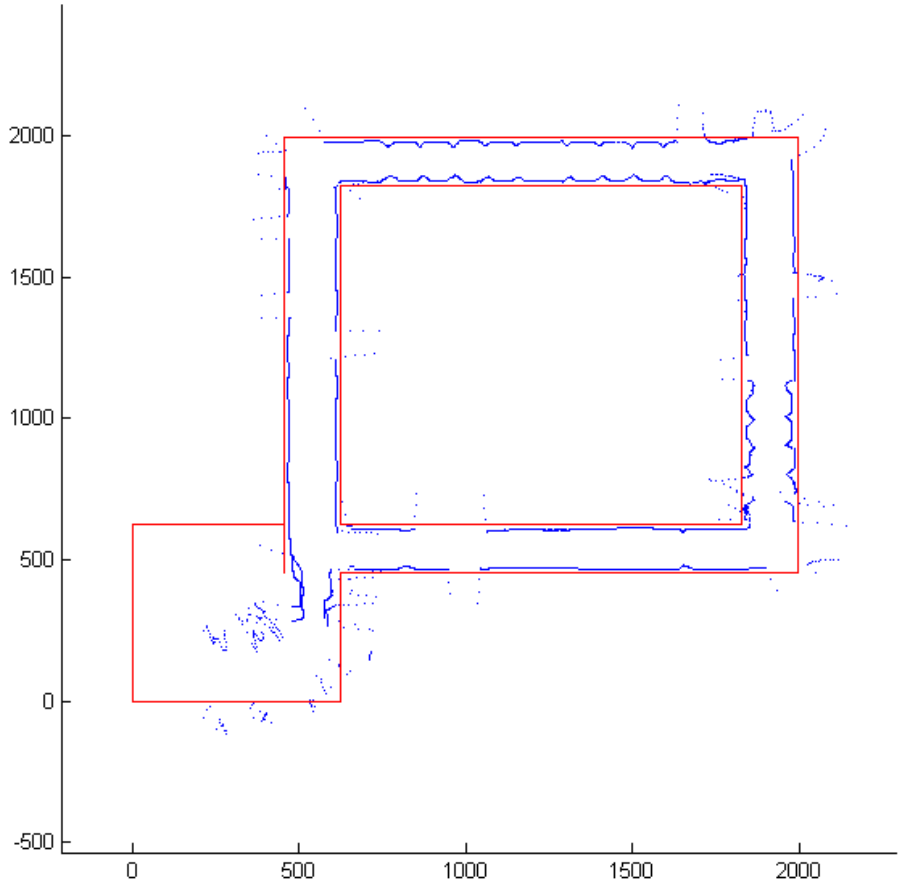

Figure 11: Map made from sonar's data

Figure 12 shows the difference in the robot's position that would be obtained by using only the information from the odometry (green line) and the positions obtained using the sonars and the differential odometry (pourple line). We can clearly see that using only the odometry, the robot would have ran into walls and it strugles to converge to the reference path.

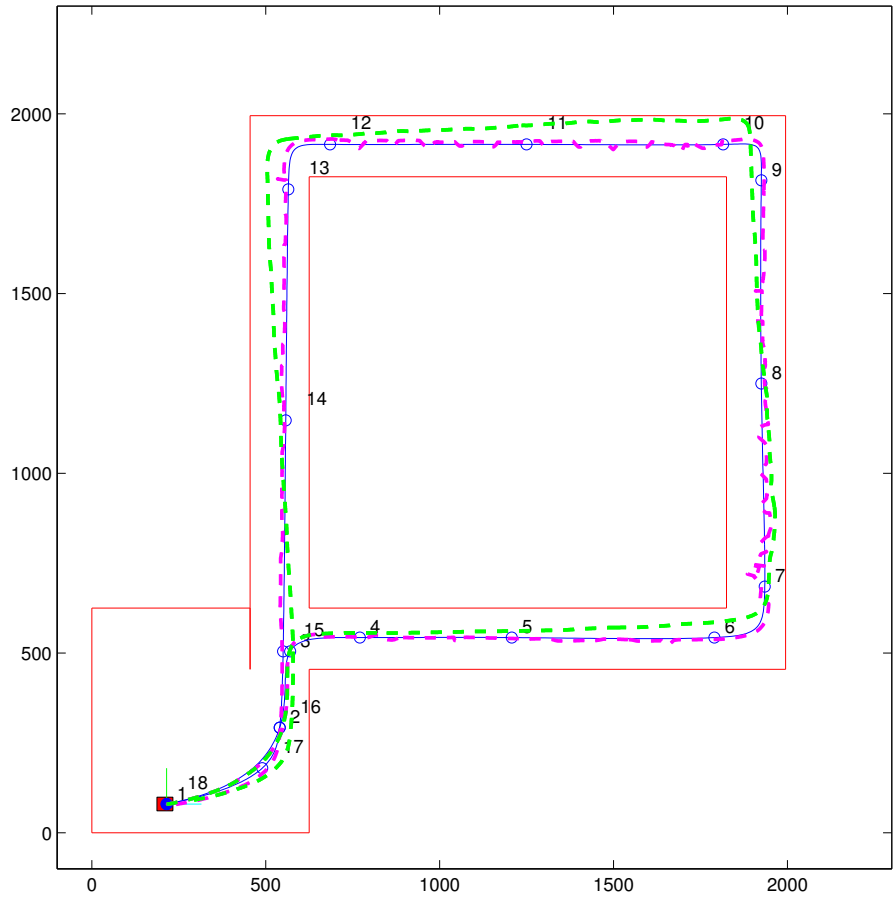

Figure 12: Comparison between the positions obtained using only odometry and using the sonars and the differential odometry combined.

A video showing the performance of our controller can be found at. (3)

8 Conclusions

In this work, we design and implemented a controller based on model predictive control that would allow our robot to move following a given path. After tuning all controller parameters in a simulation environment, we had to face the reality that in real life robot's have flaws that cause errors that wont happen in a simulation.

In real robots, the position sensors take time to update its values, therefore we end up with much less information compared to the simulation because in simulation, close times will have a different positions and in our robot several consecutive time instance will have the same information since the values have not been updated yet. In addiction,

the simple fact that one wheel is more worn out than other will cause the robot to steer away from the path without the position sensors noticing it. This type of errors can be difficult to correct without knowing the exact model of the error for a specific robot. Knowing this model can lead to very high performances for a specific robot, but since the model of the error varies with the robot, this type of approaches cannot achieve a relatively good performance in all the robots with a single error model.

In our approach, we used information from other onboard sensors, the sonars, to correct this errors in the robot's position by trying to find where the robot is based on the distance to the walls. Although this information also being noisy, it can be treated much easy by applying a simple mean filter.

After correcting this problem, we adjusted some of the parameters such as the minimum distance to a checkpoint before it moves and the limit to each we consider that our controller has converged. The combination of those corrections and adjustments, allow our real robot to behave similarly to our simulation enabling it to follow the complete path.

References

[1] Weisstein, Eric W. "Curvature." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Curvature.html

[2] https://www.khanacademy.org/computer-programming/car-parking/6218382018871296

[3] http://1drv.ms/1JzRmRp