DIETI UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

Differential-Drive Robot

Technical project #2

Professors
Prof. Jonathan Cacace
Prof. Fabio Ruggiero

Candidate
Andrea Fimiani
P38/12

Academic year 2020/2021

Summary

Problem specification

The purpose of the project is to simulate a robot that must navigate an unknown environment looking for markers placed in unknown positions in different rooms, knowing only the coordinates of the areas to be explored. During the exploration task the robot needs to localize itself inside the environment, estimating its pose just using a localization technique based on information retrieved by its sensors. To complete the task, the robot must navigate all the rooms until the id requested by the user is found or all the rooms are explored, then it must return to its starting position and save the map of the explored areas.

Package structure

The package built to solve the problem is written in C++ and based on ROS. The robot used in the package is a differential drive which will navigate all the environment searching for QR codes. The motion task is carried out setting progressive locations to reach. To assign the robot's current destination the package modifies the values of two *rosparameters* named "*xf*" and "*yf*". The user can also change these values at runtime to specify a new area to be explored. Once arrived at its destination, the robot starts to search for marker inside the area. When a QR code is found the robot continues to explore the other rooms, if the id is the desired one it goes back to its starting position. Lastly, when returned on its starting position, it saves the map of the environment explored. To fulfil the problem specifications five nodes have been used, each one with specific tasks to manage. The five nodes are:

- The Odom node: elaborate and send the odometrical information
- The Gmapping node: elaborates and publish the Occupancy Grid
- The Planner node: sends the desired trajectory to the control node, runs the obstacle detection and saves the map when the exploration finishes
- The OpenCV node: analyses the camera messages and looks for marker
- The Control node: retrieves the desired acceleration and elaborates the commands for the robot

The following diagram shows the connections between the package nodes. The topics and the message's types are indicated next to the connection arrows.

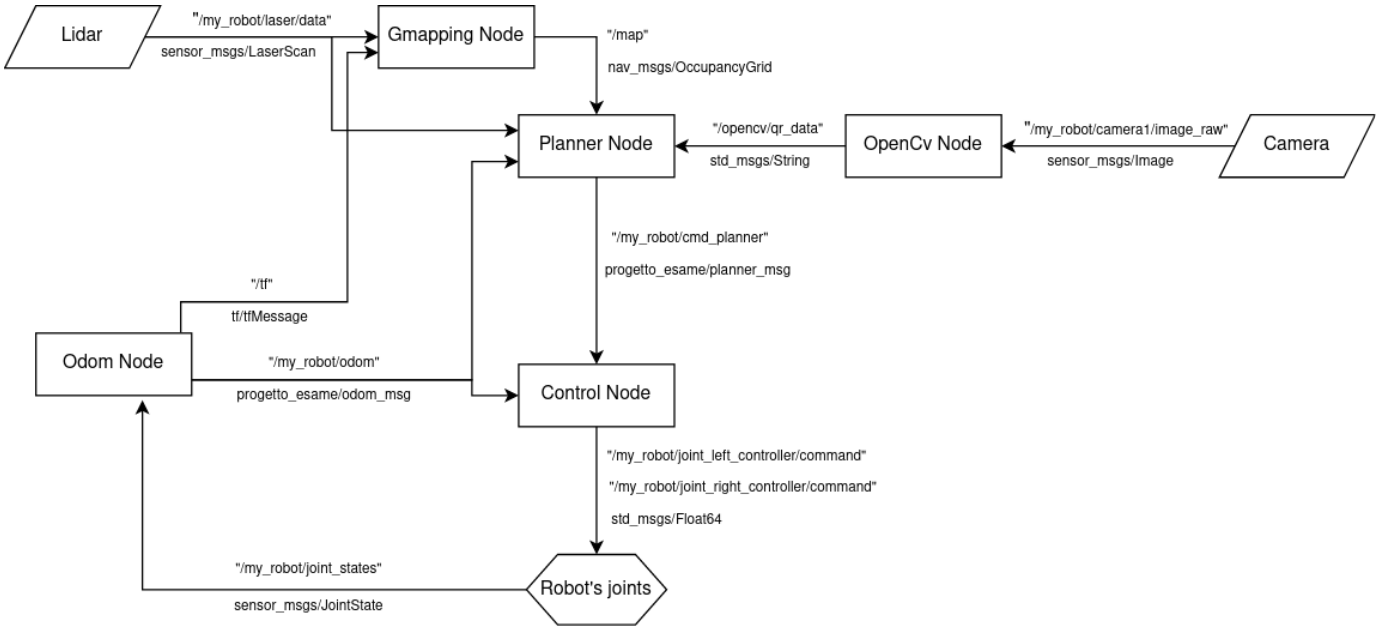


Figure 1 - Package connections

Models employed

Robot description

The robot used for the project is a differential drive robot. The model has been chosen after state-of-the-art research between commercial differential drive robots. The model chosen is available on GitHub [1]. Two wheels are connected to the robot's base link by rotational joints. The base link has a cylindrical shape of radius 0.165 m and height of 0.071 m and has a mass of 2 Kg. The wheels rotation axes are orthogonal to the central axis of the base link. The wheels are symmetrically attached to the base link at 0.12 m from the center, each one has a mass of 0.01 Kg, width of 0.023 m and radius of 0.033 m. The caster wheel is attached on the front at 0.12 m from the center, it has a mass of 0.01 Kg and radius of 0.018 m.

The original URDF has been modified to simplify the structure of the robot: the bumper has been deleted, a camera and a laser sensor have been added. All the sensors have been attached on the fore side of the robot.

The camera specifications are:

- Resolution: 1280x720
- Image format: R8G8B8
- Update rate: 30.0
- Horizontal field of view: 1.39

The laser specifications are:

- Minimum and maximum angle: ±3.14
- Min range 0.15
- Max range 4.0
- Update rate 40
- Number of rays 360

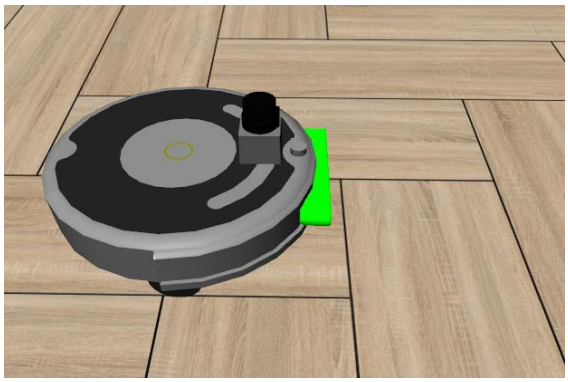


Figure 2 - Robot model in Gazebo

World description

The world created for the simulation is large 14 x 14 m, has four bounded areas that represent three rooms to be explored and a room where the robot starts its task.

Figure 3 - World seen from above

The chosen reference positions for each room are:

- Room #1: $(x = 3, y = 8)$
- Room #2: $(x = 7, y = 11)$
- Room #3: $(x = 11, y = 6)$
- Starting room: $(x = 5.5, y = 1.5)$

These positions were chosen to facilitate the setup of different case studies. For example, by simply choosing room #1 as first room to be explored, it's possible to simulate the problem of local minimum of the artificial potential method.

The QR codes are placed inside the rooms such that the robot needs at least one cycle of exploration routine to be found. The positions are:

- Marker “id1” in room #1 position $(x = 0.05, y = 5.5)$
- Marker “id2” in room #2 position $(x = 9.9, y = 12.3)$
- Marker “id3” in room #3 position $(x = 10.05, y = 4)$

All the markers are at 0.3 m height so that the robot’s camera can see them. At the center of the world there are models taken from gazebo library [2] used as obstacles.

Figure 4 - View of the world and obstacles

Below the QR code used

Figure 5 - Marker id1

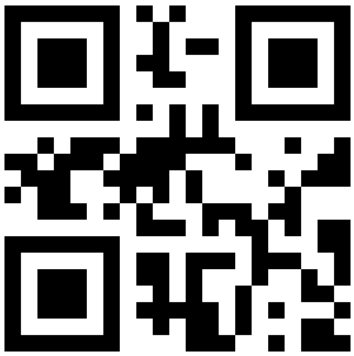


Figure 6 - Marker id2

Figure 7 - Marker id3

Odometry

The odometry node must calculate the exact pose of the robot using only the information about the wheels angular position retrieved in the topic *"/my_robot/joint_states"*. To increase precision on the pose estimation this node runs at 1000 Hz.

Starting from the differences $\Delta\phi_R$ and $\Delta\phi_L$ between the old and the new angular position of the two wheels, the linear and the angular velocity of the robot are calculated using:

$$\begin{cases} v_k = \dfrac{\rho}{2}\dfrac{(\Delta\phi_R + \Delta\phi_L)}{dt} \\ \omega_k = \dfrac{\rho}{2D}\dfrac{(\Delta\phi_R - \Delta\phi_L)}{dt} \end{cases}$$

where ρ is the wheel's radius and D the wheels distance. Then if the angular rotation ω_k is not zero, the exact integration is used to update the robot position:

$$\begin{cases} x_{k+1} = x_k + \dfrac{v_k}{\omega_k}(\sin(\theta_k + \omega_k\, dt) - \sin(\theta_k)) \\ y_{k+1} = y_k - \dfrac{v_k}{\omega_k}(\cos(\theta_k + \omega_k\, dt) - \cos(\theta_k)) \end{cases}$$

otherwise Runge–Kutta method of second order is used

$$\begin{cases} x_{k+1} = x_k + v_k \cos\left(\theta_k + \dfrac{\omega_k\, dt}{2}\right) dt \\ y_{k+1} = y_k + v_k \sin\left(\theta_k + \dfrac{\omega_k\, dt}{2}\right) dt \end{cases}$$

then all the information is published on the *"/my_robot/odom"* topic using a custom message *"progetto_esame/odom_msg"* constructed as follow

progetto_esame/odom_msg
Header header **String** child_frame_id **Geometry_msgs/Point** position **Float64** yaw **Float64** vel_x **Float64** vel_y **Float64** w

The first two fields have information about the message sent, the position field has the world coordinate of the robot then the remaining fields are for its orientation, linear and angular velocity.

Trajectory planner

Planner Node

The planner node has the role of path planner. This node needs to subscribe to the odometry topic *"/my_robot/odom"* to retrieve the pose information; to the Lidar topic *"/my_robot/laser/scan"* to get the laser information; to the Marker topic *"/opencv/qr_data"* to select a new location once a marker is found and lastly to the map topic *"/map"* to retrieve the map at the end of the simulation to save it. The planning is carried out using the artificial potential method. Given the robot's pose and the goal position, the planner node computes the attractive potential and the repulsive potential.

For the calculation of the attractive force the node uses two expressions:

$$F_{a\_far} = k_a \frac{e}{\|e\|}; \qquad e = \begin{bmatrix} x_f - x \\ y_f - y \end{bmatrix};$$
$$F_{a\_close} = k_a e;$$

Denoting with (x, y) is the robot position. The first expression is used when the robot is far from the goal and uses the norm of the error such that the force doesn't go to infinite. The planner switches to the second expression when the error's norm is less or equal to 1. The same value of the gain $k_a = 20$ was chosen for both expressions.

To evaluate the repulsive forces coming from obstacles it was developed an obstacle detection algorithm based on the information retrieved from the laser. In the laser callback the planner scrolls through the message and when it sees a value less than the defined obstacle range of influence, it raises an *obstacleFound* flag and continues scrolling the message, keeping track of the point of minimum distance, until the value read becomes grater then the obstacle range of influence. When this occurs the *obstacleFound* flag is lowered and the repulsive potential of that obstacle is calculated and added to the total repulsive potential. Inside the obstacles detection there is also implemented the solution for the problem due to the laser discontinuity, that causes to see an obstacle in front of the laser as two different obstacles. Using this obstacle detection routine there is an issue near corners, the robot aligns to the bisector and crashes in one of the walls. This behaviour is due to the two walls forming a corner that are seen as one obstacle which nearest position alternates from the left to the right side of the robot. An algorithm was written to avoid this issue: if more than 20 consecutive laser rays detect an obstacle, it is split in two, enabling the calculation of two different position for the walls. The expression for the elaboration of the repulsive force is

$$F_r = \sum_i \begin{cases} \frac{k_r}{\eta_i^2}\left(\frac{1}{\eta_i} - \frac{1}{\eta_o}\right)^{\gamma-1} \nabla\eta_i, & \eta_i \le \eta_o \\ 0, & \eta_i > \eta_o \end{cases}$$
$$k_r = 4;\ \eta_o = 1.2\ m;\ \gamma = 2;$$

Where η_i represents the i-th obstacle distance and η_o the obstacle range of influence.

To elaborate the gradient $\nabla\eta_i$ numerically the follow derivation has been made.

Defined η_i as the minimum distance from the robot and the obstacle, and indicated as $\begin{bmatrix} x \\ y \end{bmatrix}$ the robot's coordinate in world representation

$$\eta_i = \min_{j \in CO_i} \left\| \begin{bmatrix} x_{o,j} \\ y_{o,j} \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix} \right\|$$

If we rewrite the obstacle position as function of the laser information retrieved by the robot

$$\begin{bmatrix} x_{o,j} \\ y_{o,j} \end{bmatrix} = \begin{bmatrix} x + \rho_j \cos\varphi_j \\ y + \rho_j \sin\varphi_j \end{bmatrix};$$

Denoting ρ as the distance from the robot to the object and φ as the sum of the robot's yaw and the angle between the robot's heading direction and the straight line connecting the robot to the obstacle.

Replacing in the function η_i we have

$$\eta_i = \min_{j \in CO_i} \left\| \begin{matrix} \rho_j \cos\varphi_j \\ \rho_j \sin\varphi_j \end{matrix} \right\|$$

Then applying the definition of gradient for cylindrical coordinate:

$$\nabla\eta_i = \frac{\partial\eta_i}{\partial\rho}\hat{\rho} + \frac{1}{\rho}\frac{\partial\eta_i}{\partial\varphi}\hat{\varphi}$$

with

$$\frac{\partial\eta_i}{\partial\rho} = \begin{bmatrix} \cos\varphi \\ \sin\varphi \end{bmatrix};$$

$$\frac{\partial\eta_i}{\partial\varphi} = \begin{bmatrix} -\rho\sin\varphi \\ \rho\cos\varphi \end{bmatrix};$$

Once the total force has been calculated, the node publishes it in the command topic *"/my_robot/cmd_planner"* using a custom message *"progetto_esame/planner_msg"* with the following structure

progetto_esame/planner_msg
float64 acc_x **float64** acc_y **float64** theta_acc **bool** stay_still

the first three value are information relative to the acceleration vector and the fourth value is a flag that stops the robot wheels when raised.

This node also implements the detection of the local minima and gives the robot new random locations to be reached inside the world in a maximum of 60 sec. To speed up the simulation, the robot checks along the four main directions and choses a new random location in an obstacle-free direction.

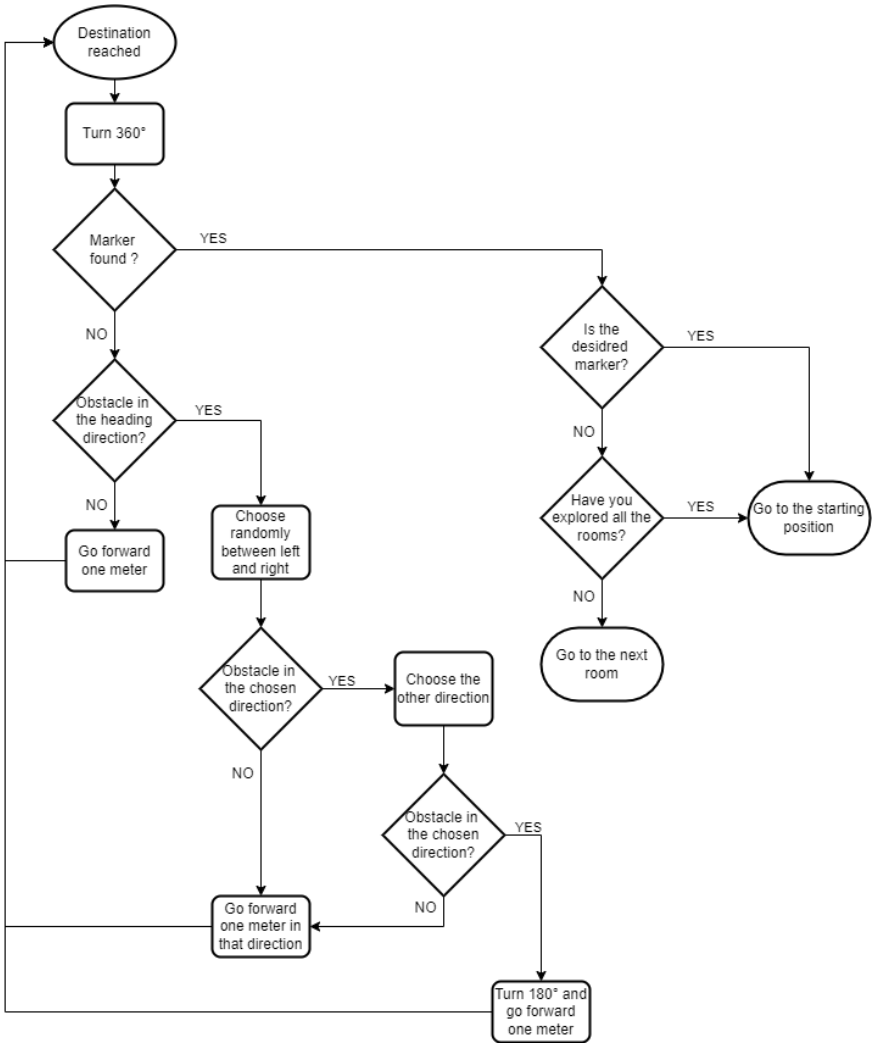
At the end of the simulation, when the robot has come to his starting position, the node calls the *SaveMap* function to elaborate the last Occupancy Grid retrieved from gmapping and saves the map of the explored environment in the package's folder.

Exploration routine

For the exploration routine the differential drive robot exploits information received from the laser sensor relative to the obstacles distance along four directions: heading, backward, left and right.

As the robot reaches the desired position in the room turns 360° and looks for the marker. If not found, the robot starts the main cycle: proceeds straight for a meter and turn 360°, until the marker is found or has in its heading direction an obstacle closer than 2.2m.

From this position, chooses randomly a new heading direction between right and left, checks for the obstacle clearance, and if there are obstacle in the chosen direction picks the other one, then repeats the main cycle. If the robot can't move along any of the three directions, turn 180° and continues with the main cycle. If during the exploration task the robot finds the desired marker returns to the start position, otherwise adds the marker to list and proceeds to the next room.



Gmapping node

The ROS Gmapping package provides a wrapper for the laser-based Simultaneous Localization And Mapping (SLAM), as a ROS node called slam_gmapping. This node creates a 2D occupancy grid map from the laser and pose data collected by the robot. To function properly it needs the transforms between the frame of the link to which the laser is attached, the base link's frame and the odometry frame. It subscribes to the *"/tf"* and *"/my_robot/laser/scan"* topics and publishes the transform between the map and the Odom frames and the Occupancy grid on the *"/map"* topic. Below an image of the Transformation's tree

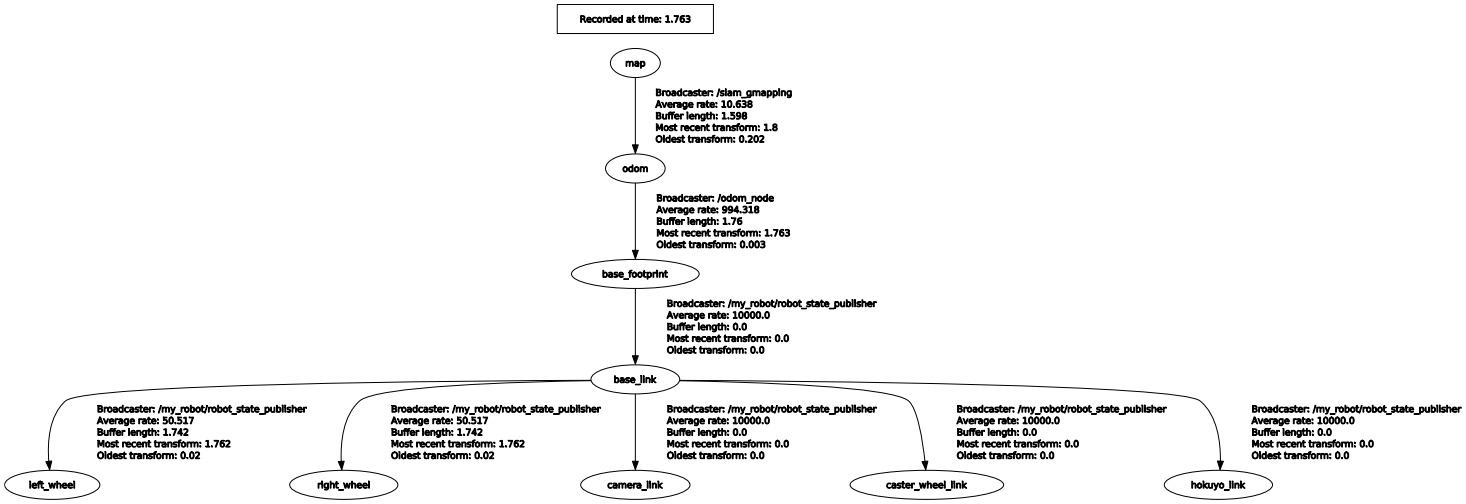


Figure 8 - Tf tree

Some node's parameters have been changed from the default

- Transform publish period default 0.05 →0.1
- Particles 30 →80
- Angular update 0.5 →0.436
- Linear update 1 →0.5
- srr 0.1 →0.01
- srt 0.2 → 0.02
- str 0.1 →0.01
- stt 0.2 → 0.02
- Minimum Score 0 → 100
- Max Urange 80 →6
- Max Range 8
- Map update interval 0.5 → 1

Here the map retrieved from the simulation:

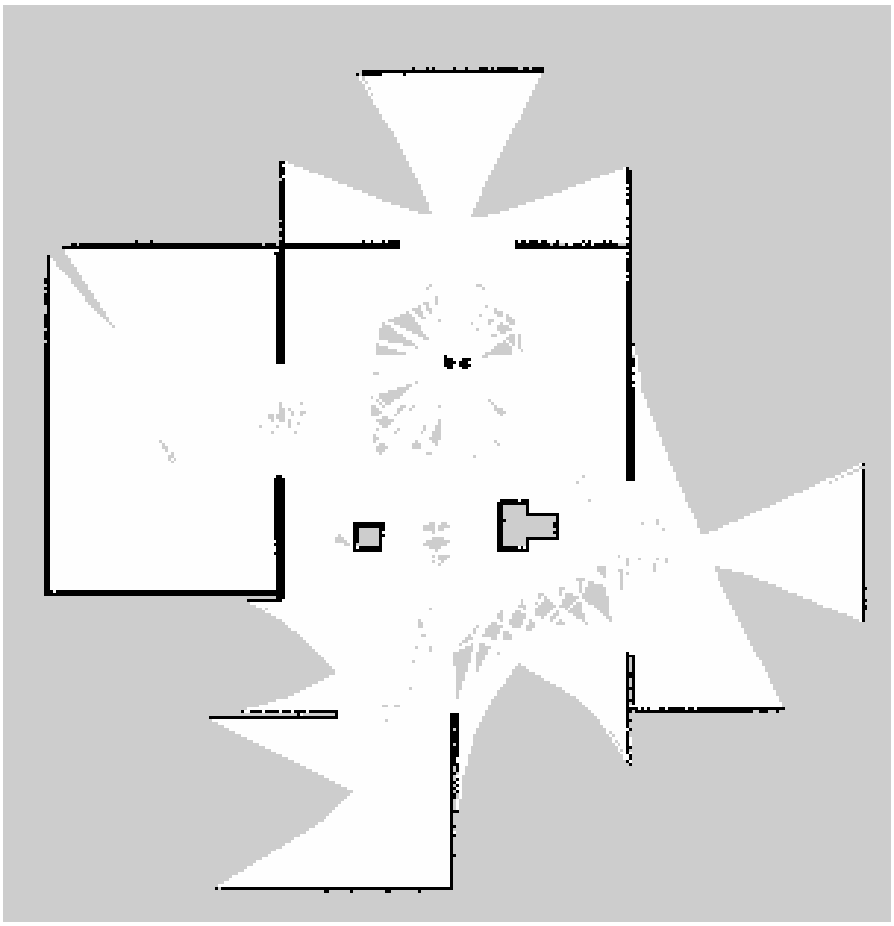


Figure 9 - Map retrieved during the simulation

QR codes detection

The OpenCV node task is the image analysis. For the QR code detection task the node uses the OpenCV [4] and Zbar [5] libraries. To detect the marker's data the camera publishes images on the topic *"/my_robot/camera1/image_raw"*. The class ROS_IMG_READER subscribes to the topic and uses a callback to transform the image. Inside this function the image is turn in grayscale and then, using a fixed threshold of 50, is turned into a binary image. Lastly the binary image is passed to the Zbar's scan which looks for the QR code and then the node publishes all the symbols found into the *"/opencv/qr_data"* topic.

Below the pictures of the QR code seen by the camera and the same QR code once binarized.

Figure 11 - Marker id1 seen by the camera

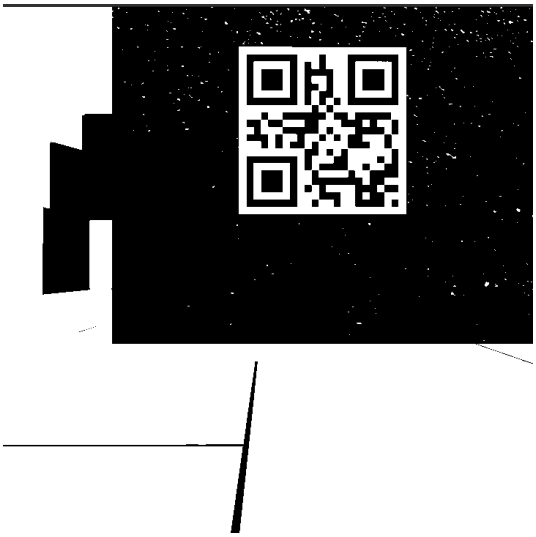


Figure 12 - Marker id1 filtered

Figure 13 - Marker id2 seen by the camera

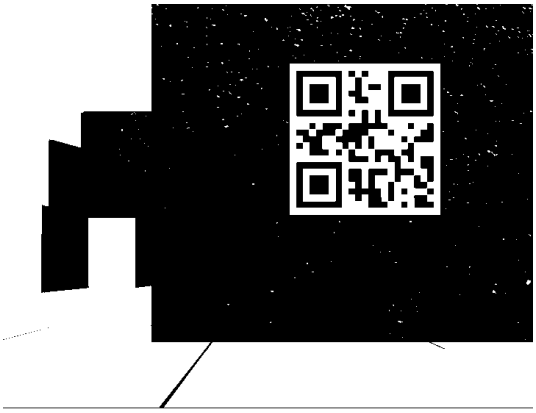


Figure 10 - Marker id2 filtered

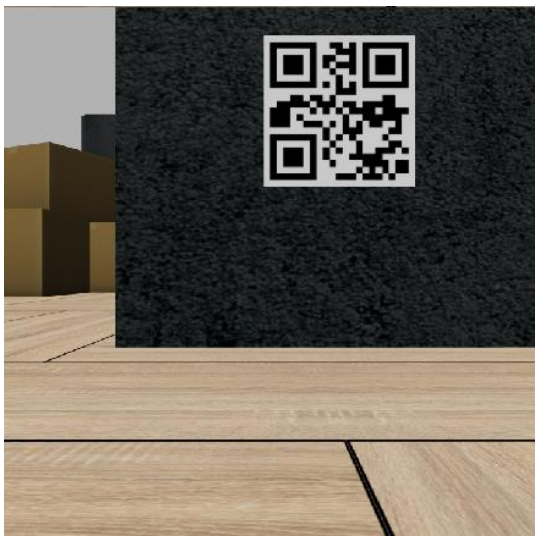


Figure 14 – Marker id3 seen by the camera

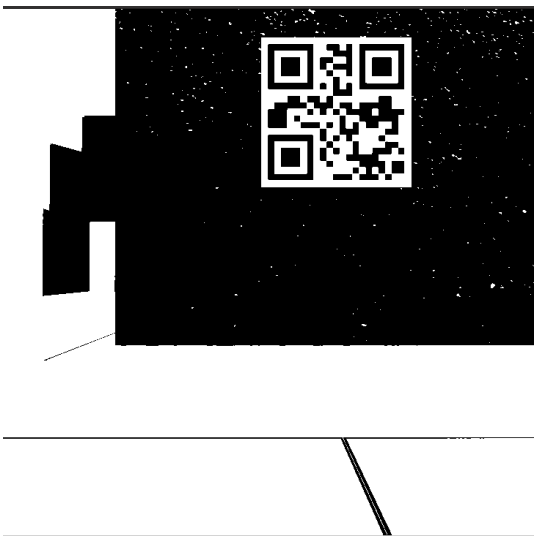


Figure 15 - Marker id3 filtered

Control

In the Control node it is implemented a trajectory tracking control. Defining the state tracking error as:

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{bmatrix}$$

and substituting the kinematic model of the unicycle:

$$\begin{cases} \dot{e}_1 = v_d \cos e_3 - v + e_2\omega \\ \dot{e}_2 = v_d \sin e_3 + e_1\omega \\ \dot{e}_3 = \omega_d - \omega \end{cases}$$

we can define u1 and u2 as two virtual inputs and write the following transformations:

$$\begin{cases} v = v_d \cos e_3 - u_1 \\ \omega = \omega_d - u_2 \end{cases}$$

Then using the transformation to redefine the error as:

$$\dot{e} = \begin{bmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} e + \begin{bmatrix} 0 \\ \sin e_3 \\ 0 \end{bmatrix} v_d + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

where the two virtual inputs were chosen in accordance with the Nonlinear Control approach as:

$$u_1 = -k_1(v_d, \omega_d)e_1$$
$$u_2 = -k_2 v_d \frac{\sin e_3}{e_3} e_2 - k_3(v_d, \omega_d)e_3$$

K2 is a positive constant, while K1 and K3 are two continuous positive gain functions of v_d and ω_d. After state-of-the-art research [3], the two gain functions for the control were chosen as:

$$k_1(v_d, \omega_d) = k_3(v_d, \omega_d) = 2\xi\sqrt{\omega_d^2 + b v_d^2} \quad k_2 = b$$

where ξ is a gain in the range $[0,1]$. The control inputs necessary to the calculation of the joint commands are the information relative to the desired pose, linear and angular velocity. The Control node calculates this information starting from the estimated robot pose, given by the Odom node, and the desired acceleration vector provided by the planner. First integrates the acceleration to get the desired linear velocity vector and then using the following expression calculates the desired angular acceleration:

$$\omega_d = \frac{\ddot{y}_d \dot{x}_d - \ddot{x}_d \dot{y}_d}{x_d^2 + y_d^2}$$

then integrates the velocity to get the desired pose information. The control node ensures that values are always below the admissible velocities' limits.

Once the control elaborates the velocity commands for the differential drive, these are turned into joint velocity commands using

$$\omega_L = \frac{2v - D\omega}{2\rho}$$
$$\omega_R = \frac{2v + D\omega}{2\rho}$$

where ρ is the wheel's radius and D the wheels distance. The velocity commands are then sent by the publisher on the relative joint topics *"/my_robot/joint_left_controller/command"* *"/my_robot/joint_right_controller/command"*.

The parameter used for the control are:

- $v_{Max} = 0{,}066\ m/s$
- $\omega_{Max} = 0{,}275\ rad/s$
- $k_2 = 10$
- $\xi = 0.7$

To improve joint's velocity controllers the following value for the PID has been experimentally selected in the *controller.yaml* file:

- p: 0.04
- i: 0.01
- d: 0.0
- i_clamp: 0.15

Results

A video is produced in appendix showing the complete simulation test. The robot starts and explores all the rooms in the following order: Room#3, Room #2, Room #1. Finds the QR code with “id1” and returns to the starting position. During the navigation to its starting position the robot enters a local minimum condition then after few iterations reaches the starting position. Lastly, once it completes the navigation task, it saves the map of the environment. Here the trajectory followed in the simulation.

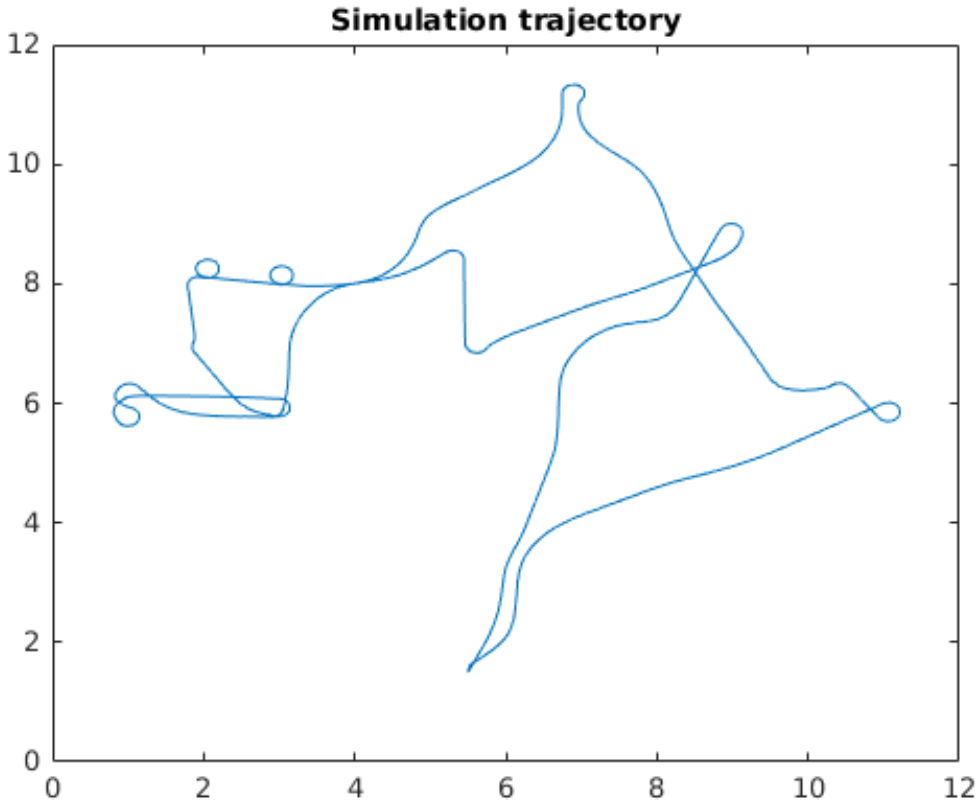


Figure 16 - Plot of the trajectory done in the complete simulation

In the following will be discussed the results of an interval of the simulation done in the environment described in the *“World model”* chapter; then will be discussed the results of a short simulation done in an empty environment.

Results of the simulation in the world environment

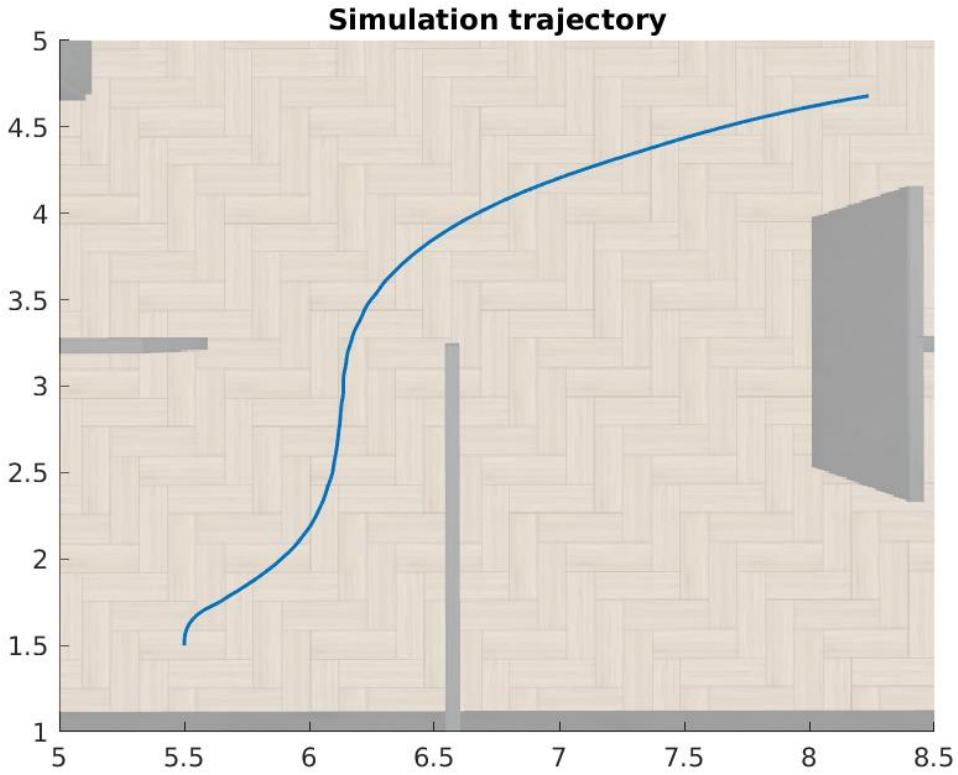


Figure 17 - Plot of the trajectory done

Above the trajectory followed in the simulation for 70 sec., while below the force computed by the planner node along the trajectory.

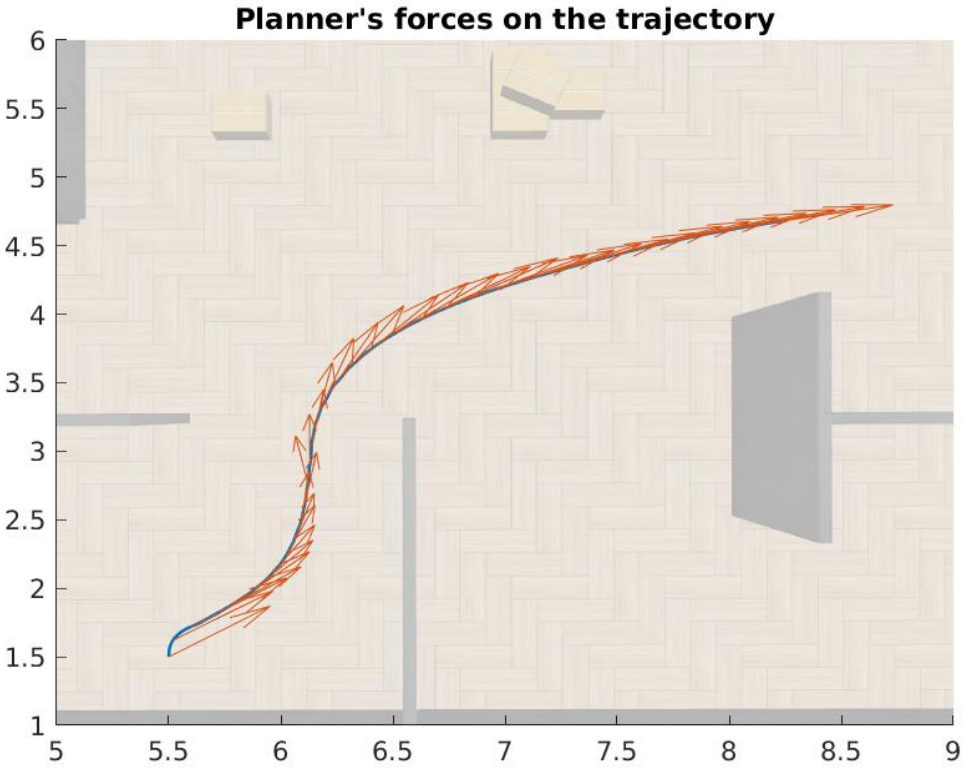


Figure 18 - Plot of the planner's force along the trajectory

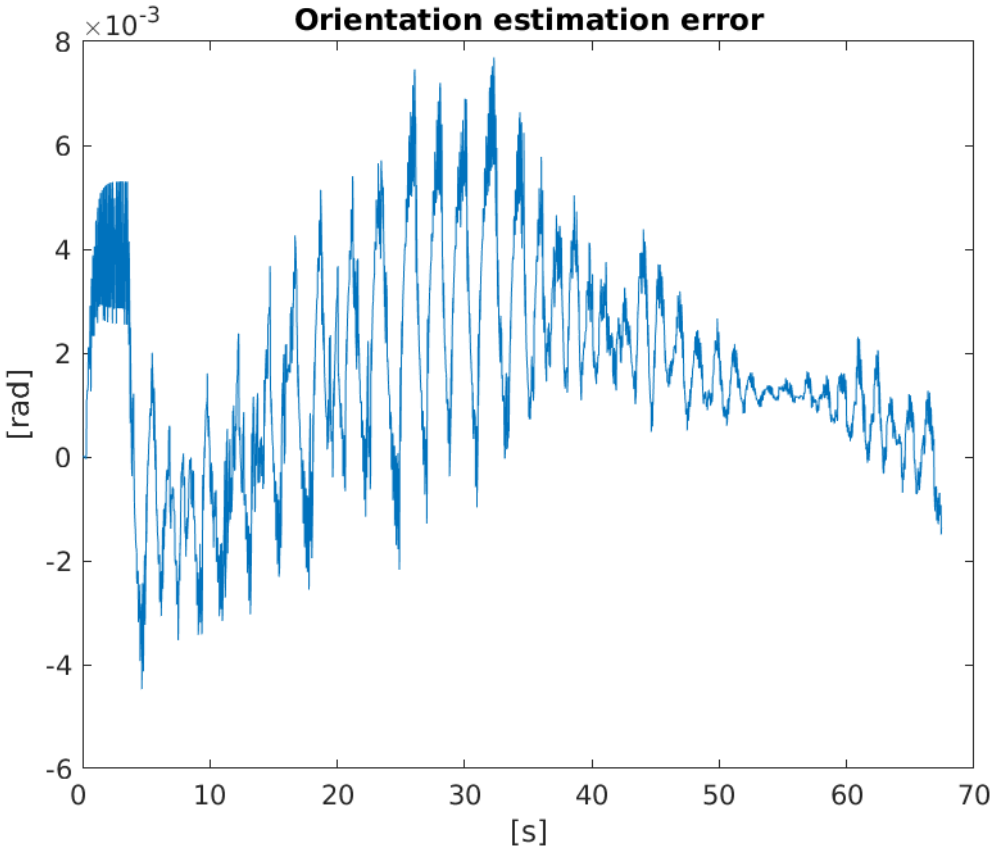


Figure 19 – Plot of the error on the yaw estimation

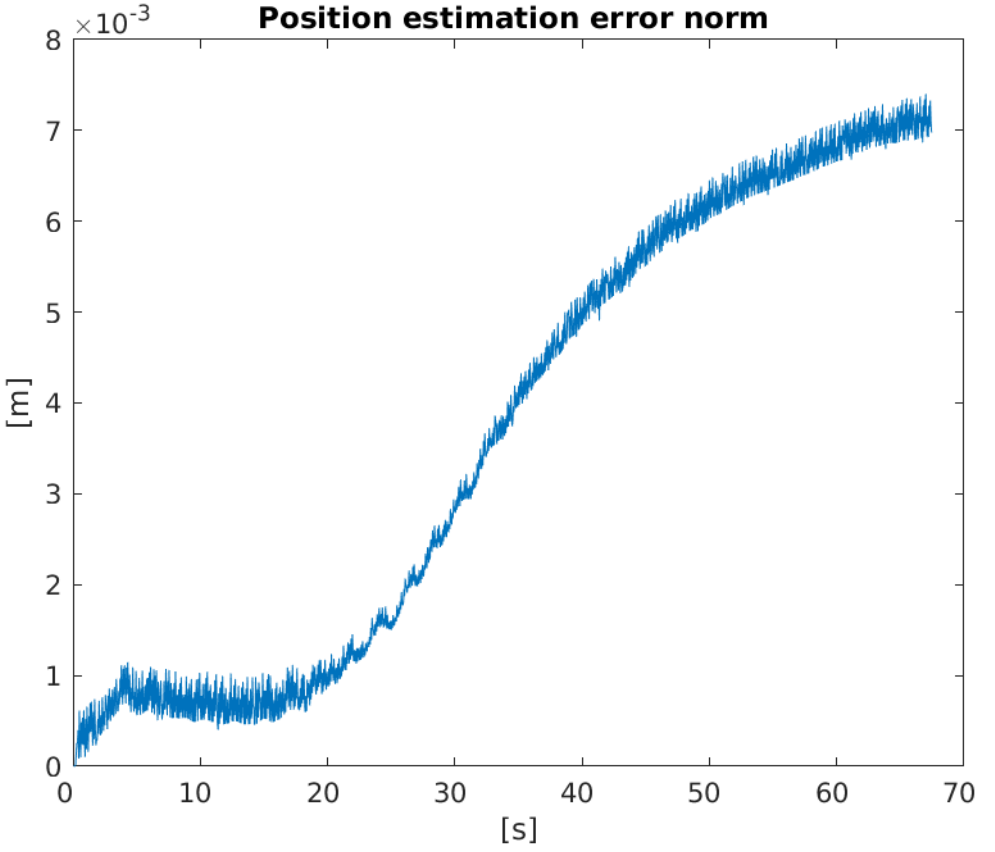


Figure 20 - Plot of the position error norm

The error norm on the position is calculated as $e = \sqrt{e_x^2 + e_y^2}$, where e_x and e_y are the errors on x and y respectively. The pose estimation's error norm is lower than $8 \times 10^{-3}\ m$ and it stays in this order throughout the simulation of the complete navigation. Below the plot of e_x and e_y

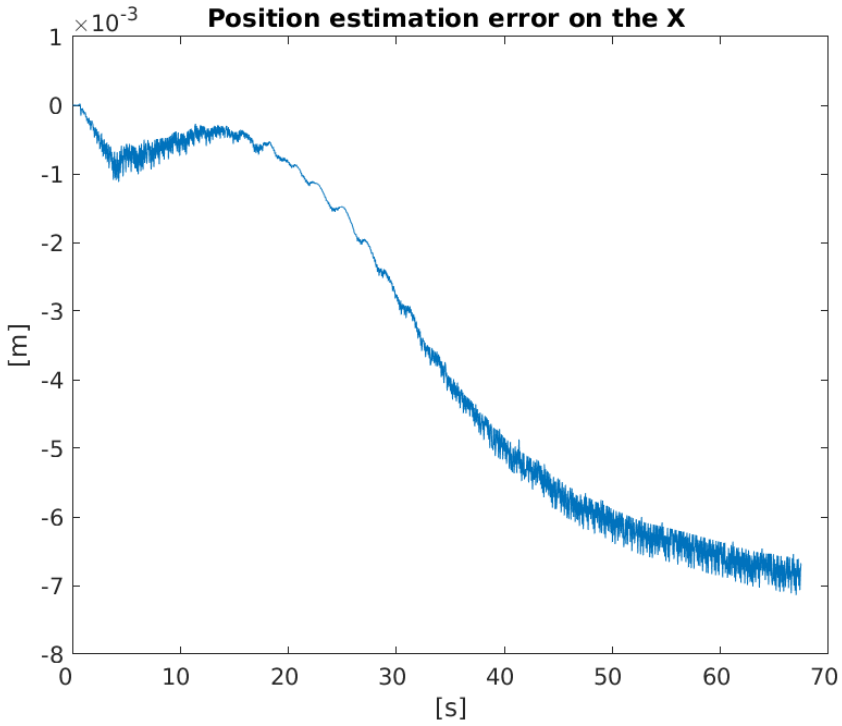


Figure 21 - Plot of the position error on the X axes

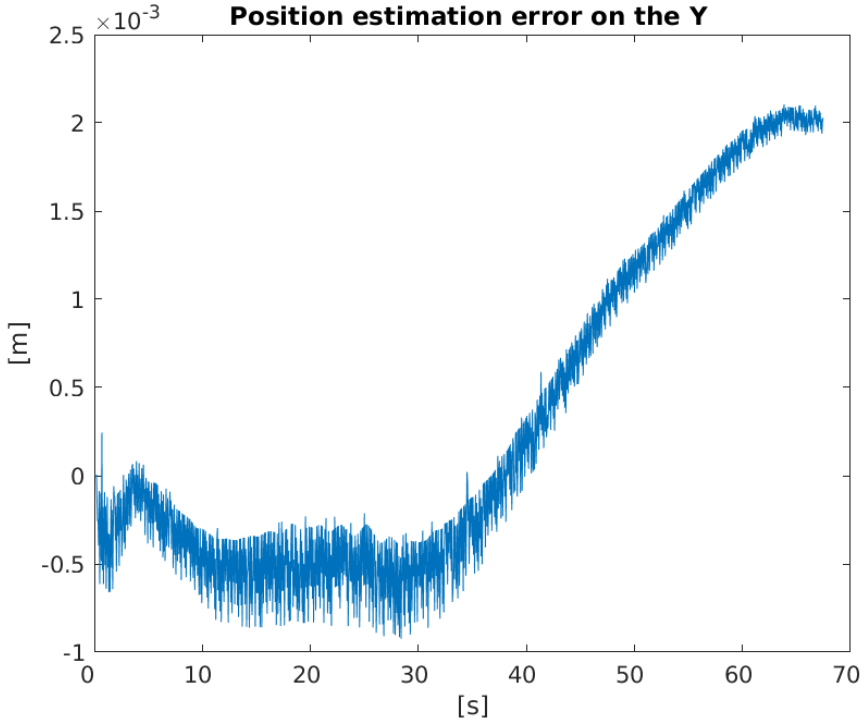


Figure 22 - Plot of the position error on the Y axes

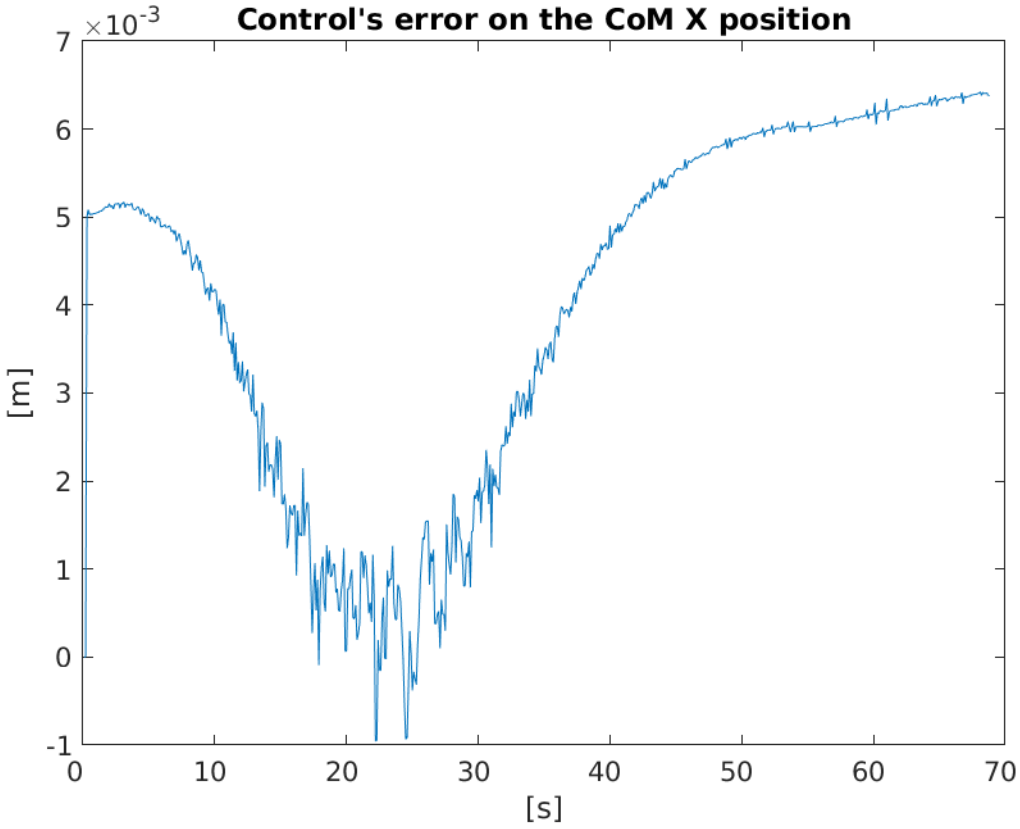


Figure 24 - Plot of the control's error on the CoM X position

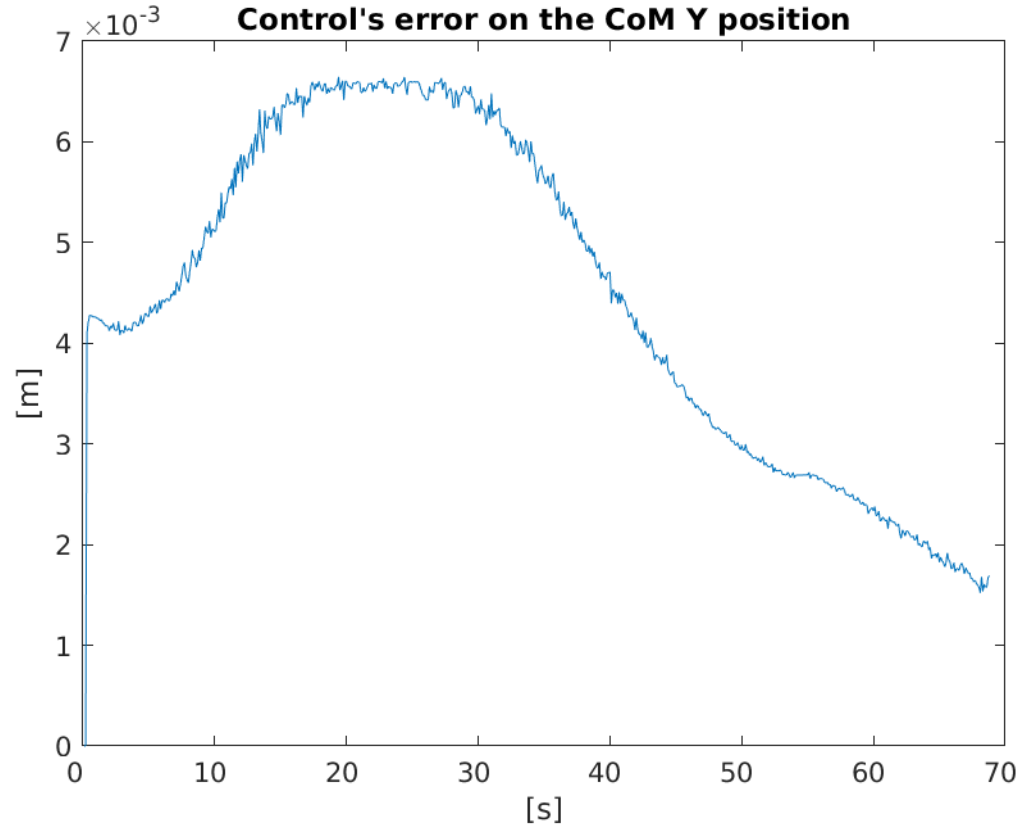


Figure 23 - Plot of the control's error on the CoM Y position

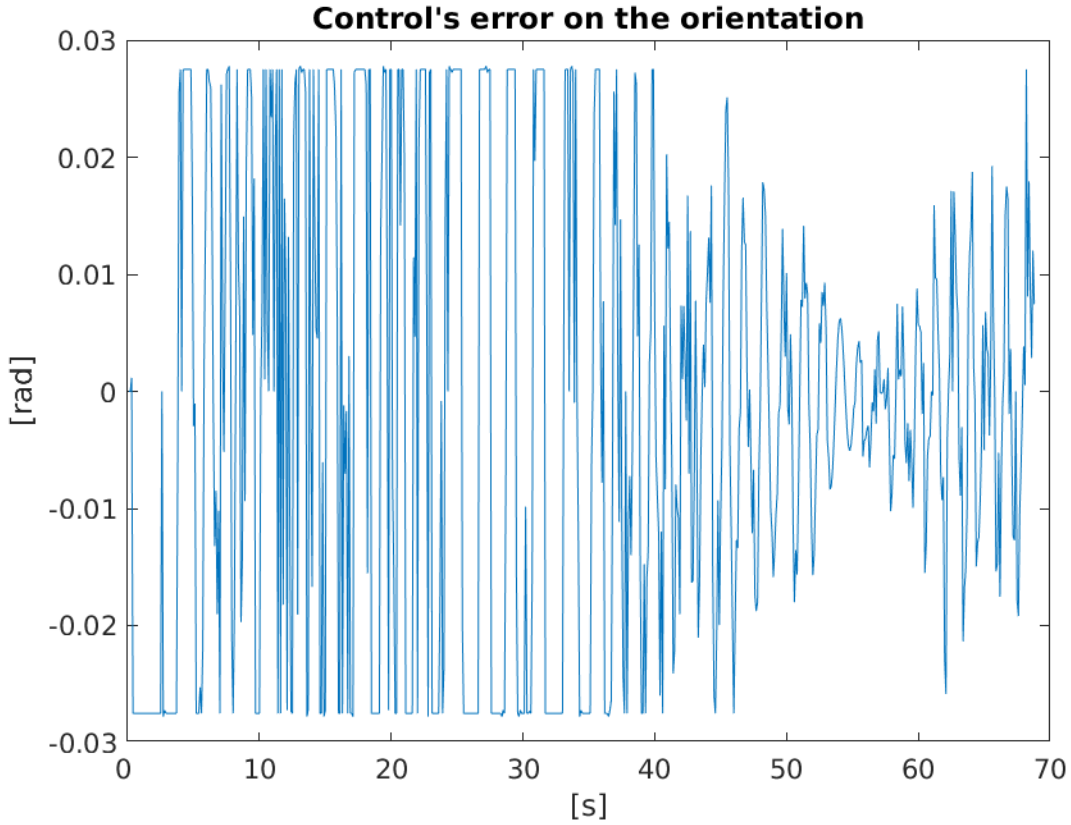


Figure 26 - Plot of the control's error on the CoM orientation

It is possible notice that the orientation error is below 2° and oscillates in the range of 0 . The position error on CoM position is kept below $7 \times 10^{-3}\ m$

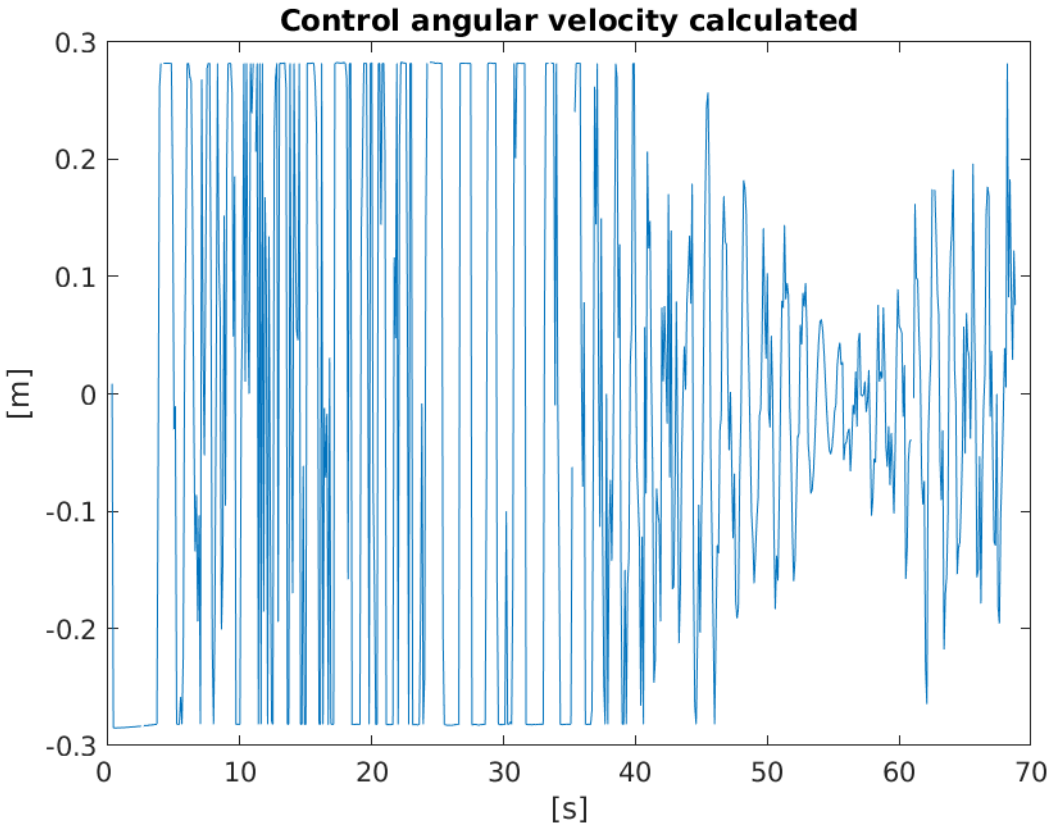


Figure 25 - Plot of the angular velocity elaborated by the control loop

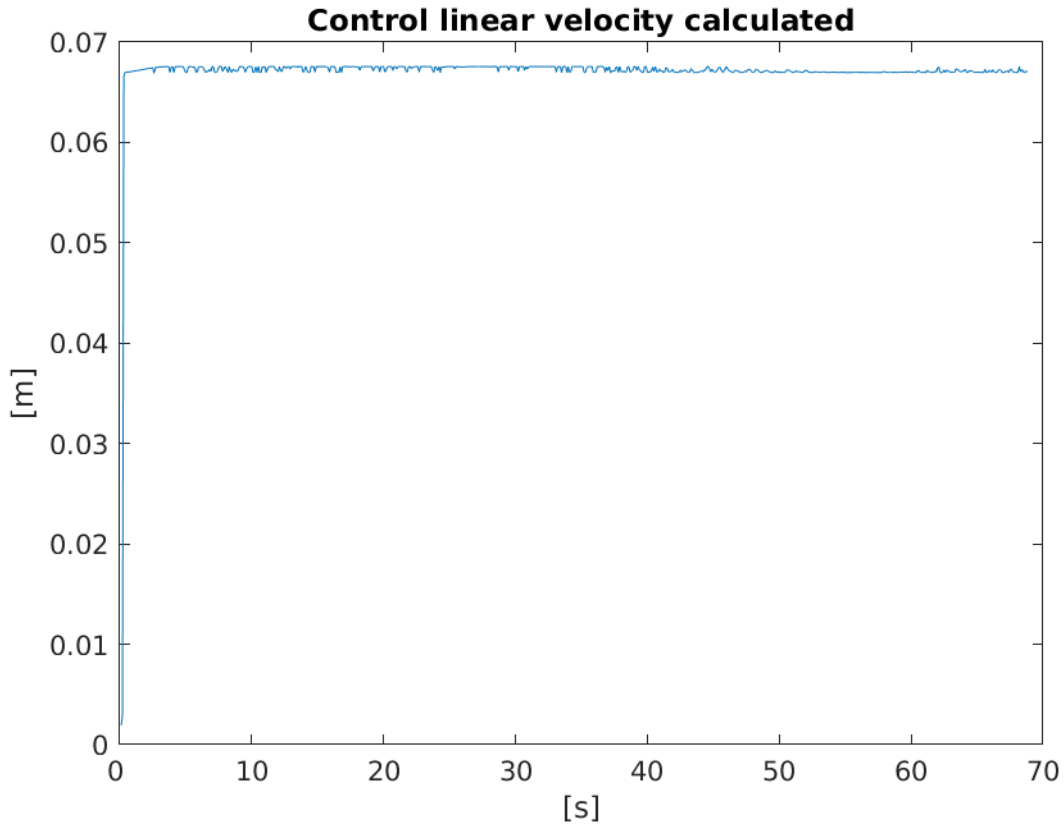


Figure 28- Plot of the linear velocity elaborated by the control loop

The linear and the angular velocities calculated by the control oscillates under the max velocities imposed. In the follow the control signal elaborated for the wheel's joint.

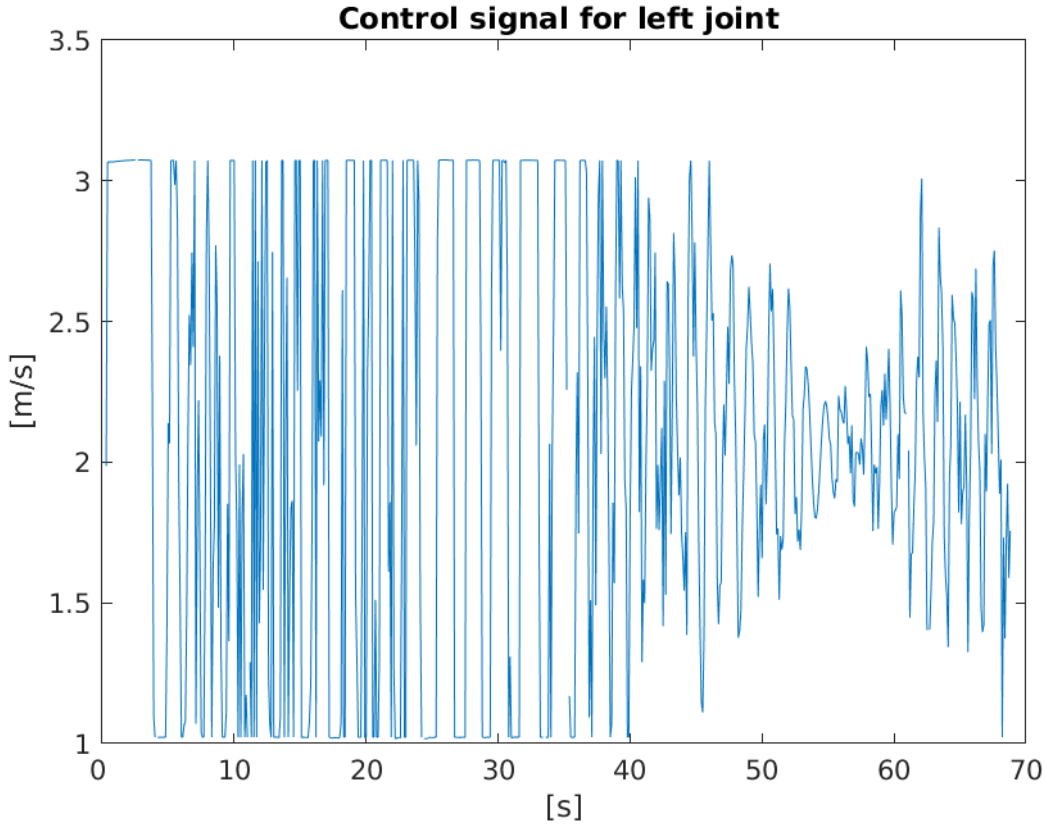


Figure 27 - Plot of the command sent to the left joint

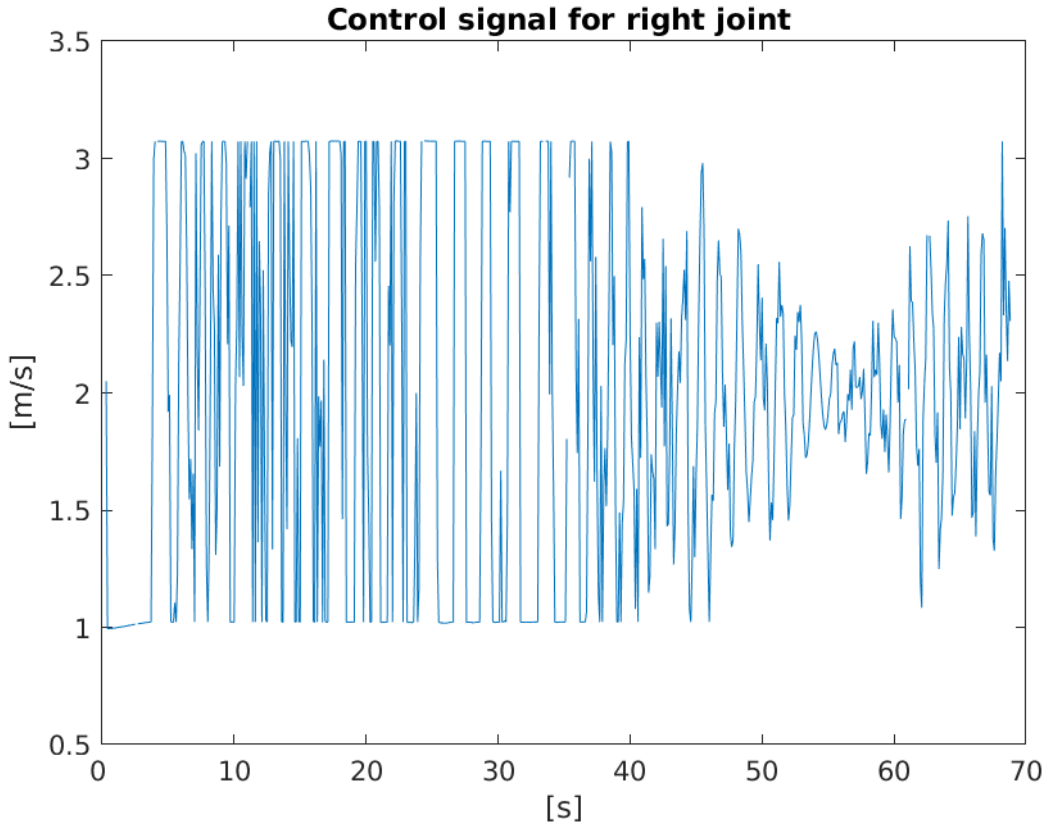


Figure 29 - Plot of the command sent to the right joint

Results of the simulation on the empty environment

In the simulation made on the empty world the robot follows the trajectory in figure 30.

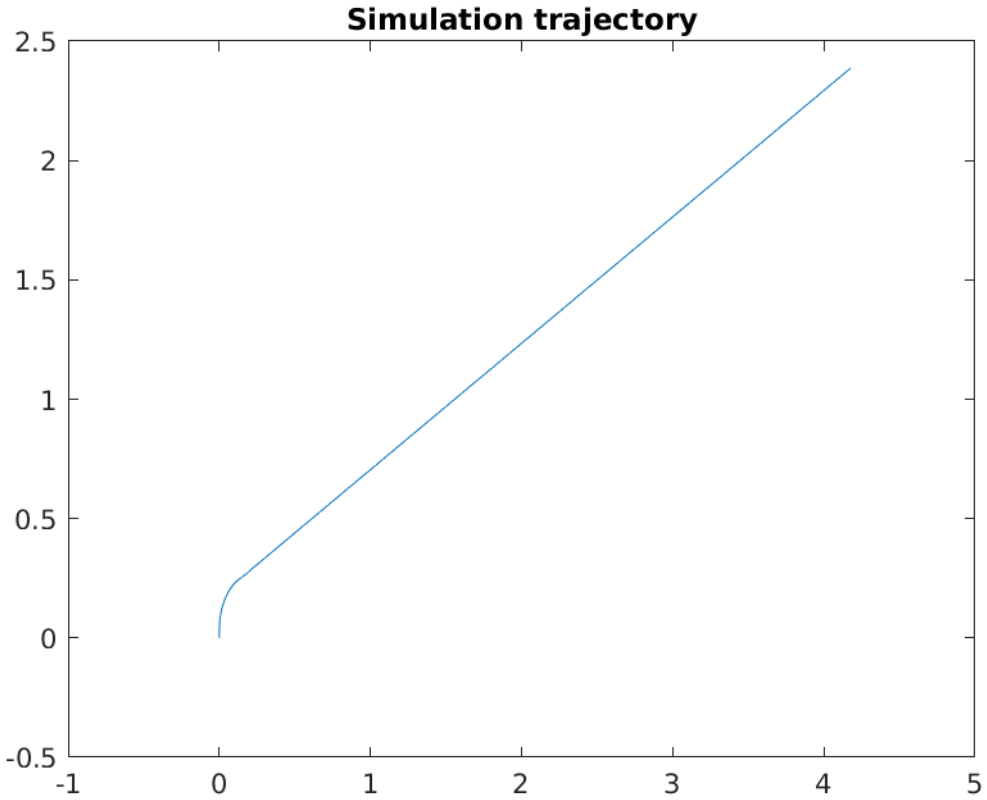


Figure 30 - Trajectory done in the empty world

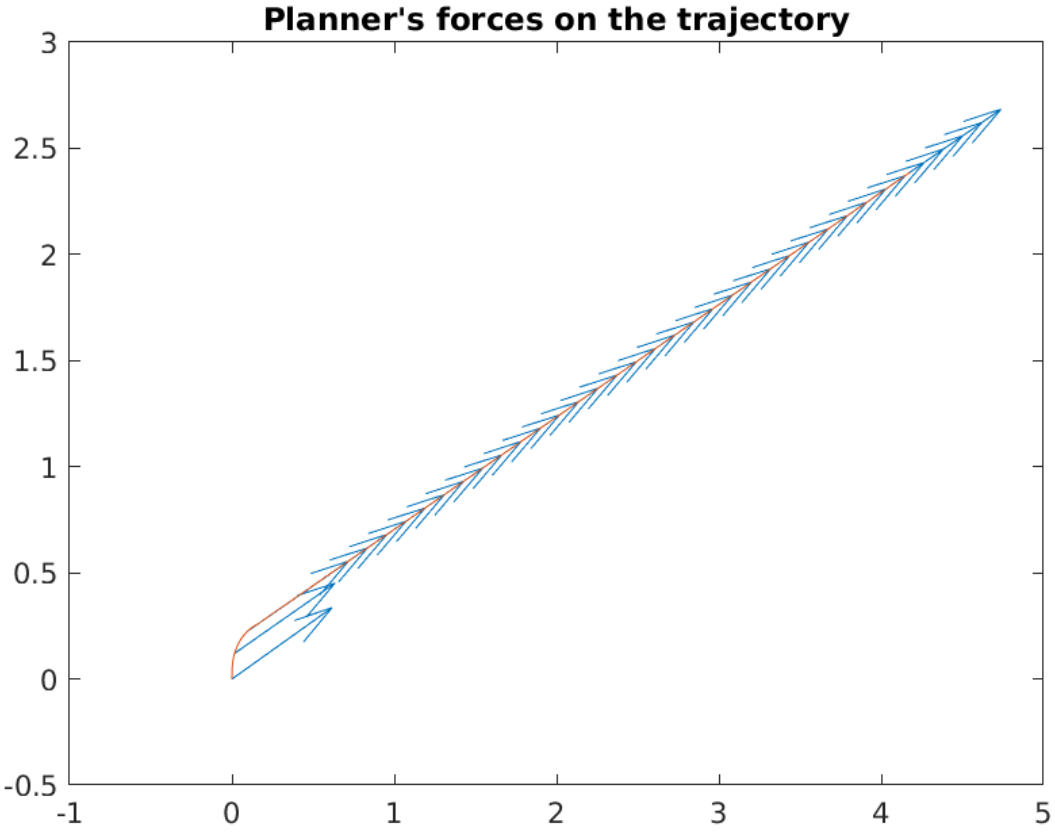


Figure 31 - Planner's forces along the trajectory

The pose estimation's error is lower then the previous case.

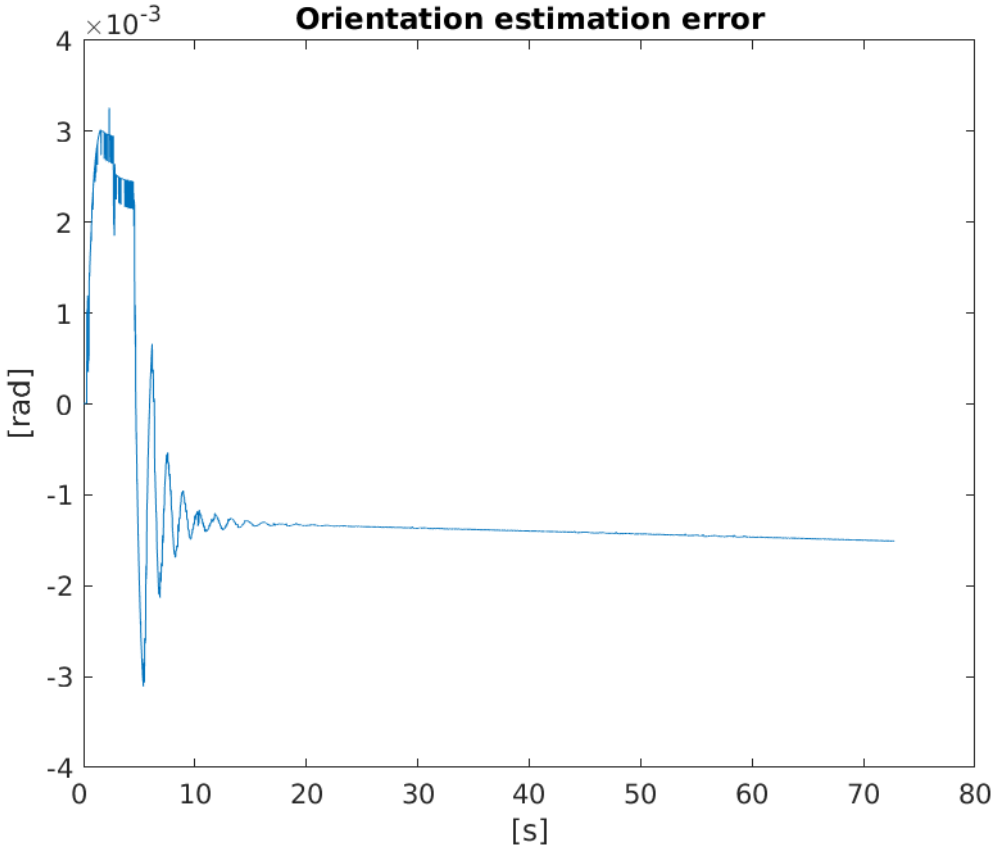


Figure 33 - Plot of the error of the yaw estimation

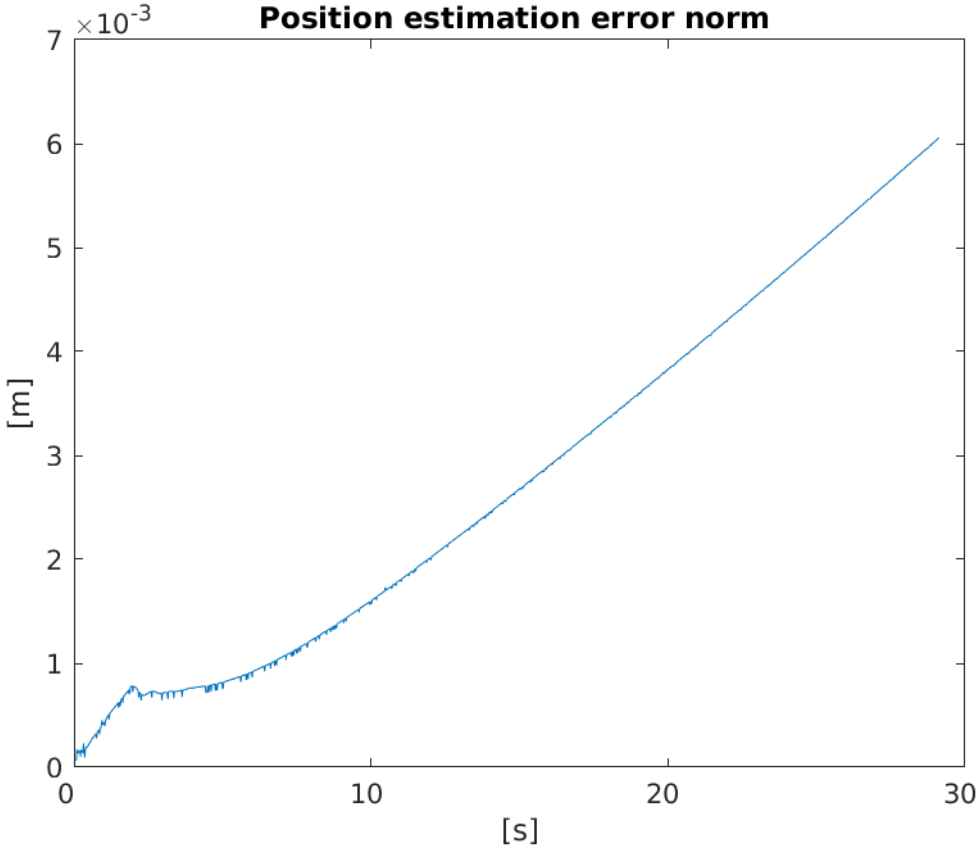


Figure 32- Plot of the error on the position estimation

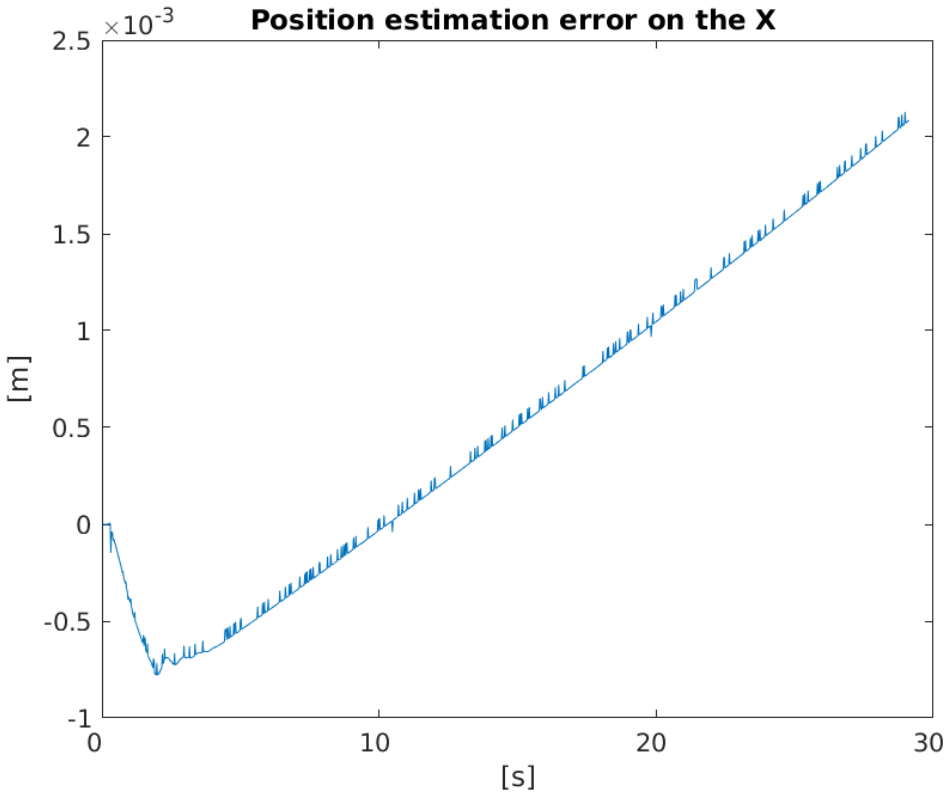


Figure 34 -Plot of the error on the position estimation along X

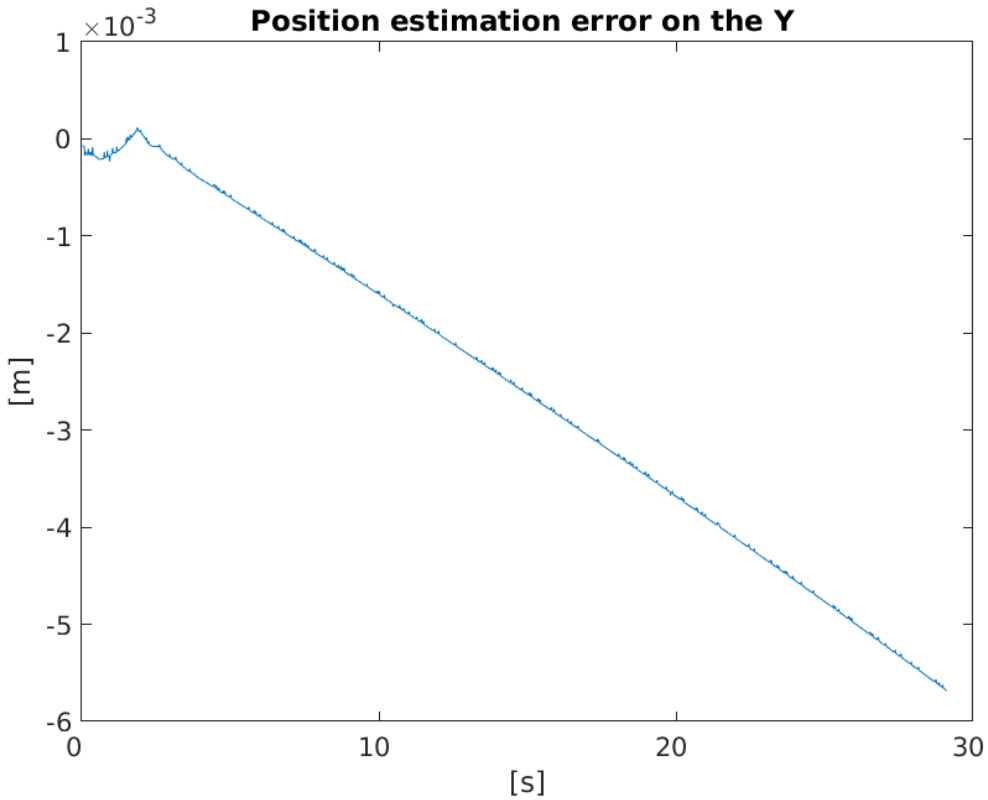


Figure 35 - Plot of the error on the position estimation along Y

As the force calculated by the planner node are lower, the errors on the pose estimation oscillate less.

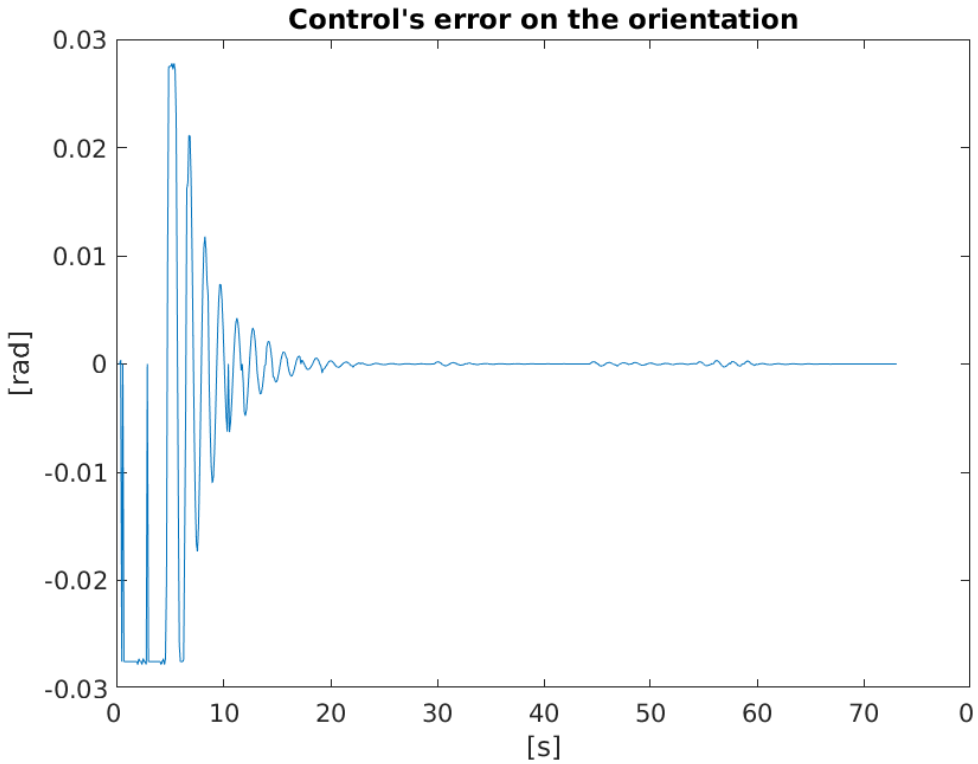


Figure 37 - Control's error on the robot's yaw

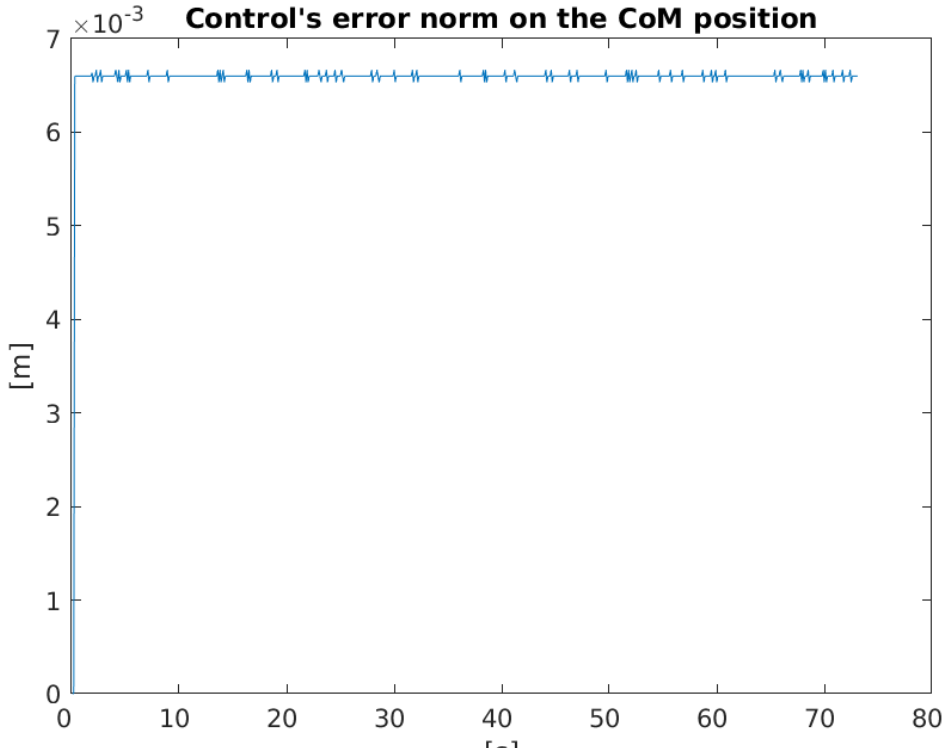


Figure 36 - Control's error on the CoM position

Because the trajectory that the robot must follow is a straight line, the control's error on the yaw converges to 0, while the error on the CoM's position is below $7 \times 10^{-3}\, m$ like the previous case. Below the plots of the control's command sent to the wheels.

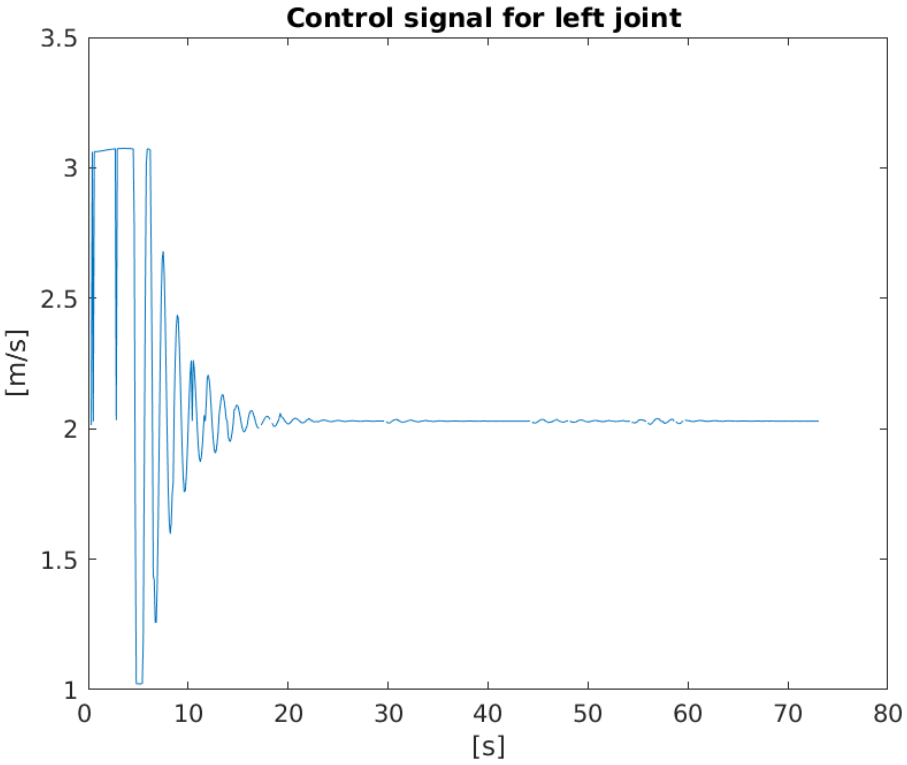


Figure 38 - Command sent to the left joint

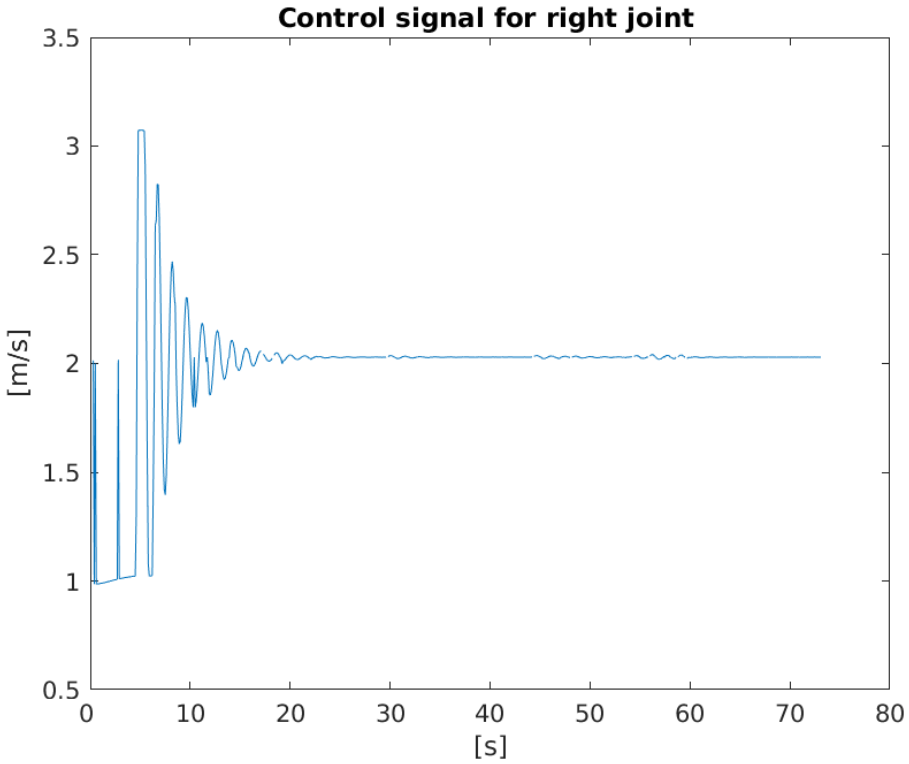


Figure 39 - Command sent to the right joint

Bibliography

[1] https://github.com/goncabrita/roomba_robot

[2] https://github.com/osrf/gazebo_models

[3] De Luca A., Oriolo G., Vendittelli M. (2001) Control of Wheeled Mobile Robots: An Experimental Overview. In: Nicosia S., Siciliano B., Bicchi A., Valigi P. (eds) Ramsete. Lecture Notes in Control and Information Sciences, vol 270. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45000-9_8

[4] https://github.com/ros-perception/vision_opencv

[5] https://github.com/ros-drivers/zbar_ros