PDEEC@FEUP: Mobile Robotics
2020/2021: 2nd Semester

A Pose Control Algorithm for Omnidirectional Robots

Student: Ricardo B. Sousa[1,2]
Supervisors: António P. Moreira[1,2], Héber Miguel Sobreira[2]

Professor: Paulo G. Costa[1,2]

[1] Faculty of Engineering of the University of Porto
(email: `up201503004@edu.fe.up.pt`, `amoreira@fe.up.pt`)
[2] INESC TEC – Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
(email: `ricardo.b.sousa@inesctec.pt`, `heber.m.sobreira@inesctec.pt`)

Abstract. The controller algorithm on a robot must guarantee that the robot gets to the desired goal at the desired time instant without deviating from the generated trajectory. Omnidirectional robots are of great interest due to their complete maneuverability. So, this report formulates a pose control algorithm for omnidirectional robots. We use Proportional-Integrative (PI), Proportional-Derivative (PD), and Feed-Forward(FF) controllers to control the robot's pose in both space and time. The proposed controller approximates the future trajectory (a subset of points from the trajectory between the start and desired poses) on parametric polynomials using the linear least-squares algorithm for computing the first and second-order derivatives needed in the FF. Furthermore, the controller was integrated as a local planner plugin into the `move_base` ROS package. The controller was tested with the trajectories generated by the `global_planner` algorithm using the A* shortest path algorithm. The results in both simulation and real environments demonstrate that the proposed controller leads to a near-zero steady-state error for ramp inputs, while having predictive characteristics due to the FF's derivatives computation.

1 Introduction

In mobile robotics, it is not only important for the robot to reach the desired pose (position and orientation) but also how and when the robot gets there. The best case is when the robot can follow any trajectory through its workspace of poses. Omnidirectional robots have complete maneuverability due to being capable of moving in all directions at any time. So, these robots are more suitable for dynamic environments in comparison to other steering geometries (e.g., differential-drive or tricycle). The translation and rotation components of omnidirectional steering geometries are independent of each other. One popular use of omnidirectional robots is in robot soccer games of the RoboCup competition and, nowadays, these robots are used more and more in industrial applications [1,2].

The control of omnidirectional robots is crucial to control the pose of the robot over time. Several works focused on characterizing nonlinear parameters and uncertain factors of these robots. These approaches are related to adaptive control [3], sliding mode control [4,3], fuzzy control [5,2,6], and neural networks [7]. However, the consideration of nonlinearities increases the complexity of implementing these works. Other works use more classical approaches based on PI, PD, and/or PID controllers [8,9]. Two disadvantages of these approaches are the overshoot being dependent on the robot's dynamics (when sudden changes happen in the position reference) and the existence of a possible delay between the reference and actual pose of the robot.

In addition to controlling the robot, it is necessary to generate a trajectory to achieve the desired goal pose. Considering an occupancy grid to describe the environment, a graph describes the connection between the different cells. The advantage of this approach is the possibility to use known graph shortest-path algorithms to obtain the set of points that describe the trajectory between the robot's current pose and the desired goal. So, algorithms such as Dijkstra's [10] and A* [11] can be used to search this shortest path.

For robot applications, the Robot Operation System (ROS) [12] is more and more used both in industry and academy. Its modular and distributed architecture allows using ROS packages available in the community while developing other ones to satisfy the desired requirements of an application. In terms of navigating a mobile robot, the open-source `move_base` [13] ROS package implements an action that, given the desired goal, will attempt to reach the desired pose. This ROS package allows the development of specific plugins for the global and local planner while using other plugins to reach the desired goal. The first planner is relative to generating the trajectory in the global map frame. The second one mainly focused on avoiding obstacles.

This work is intended for controlling the pose of omnidirectional robots given a set of points in space with timestamps (position, orientation, and time) associated with each one. The controller is implemented as a plugin adhering to the local panner's base interface of `move_base` [13]. The pose control uses PI controllers for the angular speed of the wheels, PD for positioning control, and Feed-Forward (FF) controllers to mitigate the delay between the robot's reference and actual pose when using only PD controllers. Also, the pose control uses a subset of points from the desired trajectory as future poses of the robot to compute the derivatives for the FF controllers. Then, the `global_planner` [14] plugin is used to generate the trajectories for the robot achieving the desired goal. This planner can be configured to use either the Dijkstra's [10] or A* [11] algorithms for searching a path to the goal. The experimental results demonstrate that the proposed controller leads to not only the robot following the trajectory on the desired time instants while predicting sudden changes in the pose's reference of the robot.

Our previous work [15] is the basis of the control algorithm for omnidirectional robots. The control algorithm was only tested previously in a simulation environment. The work presented in the report extends the results to a three-wheeled omnidirectional robot of the 5DPO Middle Size League (MSL) robot soccer team [16].

The report is organized as follows. Section 2 formulates the trajectory definition taken into account in this work. Section 3 formulates a classical approach to pose control using only PI and PD controllers. Section 4 presents the modifications made to the classical approach based on FF controllers. Section 5 explains the use of the `move_base` [13] ROS package in this work. Section 6 describes the architecture implemented on the ROS framework for a three-wheeled omnidirectional controller. Section 7 presents the experimental results obtained from a simulation environment and with a three-wheeled omnidirectional robot. Finally, section 8 presents the conclusions taken from this work.

2 Trajectory Definition

2.1 Type of trajectory

Normally, a planner of trajectories defines these as a set of points, parametric splines, or polynomials. In this work, we considered a trajectory T as a set of points P_k defined both in space (position and orientation of the robot) and in time: $T = \{P_0\left(t_0\right), P_1\left(t_1\right), \ldots, P_{N-1}\left(t_{N-1}\right)\}$, where $P_k\left(t_k\right) = \{P_{X,k}, P_{Y,k}, P_{\theta,k}\}\left(t_k\right)$. Even if curves such as parametric splines are used, they can be discretized to obtain a set of points with timestamps defined for each point.

The set of points in time can be characterized in three different ways. The most general alternative is defining the points without any restrictions in space or in time. Another approach is maintaining a certain distance between consecutive points. The one used in this paper is restricting the time interval as a constant value between consecutive points. However, the proposed controller works also with the first two approaches.

2.2 Coordinate frame of the trajectory

A trajectory could be relative to the world's coordinate frame ($\{X^W, Y^W\}$) or to the robot's local frame ($\{X^R, Y^R\}$). These two different alternatives are illustrated in figure 1 for the point P_k, where P_k^W represents the point's desired pose relative to the world and P_k^R to the robot. The orientation matrix $R^{-1}(\theta_{rb}^W)$ defined in equation 1 ($\cos(\theta_{rb}^W)$ and $\sin(\theta_{rb}^W)$ are represented by $\mathrm{c}_{\theta_{rb}^W}$ and $\mathrm{s}_{\theta_{rb}^W}$, respectively) is the orientation of the world relative to the robot's frame depending on the robot's current orientation (θ_{rb}^W). Then, the transformation of the trajectory from the world to the robot's local frame can be defined as in equation 2.

$$R^{-1}(\theta_{rb}^W) = \begin{bmatrix} \mathrm{c}_{\theta_{rb}^W} & \mathrm{s}_{\theta_{rb}^W} & 0 \\ -\mathrm{s}_{\theta_{rb}^W} & \mathrm{c}_{\theta_{rb}^W} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} P_{X,k}^R \\ P_{Y,k}^R \\ P_{\theta,k}^R \end{bmatrix} = R^{-1}(\theta_{rb}^W) \cdot \left[\begin{bmatrix} P_{X,k}^W \\ P_{Y,k}^W \\ P_{\theta,k}^W \end{bmatrix} - \begin{bmatrix} X_{rb}^W \\ Y_{rb}^W \\ \theta_{rb}^W \end{bmatrix} \right] = \begin{bmatrix} (P_{X,k}^W - X_{rb}^W) \cdot \mathrm{c}_{\theta_{rb}^W} + (P_{Y,k}^W - Y_{rb}^W) \cdot \mathrm{s}_{\theta_{rb}^W} \\ (-P_{X,k}^W + X_{rb}^W) \cdot \mathrm{s}_{\theta_{rb}^W} + (P_{Y,k}^W - Y_{rb}^W) \cdot \mathrm{c}_{\theta_{rb}^W} \\ P_{\theta,k}^W - \theta_{rb}^W \end{bmatrix} \tag{2}$$

One advantage of the trajectory being relative to the robot's local frame is that ${P_k}^R$ is the pose error relative to the current robot's pose. Moreover, the motion of omnidirectional robots can be controlled independently in terms of translation and rotation [1,2]. The translation part is possible to decouple in the directions of X^R and Y^R. Thus, $P_{X,k}^R$, $P_{Y,k}^R$, and $P_{\theta,k}^R$ represent the error components in the direction of X^R, Y^R, and on the orientation of the robot.

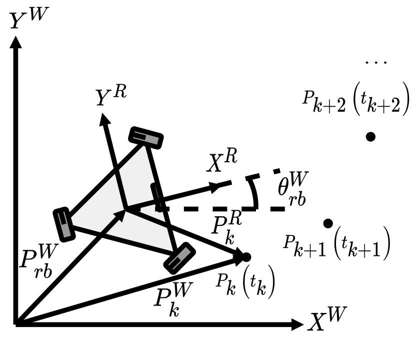


Fig. 1. Trajectory relative to the world and robot's coordinate frames

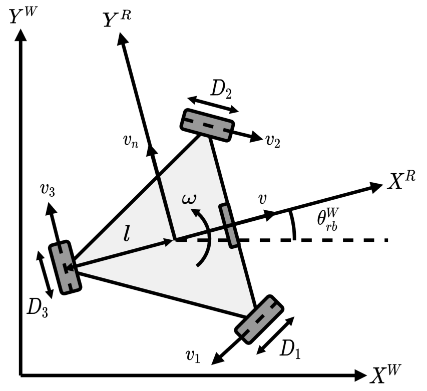


Fig. 2. Three-wheeled omnidirectional robot

3 A Classical Approach to Pose Control

In this section, it is proposed a pose control system for omnidirectional robots (the three-wheeled robot used in the simulation is illustrated in figure 2) using PI controllers for the angular speed of the motors and PD controllers for the robot's pose relative to its own frame. Even though the formulation is focused on the three-wheeled robot illustrated in figure 2, the only adaptation for other omnidirectional robots is redefining their inverse differential kinematics.

3.1 Inverse differential kinematics

First, it is necessary to characterize the inverse differential kinematics of omnidirectional robots to control their velocity through a specific trajectory. Given the linear (v and v_n along the directions of X^R and Y^R, respectively) and angular (ω) velocity desired for the robot, equation 3 computes the linear velocity (v_i) of each wheel i (considering also the distance l between the wheels and the robot's geometric center) [1]. The angular speed of each motor ($\dot{\varphi}_i$) is computed depending on the gear's reduction ratio (n) and on the diameter of the wheels (D_i), as computed in equation 4.

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} -\frac{\sqrt{3}}{2} & -\frac{1}{2} & -l \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} & -l \\ 0 & 1 & -l \end{bmatrix} \cdot \begin{bmatrix} v \\ v_n \\ \omega \end{bmatrix} \tag{3}$$

$$\dot{\varphi}_i = \frac{2n}{D_i} \cdot v_i \tag{4}$$

3.2 Angular speed control of the wheels

Next, the angular speed of the motors ($\dot{\varphi}_i$) can be controlled using a PI controller to set the motors' input voltage. This controller implemented in the simulation was tuned using the Internal Model Control (IMC) method [17]. Given experiments performed with a real robot and in simulation, the system can be considered as a first-order system. So, [17] requires the output gain ($K_{p,\dot{\varphi}}$), the time constant ($\tau_{\dot{\varphi}}$), and the lag ($L_{\dot{\varphi}}$) to be estimated. The PI parameters

Algorithm 1: Hammerstein Nonlinear Block

```
input  : $V_{in,i}$,$V_{d,i}$,$V_{0,i}$
output: $V_{mot,i}$
1 if $V_{in,i} > V_{d,i}$ then
2     $V_{mot,i} = (V_{in,i} - V_{d,i}) + V_{0,i}$
3 else if $V_{in,i} > -V_{d,i}$ then
4     if $V_{d,i} \neq 0$ then $V_{mot,i} = V_{in,i} \cdot V_{0,i}/V_{d,i}$
5     else $V_{mot,i} = 0$
6 else
7     $V_{mot,i} = (V_{in,i} + V_{d,i}) - V_{0,i}$
```

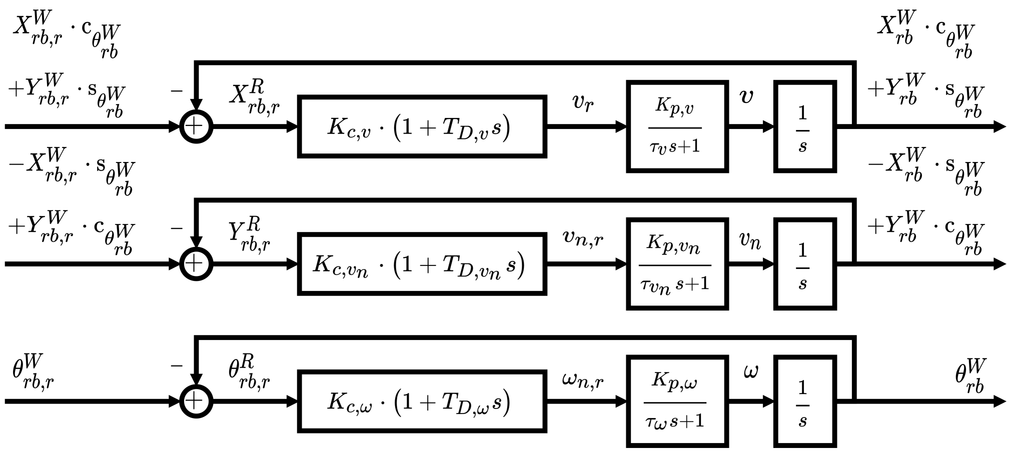

$X^W_{rb,r} \cdot c_{\theta^W_{rb}}$
$+Y^W_{rb,r} \cdot s_{\theta^W_{rb}}$
$-$
$X^R_{rb,r}$
$K_{c,v} \cdot (1 + T_{D,v}s)$
v_r
$\frac{K_{p,v}}{\tau_v s+1}$
v
$\frac{1}{s}$
$X^W_{rb} \cdot c_{\theta^W_{rb}}$
$+Y^W_{rb} \cdot s_{\theta^W_{rb}}$
$-X^W_{rb,r} \cdot s_{\theta^W_{rb}}$
$+Y^W_{rb,r} \cdot c_{\theta^W_{rb}}$
$Y^R_{rb,r}$
$K_{c,v_n} \cdot (1 + T_{D,v_n}s)$
$v_{n,r}$
$\frac{K_{p,v_n}}{\tau_{v_n} s+1}$
v_n
$-X^W_{rb} \cdot s_{\theta^W_{rb}}$
$+Y^W_{rb} \cdot c_{\theta^W_{rb}}$
$\theta^W_{rb,r}$
$\theta^R_{rb,r}$
$K_{c,\omega} \cdot (1 + T_{D,\omega}s)$
$\omega_{n,r}$
$\frac{K_{p,\omega}}{\tau_\omega s+1}$
ω
θ^W_{rb}

Fig. 3. PD controllers for the robot's pose

(the proportional gain $K_{c,\dot{\varphi}}$ and integration time $T_{I,\dot{\varphi}}$) are computed using equation 5, given a desired time constant for the closed-loop ($\tau_{cl,\dot{\varphi}}$).

$$\begin{cases} K_{c,\dot{\varphi}} = \frac{1}{K_{p,\dot{\varphi}}} \cdot \frac{\tau_{\dot{\varphi}}}{\tau_{cl,\dot{\varphi}} + L_{\dot{\varphi}}} \\ T_{I,\dot{\varphi}} = \tau_{\dot{\varphi}} \end{cases} \tag{5}$$

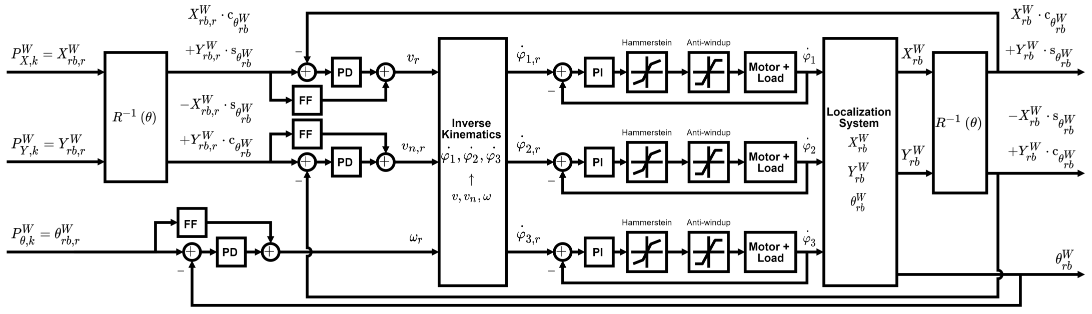
Considering that the motors have a dead zone, a Hammerstein nonlinear block (described in the algorithm 1) was used to compensate for the existence of this zone. If the wheel/motor (with the robot on the ground) starts to rotate at a certain voltage V_0, V_d would be the new dead zone.

Finally, the windup effect is compensated by limiting the voltage computed from the PI controllers and the Hammerstein block to the maximum value supported by the motors. When the voltage exceeds these limits, the integration part of the PI controller remains unchanged.

3.3 Position control

Similarly to the angular speed control, the evolution of $v/v_n/\omega$ relative to their reference ($v_r/v_{n,r}/\omega_r$) resembles a first-order system (illustrated in figure 3). The integration of v, v_n, and ω estimates the robot's pose on a coordinate frame aligned with the local frame but with the same origin as the world frame. The error of the reference relative to the actual value of each component illustrated in figure 3 is the reference for the robot's pose on its local frame. So, the control of v_r, $v_{n,r}$, and ω_r with a PD controller is independent from each other.

The parameters of the PD controllers (the proportional gain $K_{c,j}$ and the derivative time $T_{D,j}$, where $j = v, v_n, \omega$) depend on the desired closed-loop poles. These poles can be chosen considering the roots of normalized Bessel polynomials corresponding to a settling time ($T_{sett.,j}$) of 1 second. Given the consideration of first-order systems and that the position control is a second-order system, the poles defined by Bessel polynomials are $p = -4.0530 + j2.3400$ and $p^* = -4.0530 - j2.3400$. The closed-loop characteristic polynomial is defined in equation 6 for these two poles ($T_{sett.,j}$



$P^W_{X,k} = X^W_{rb,r}$
$P^W_{Y,k} = Y^W_{rb,r}$
$P^W_{\theta,k} = \theta^W_{rb,r}$
$R^{-1}(\theta)$
$X^W_{rb,r} \cdot c_{\theta^W_{rb}}$
$+Y^W_{rb,r} \cdot s_{\theta^W_{rb}}$
$-X^W_{rb,r} \cdot s_{\theta^W_{rb}}$
$+Y^W_{rb,r} \cdot c_{\theta^W_{rb}}$
PD
FF
v_r
$v_{n,r}$
ω_r
Inverse Kinematics
$\dot{\varphi}_1, \dot{\varphi}_2, \dot{\varphi}_3$
$\uparrow$
v, v_n, ω
$\dot{\varphi}_{1,r}$
$\dot{\varphi}_{2,r}$
$\dot{\varphi}_{3,r}$
PI
Hammerstein
Anti-windup
Motor + Load
$\dot{\varphi}_1$
$\dot{\varphi}_2$
$\dot{\varphi}_3$
Localization System
X^W_{rb}
Y^W_{rb}
θ^W_{rb}
$R^{-1}(\theta)$
$X^W_{rb} \cdot c_{\theta^W_{rb}}$
$+Y^W_{rb} \cdot s_{\theta^W_{rb}}$
$-X^W_{rb} \cdot s_{\theta^W_{rb}}$
$+Y^W_{rb} \cdot c_{\theta^W_{rb}}$
θ^W_{rb}

Fig. 4. Proposed trajectory controller for omnidirectional robots

should be the lowest value possible considering the limits of the discrete real system, e.g., the control period), and it is defined in equation 7 for the PD controller and the systems' models. Then, equation 8 defines the PD parameters.

$$s^2 - \frac{2 \cdot \mathrm{Re}\{p\}}{T_{sett.,j}} \cdot s + \frac{|p|^2}{{T_{sett.,j}}^2} = 0 \tag{6}$$

$$s^2 + \frac{K_{c,j} K_{p,j} T_{D,j} + 1}{\tau_j} \cdot s + \frac{K_{c,j} K_{p,j}}{\tau_j} = 0 \tag{7}$$

$$\begin{cases} K_{c,j} = \frac{\tau_j}{K_{p,j}} \cdot \frac{|p|^2}{{T_{sett.,j}}^2} \\ T_{D,j} = \frac{-\frac{2 \cdot \mathrm{Re}\{p\}}{T_{sett.,j}} \cdot \tau_j - 1}{K_{c,j} K_{p,j}} \end{cases} \tag{8}$$

However, as it is analyzed in section 7, the use of only PD controllers for pose control lead to a delay between the reference and the actual value of the controlled variables.

4 Pose Control with Feed-Forward Controllers

Therefore, we propose the use of Feed-Forward (FF) controllers to eliminate the delay between the reference and the actual value of the control variables. This controller does not affect the stability of the system. Also, it has predictive characteristics due to the requirement of the references' first and second derivatives. The proposed control system for the omnidirectional robot is illustrated in figure 4.

4.1 Feed-forward definition

Equation 9 defines the transfer function of a FF controller for the systems shown in figure 3. Note that the output of each FF controllers adds to the outputs of the PD controllers to compute the references for the linear (v_r and $v_{n,r}$) and angular (ω_r) velocities of the robot.

$$H_j(s) = \frac{\tau_j}{K_{p,j}} \cdot s^2 + \frac{1}{K_{p,j}} \cdot s \tag{9}$$

4.2 Computation of the derivatives

Given that our trajectory T is a set of points ($T = \{P_0(t_0), P_1(t_1), \ldots, P_{N-1}(t_{N-1})\}$), we already know the future poses desired for the robot. Consequently, the derivatives could be estimated considering these points in space and in time by approximating the set of points into parametric polynomials. Also, as already mentioned in section 2, the time interval between consecutive points is considered to be constant. In order to normalize this time interval, we assume that it is 1 on a time domain u. The transformation $t \rightarrow u$ is characterized by equation 10.

$$u = K_T \cdot t \tag{10}$$

So, first, we define the future trajectory as a subset of points F (where $F = \{P_{l+0}(u_{l+0}), \ldots, P_{l+M-1}(u_{l+M-1})\}$ and $F \subset T$) with M elements on the time domain u. These elements represent the future poses of the robot on the world frame relative to the current one. When the robot is reaching the desired pose ($u = u_{N-M}, u_{N-M+1}, ..., u_{N-1}$), the remaining poses of the future trajectory buffer are completed with the desired pose $P_{N-1}(t_{N-1})$. The derivates computation predict the approximation to the desired goal and beginning the deceleration of the robot. Next, the second derivative ($\ddot{f}_h$ where $h = X^W_{rb,r}, Y^W_{rb,r}, \theta^W_{rb,r}$), i.e., the acceleration, is computed using a second-degree polynomial approximation of the future trajectory, as illustrated in equation 11. The approximation is restricted by the initial position and velocity estimated based on the future trajectory, as defined in equation 12. Finally, equation 13 defines the least-squares solution with the Monroe-Penrose inverse matrix ($[\ldots]^\dagger$) for the coefficients $a_{h,2}$ (equivalent to the robot's acceleration).

$$\begin{aligned} f_h(u) &= a_{h,0} + a_{h,1}\cdot u + (1/2)\cdot a_{h,2}\cdot u^2 \\ \dot{f}_h(u) &= a_{h,1} + \cdot a_{h,2}\cdot u \\ \ddot{f}_h(u) &= a_{h,2} \end{aligned} \tag{11}$$

$$\begin{aligned} &\begin{cases} f_h(u_0) = a_{h,0} \\ \dot{f}_h(u_0) = a_{h,1} \end{cases} \\ \text{if } u = 0,1,...,M-1 \text{ then } &\begin{cases} a_{h,0} = P_{h,l+0} \\ a_{h,1} \approx P_{h,l+1} - P_{h,l+0} \end{cases} \end{aligned} \tag{12}$$

$$\begin{aligned} &\begin{bmatrix} P_{h,l+0} - a_{h,0} - a_{h,1}\cdot 0 \\ P_{h,l+1} - a_{h,0} - a_{h,1}\cdot 1 \\ \vdots \\ P_{h,l+M-1} - a_{h,0} - a_{h,1}\cdot (M-1) \end{bmatrix} = \begin{bmatrix} 0^2/2 \\ 1^2/2 \\ \vdots \\ (M-1)^2/2 \end{bmatrix} \cdot a_{h,2} \\ \Leftrightarrow a_{h,2} = &\begin{bmatrix} 0^2/2 \\ 1^2/2 \\ \vdots \\ (M-1)^2/2 \end{bmatrix}^\dagger \cdot \begin{bmatrix} P_{h,l+0} - a_{h,0} - a_{h,1}\cdot 0 \\ P_{h,l+1} - a_{h,0} - a_{h,1}\cdot 1 \\ \vdots \\ P_{h,l+M-1} - a_{h,0} - a_{h,1}\cdot (M-1) \end{bmatrix} \end{aligned} \tag{13}$$

Next, we estimate the first derivative ($\dot{g}_h$) with a first-order approximation polynomial illustrated in equation 14. The coefficients of the first-order are estimated using the least-squares algorithm, as illustrated in equation 15. An important note for both first and second-order approximations is that the orientation of F must be unwrapped to compute the linear approximations (i.e., without discontinuities). This step is required for avoiding wrong parameter estimations by the linear least-squares algorithm.

$$\begin{aligned} g_h(u) &= b_{h,0} + b_{h,1}\cdot u \\ \dot{g}_h(u) &= b_{h,1} \end{aligned} \tag{14}$$

$$\begin{bmatrix} P_{h,l+0} \\ P_{h,l+1} \\ \vdots \\ P_{h,l+M-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & (M-1) \end{bmatrix} \cdot \begin{bmatrix} b_{h,0} \\ b_{h,1} \end{bmatrix} \Leftrightarrow \begin{bmatrix} b_{h,0} \\ b_{h,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & (M-1) \end{bmatrix}^\dagger \cdot \begin{bmatrix} P_{h,l+0} \\ P_{h,l+1} \\ \vdots \\ P_{h,l+M-1} \end{bmatrix} \tag{15}$$

In figure 5, it is possible to visualize the difference between approximating a future trajectory ($M = 7$), a square corner, by a second-order polynomial with and without the restrictions of initial position and velocity. First, analyzing the approximations for the first point, it is clear that estimating the velocity and the acceleration from a second-order approximation without any restrictions would not predict correctly (at least, at an initial stage) the future behavior of the trajectory. Second, the velocity and acceleration vectors estimated with the proposed approach predicts correctly the behavior of the reference. On the robot's arrival to the square corner, the velocity starts to increase perpendicular to the arrival direction, as intended. On the exit of the corner, the derivatives estimate a zero acceleration due to the points being equidistant and the time interval between consecutive points a constant one, just as expected. Note that higher orders approximations would lead to oscillations on the approximated curves, and that would cause unwanted oscillations on the pose control. Also, the analysis of figure 5 is similar for the robot's orientation, but only on one dimension.

Lastly, if K_T is different from 1, the coefficients $a_{h,2}$ and $b_{h,1}$ relative to the u time domain should be divided by this scalar. Note that if we have points with a distance and a time interval between them of 0.05m and 1s, respectively, the velocity required from the robot is in average 0.05m/s with $K_T = 1$. For example, with $K_T = 2$, the velocity is

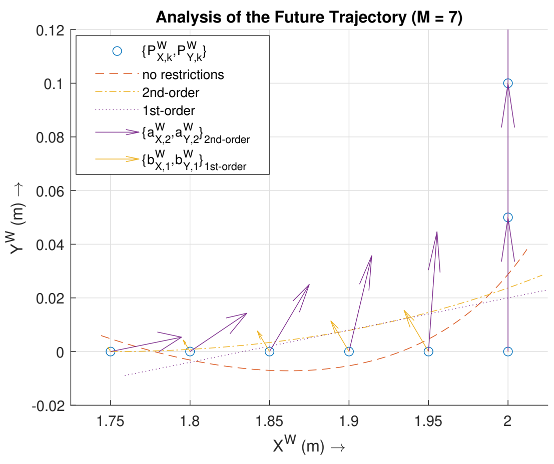


Fig. 5. Analysis of the future trajectory in XY on a square corner

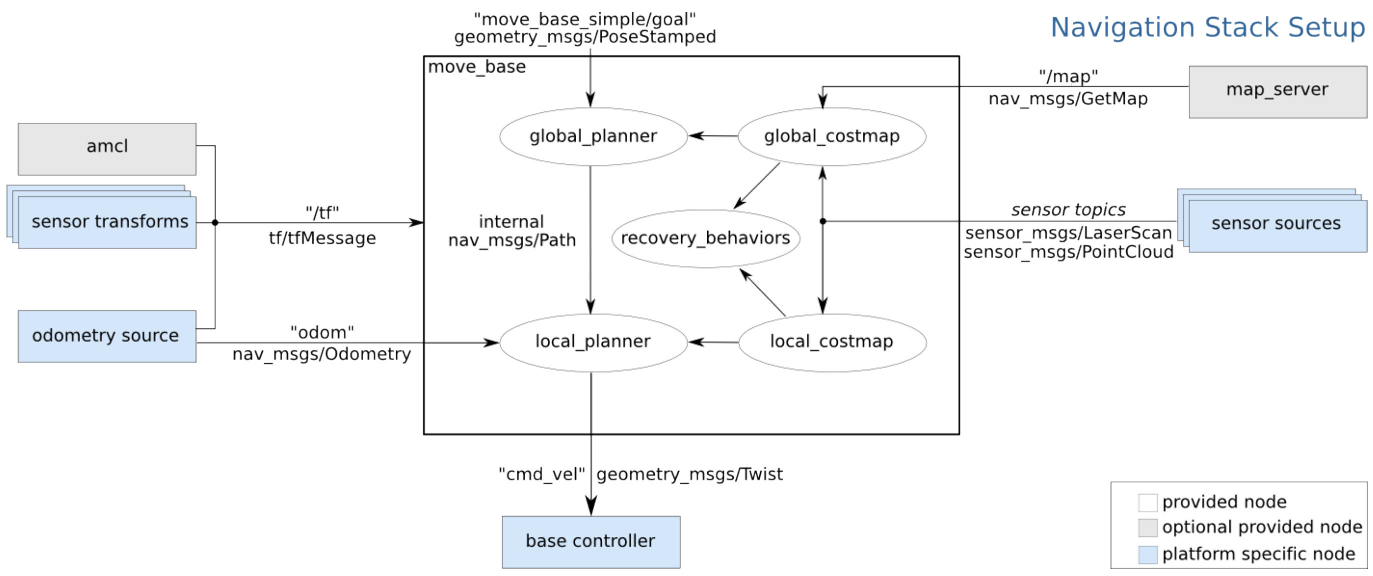


Fig. 6. Overview of the `move_base` ROS package [13]

increased by the same factor as K_T, i.e., the average will be 0.10m/s. Also, the derivatives $\ddot{f}_h$ and $\dot{g}_h$ are relative to the world frame. So, these derivatives must be multiplied by the matrix $R^{-1}(\theta^W_{rb})$ (equation 1) for the FF controllers formulated in this section correspond to the ones in figure 4.

5 ROS Package `move_base`

The proposed control algorithm was integrated into the `move_base` [13] ROS package to test in a real three-wheeled omnidirectional robot. An overview of the `move_base` [13] node is shown in figure 6. The node implements two planners: the `global_planner` and the `local_planner`. The first planner creates plans for the mobile robot. The second one usually implements the trajectory rollout and dynamic window approaches to avoid obstacles in a local occupancy grid map. Given that the `local_planner` is responsible to send the velocity commands (linear and angular ones) to the robot, the proposed control algorithm for omnidirectional robots was implemented as a plugin to the `local_planner`. This plugin had to adhere to the `nav_core::BaseLocalPlanner` interface for the `move_base` [13] node recognizing the developed plugin.

As for the `global_planner`, the plugin chosen for this work was the `global_planner` [14]. The main advantage of this plugin is being highly configurable (e.g., use or not of the grid path or selecting different search-path algorithms). Given that the robot available for the experiments only had available odometry data, the `move_base` [13] package was configured to run on the odometry frame. Even though it was necessary to create costmaps, these maps are empty because no obstacles or maps are detected or processed. The configuration considered in this work was the following

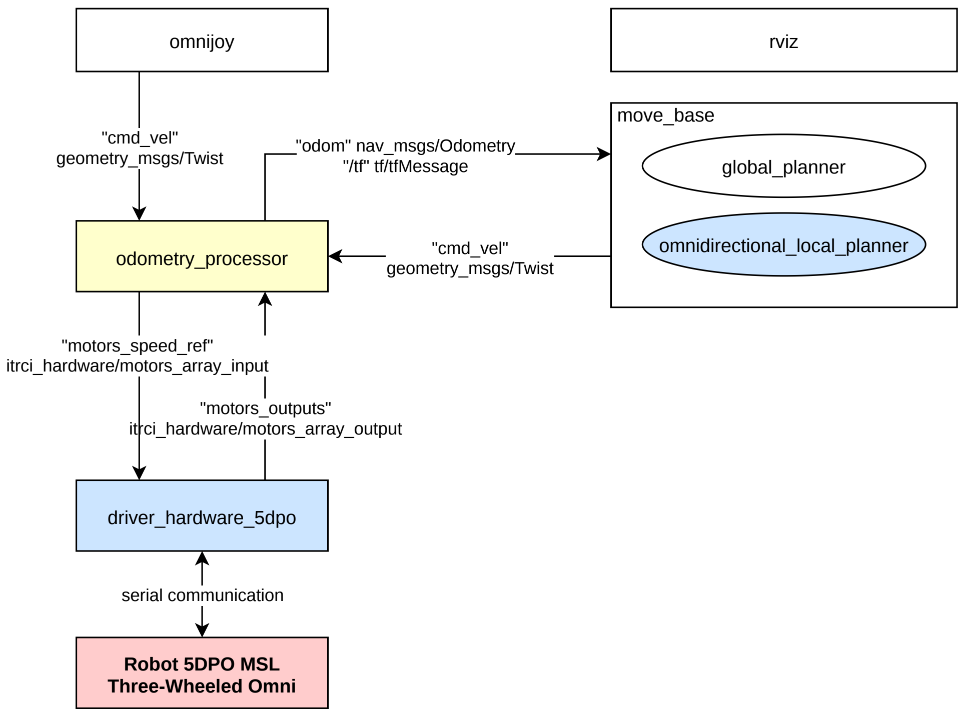

omnijoy
rviz
move_base
global_planner
omnidirectional_local_planner
"cmd_vel" geometry_msgs/Twist
"odom" nav_msgs/Odometry
"/tf" tf/tfMessage
odometry_processor
"cmd_vel" geometry_msgs/Twist
"motors_speed_ref" itrci_hardware/motors_array_input
"motors_outputs" itrci_hardware/motors_array_output
driver_hardware_5dpo
serial communication
Robot 5DPO MSL Three-Wheeled Omni

Fig. 7. Overview of the implemented architecture for the 5DPO [16] MSL omnidirectional robot

one: A* [11] search-path algorithm, interpolation mode in orientation (linear interpolation of the start and goal poses), and using the grid path (path follows the grid boundaries). The last was used to test trajectories other than straight-lines. Without the grid boundaries, the `global_planner` [14] would give mainly straight-lines due to the empty maps not being useful to test the predictive characteristics of the controller to sudden changes in the robot's pose reference.

6 Architecture

The architecture implemented over the ROS [12] framework for the omnidirectional control algorithm is illustrated in figure 7. The hardware of the 5DPO [16] MSL three-wheeled omnidirectional robot is controlled by an Arduino Mega2560 and an Arduino Nano. The 5DPO [16] team had already implemented the serial communication for the robot's firmware. However, the PI controllers for the wheels' angular speed control were changed to the approach formulated in this work (i.e., with anti-windup and Hammerstein nonlinear block to reduce the motors' deadzone), as described in section 3.

Next, a driver (`driver_hardware_5dpo` node) was developed to communicate with the robot's firmware, process the odometry data, and set the angular speed references for the wheels' motors. Even though the use of Qt [18] increases the number of dependencies of the implemented architecture, Qt-based serial communication was used because it provides a robust communication in comparison with other alternatives such as the LibSerial [19] library. The driver processes the odometry data and constructs the ROS message `motors_output` that contains the encoders count, the transmission reduction ratio, and encoders resolution. Also, it subscribes to the `motors_speed_ref` topic to update the angular speed reference of the wheels' motors through the serial communication with the robot's firmware.

The INESC TEC Navigation Stack for mobile robots had already implemented the `odometry_processor` node – processing the encoders count and estimating the odometry pose estimation of the robot and also processing the `cmd_vel` topic to compute the angular speed reference equivalent to the ROS message `motors_speed_ref` – for the differential-drive, four-wheeled omnidirectional, and tricycle steering geometries. However, the node was not compatible with three-wheeled omnidirectional robots. So, the `odometry_processor` was modified to be compatible with three-omnidirectional robots. As for the `omnijoy` node (also available in the INESC TEC Navigation Stack), it allows controlling an omnidirectional robot with a joystick.

In addition to the `move_base` [13] package already explained in section 5, the `rviz` is used mainly for visualization purposes. The trajectory generated by the global planner, the future subset, and the robot's footprint data are available for visualization in the interface. Also, `rviz` allows the definition of a 2D navigation goal (action of setting a new goal for the `move_base` [13] node).

Finally, it was developed a ROS package with only launch files. The package `sdpo_nav_conf` follows the approach used on the INESC Navigation Stack, i.e., a single launch file is used to start the ROS packages and load the parameters required for each one. This approach also allows different configurations. For example, a configuration to run in FEUP with the respective map and another. However, for the scope of this work, only one configuration was implemented without a map, only with odometry data.

7 Experiments

In this section, it is presented all the experiments made for this work. The first experimental results were retrieved from the simulation environment SimTwo [20]. These results are the same ones presented in our previous work. In addition to those results, the proposed omnidirectional controller algorithm with trajectory generation with the A* [11] algorithm was tested on a three-wheeled omnidirectional robot of the 5DPO MSL robot soccer team [16].

7.1 Simulation results

The simulations were performed in the simulator SimTwo [20] that implements the ODE physics engine. SimTwo [20] has available the simulation of omnidirectional robots. The robot's model used in this paper was based on the three-wheeled omnidirectional robot of the 5DPO [16] MSL team. The parameters of the motors and the robot were retrieved from experiments with the real robot, and the ones defined for the PI and PD controllers are the following ones:

- characteristics of the robot:
  - $n = 12$
  - $C_e = 1024$ ppr
  - $l = 0.195$m
  - $D = 0.102$ m
- motors (relative to the wheel):
  - $K_{p,\dot{\varphi}} = 2.6181$ rad.s^{-1}.V^{-1}
  - $\tau_{\dot{\varphi}} = 0.198$ s, $L_{\dot{\varphi}} = 0$ s
  - $K_{c,\dot{\varphi}} = 0.57293$ V.s.rad^{-1}
  - $T_{I,\dot{\varphi}} = 0.19831$ s
- robot $[v, v_n, \omega]$:
  - $K_p = [1, 1, 1]$
  - $\tau = [0.129, 0.128, 0.099]$ s
  - $T_{sett.} = [0.8, 0.8, 0.8]$ s
  - $K_c = [4.41721, 4.38288, 3.40473]$ s^{-1}
  - $T_D = [0.06969, 0.06792, 0.00237]$ s

In terms of analyzing the results from the simulations performed, the maximum (max) and the average (avg) of the following three quality measures are considered:

- ε_d: distance error over time;
- ε_θ: absolute orientation error over time;
- ε_t: trajectory error (distance of the robot's position to the closest point of the desired trajectory).

Next, three different analyses are presented: size's definition of the future trajectory's size (M), comparison of only using PD controllers to the proposed controller in this report, and a comparison between the proposed controller and a generic velocity controller GoToXY (follows the trajectory with a nominal velocity).

7.1.1 Size of the future trajectory (M)

The size M was studied putting the robot through a 2mx2m square with $\theta_{rb}^{W} = 0^{\text{o}}$. The analysis focused on evaluating the maximum and average of the quality measures on the first corner (the interval between 0.5m before and after the corner). Another quality measure defined for this specific path is the overshoot (*o.s.*) for the trajectory outwards. As for deciding the best value for M, it is evaluated the sum of all quality measures ($\sum_{\text{all}}^{\text{m,rad}} \varepsilon_e$, and the ones related to the orientation are converted to radians, due to order of magnitude purposes) and the sum of the maximum of these quality measures ($\sum_{\text{max}}^{\text{m,rad}} \varepsilon_e$). Table 1 presents the results of the analysis.

Analyzing table 1, the first observation is that higher M leads to a lower *o.s.* because the robot can predict the corner earlier. With an average of 1 m/s, $\varepsilon_{max,t}$ increases with M because the prediction of the corner causes a cut

Table 1. Experimental Results of Different Sizes for the Future Trajectory on a Square Corner

K_T	v_{avg} (m/s)	M	$\varepsilon_{avg,d}$ (m)	$\varepsilon_{avg,\theta}$ (º)	$\varepsilon_{avg,t}$ (m)	$\varepsilon_{max,d}$ (m)	$\varepsilon_{max,\theta}$ (º)	$\varepsilon_{max,t}$ (m)	$o.s.$ (m)	$\sum_{\text{all}}^{\text{m,rad}} \varepsilon_e$	$\sum_{\text{max}}^{\text{m,rad}} \varepsilon_e$
10	0.50	3	0.0224	0.822	0.0126	0.0776	2.548	0.0418	0.0418	0.2550	0.2057
		4	0.0130	$\underline{0.503}$	0.0093	0.0405	1.655	0.0271	0.0271	0.1547	0.1236
		$\underline{5}$	$\underline{0.0108}$	0.532	$\underline{0.0084}$	$\underline{0.0339}$	$\underline{1.316}$	$\underline{0.0239}$	0.0150	$\underline{0.1243}$	$\underline{0.0958}$
		6	0.0112	0.627	0.0088	0.0485	1.628	0.0329	0.0111	0.1519	0.1209
		8	0.0173	0.628	0.0128	0.0812	1.472	0.0561	$\underline{0.0092}$	0.2133	0.1722
15	0.75	4	0.0556	2.516	0.0292	0.1467	10.281	0.0682	0.0682	0.5913	0.4625
		6	0.0281	3.062	0.0140	0.0749	7.865	$\underline{0.0314}$	0.0250	0.3641	0.2686
		$\underline{7}$	$\underline{0.0188}$	0.914	$\underline{0.0138}$	$\underline{0.0552}$	2.257	0.0444	0.0150	$\underline{0.2025}$	$\underline{0.1540}$
		8	0.0206	0.725	0.0156	0.0730	$\underline{1.439}$	0.0567	0.0108	0.2145	0.1656
		9	0.0257	$\underline{0.573}$	0.0188	0.0914	1.477	0.0714	$\underline{0.0095}$	0.2526	0.1981
20	1.00	8	0.0549	3.646	0.0241	0.1311	11.785	$\underline{0.0543}$	0.0500	0.5837	0.4411
		9	0.0478	2.038	$\underline{0.0235}$	$\underline{0.0924}$	5.264	0.0568	0.0393	0.3872	0.2804
		$\underline{10}$	$\underline{0.0442}$	$\underline{0.978}$	0.0244	0.1004	$\underline{1.977}$	0.0693	0.0310	$\underline{0.3209}$	$\underline{0.2352}$
		11	0.0445	2.193	0.0254	0.1106	3.712	0.0764	0.0223	0.3823	0.2741
		12	0.0460	1.847	0.0280	0.1253	2.878	0.0878	$\underline{0.0180}$	0.3876	0.2813

inside. In the cases of 0.5 and 0.75 m/s with $M = 3$ and 4 respectively, it noted a shift in the previous claim because $o.s.$ becomes the maximum trajectory error with smaller values of M.

As for choosing the appropriate M depending on the robot's velocity, we can analyze $\sum_{\text{all}}^{\text{m,rad}} \varepsilon_e$ and $\sum_{\text{max}}^{\text{m,rad}} \varepsilon_e$. The analysis of these two quality measures (selecting the value of M that leads to the minimum value of these two measures) lead to a middle ground between predicting the corner and ε_d and ε_t. For 0.5 m/s, $M = 5$ results in the minimum value of ε_d (0.0339 m) and ε_t (0.0239 m) while decreasing 64.1% $o.s.$ relative to $M = 3$. For 0.75 m/s, $M = 7$ leads to the minimum value of ε_d (0.0552 m) and decreases $o.s.$ by 78% relative to $M = 4$. Finally, even though for 1 m/s $M = 10$ does not lead to the lowest values of ε_d and ε_t, the difference is less than 0.01 m to the minimum value (0.01m in a 2m x 2m square is 0.05%) while decreasing $o.s.$ in 38% relative to $M = 8$. So, we propose setting M as 5, 7, and 10 for 0.5, 0.75, and 1 m/s, respectively, for the simulated robot based on the results presented in table 1.

7.1.2 Comparison with PD-only position control

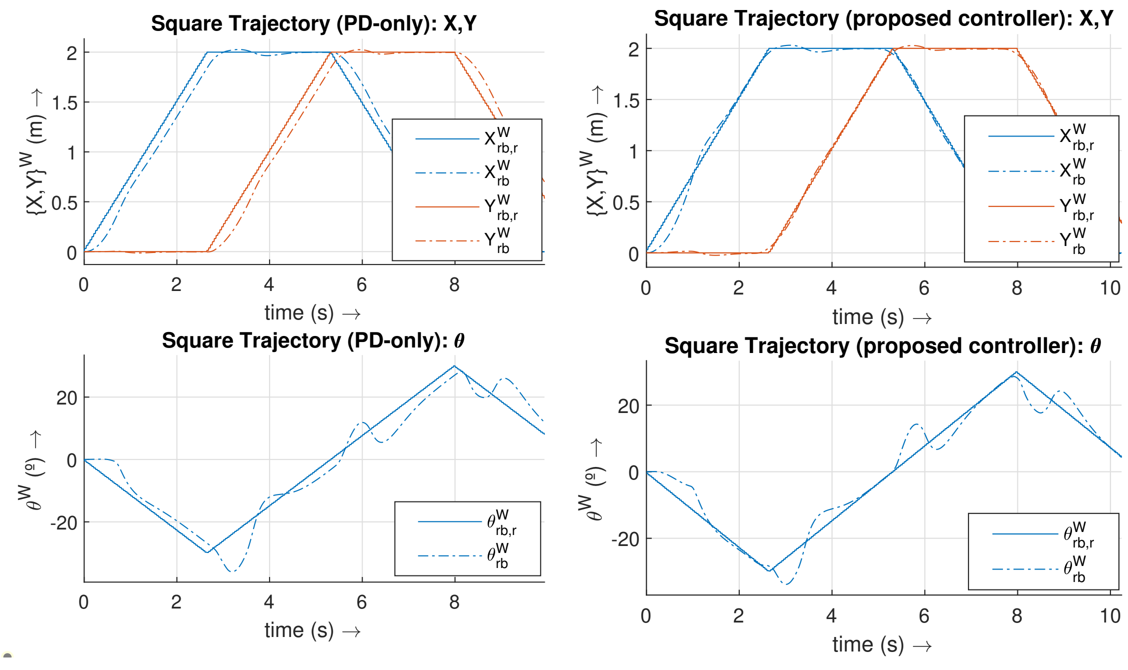
Figure 8 illustrated the comparison of the proposed controller to only using PD controllers for the robot's pose on a square trajectory with a changing orientation over time for the robot. The main disadvantage of using only PD controllers is observable in 8a as the actual pose of the robot is delayed relative to the desired one. Indeed, the delay of approximately 0.23 s for $\{X, Y\}^W$ and 0.3 s for θ^W leads to a $\varepsilon_{max,d}$ and a $\varepsilon_{max,\theta}$ of 0.2247 m and 12.677º, respectively (compared to 0.0583 m and 8.497º for the proposed controller). However, $\varepsilon_{max,t}$ is still similar to the proposed controller: 0.0360 m versus 0.0460 m, respectively.

7.1.3 Comparison with GoToXY

A comparison of the proposed controller with a generic GoToXY is illustrated in figure 9 on a S-type trajectory with an average velocity set by the reference of 0.75 m/s. With the same velocity set for the GoToXY, the controller cannot "catch" the reference. The main reason is that GoToXY does not compensate for the initial delay created by the robot starting from a standstill resulting in a 0.9697 m, 2.444º, and 0.1584 m for $\varepsilon_{max,d}$, $\varepsilon_{max,\theta}$, and $\varepsilon_{max,t}$, respectively. With the nominal velocity at 1.0 m/s, GoToXY achieves a $\varepsilon_{max,d}$, $\varepsilon_{max,\theta}$, and $\varepsilon_{max,t}$ of 0.2028 m, 9.739º, and 0.036 m. In contrast, the proposed controller have a $\varepsilon_{max,d}$, $\varepsilon_{max,\theta}$, and $\varepsilon_{max,t}$ of 0.0359 m, 1.574º, and 0.0233 m, respectively. While the proposed controller led to the robot's pose be similar to the desired one, GoToXY introduces a spacial and time delay with the same velocity.

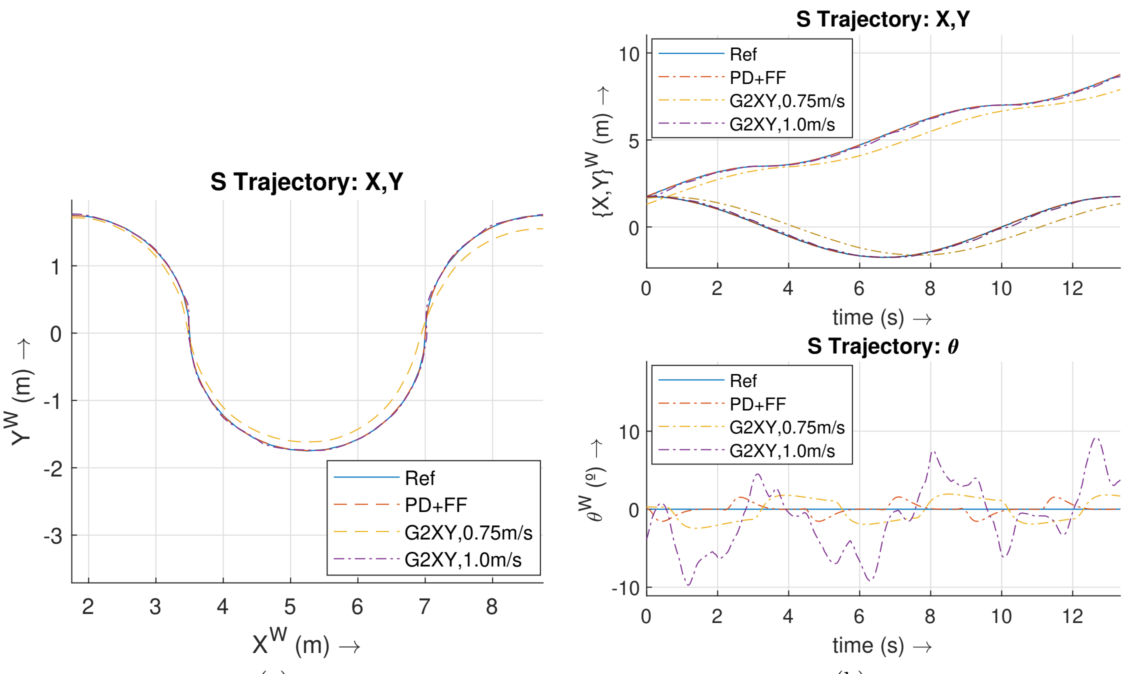
7.2 Experiments with 5DPO omnidirectional robot

As for the experiments with a real robot (illustrated in figure 10), it was performed two experiments with a future trajectory's size of 10 points (approximately 0.5m) and a nominal velocity of 0.3m/s. The main goal was to evaluate



(a) (b)

Fig. 8. Comparison of the proposed controller with a PD-only control



(a) (b)

Fig. 9. Comparison of the proposed controller with GoToXY

the time response of the position controller. The first experiment illustrated in figure 11 defines a goal for the robot with changes in both x and y directions without changing the orientation of the robot. The second one illustrated in figure 12 implicates a change in x, y, and angular directions.

First, the use of feed-forward controllers led to reducing the steady-state error for a ramp input. However, the effect of the first and second derivatives was not noticeable compared to the simulations performed. The main reason is that the controller had not stabilized when the changes in position direction happened. Given that the A* [11] is a greedy

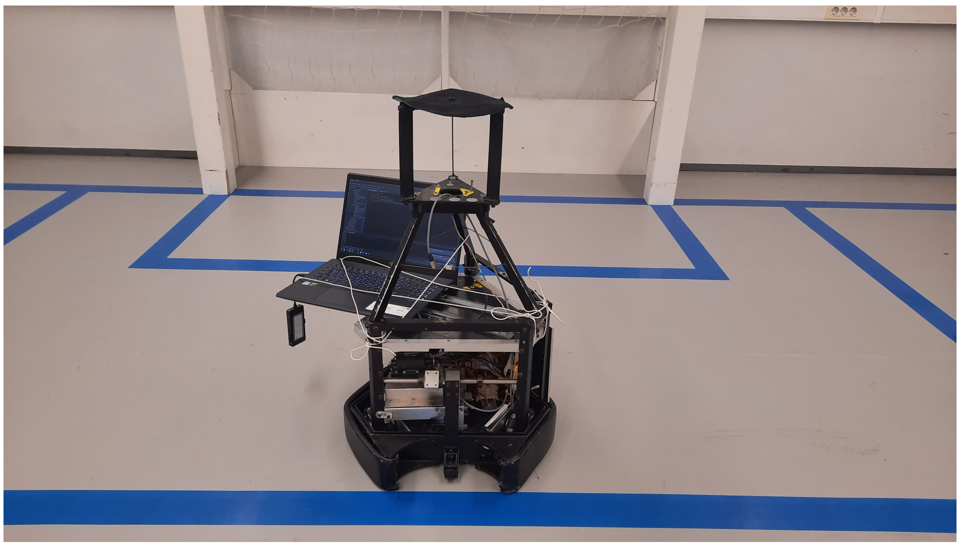


Fig. 10. Three-wheeled omnidirectional robot of the 5DPO [16] team

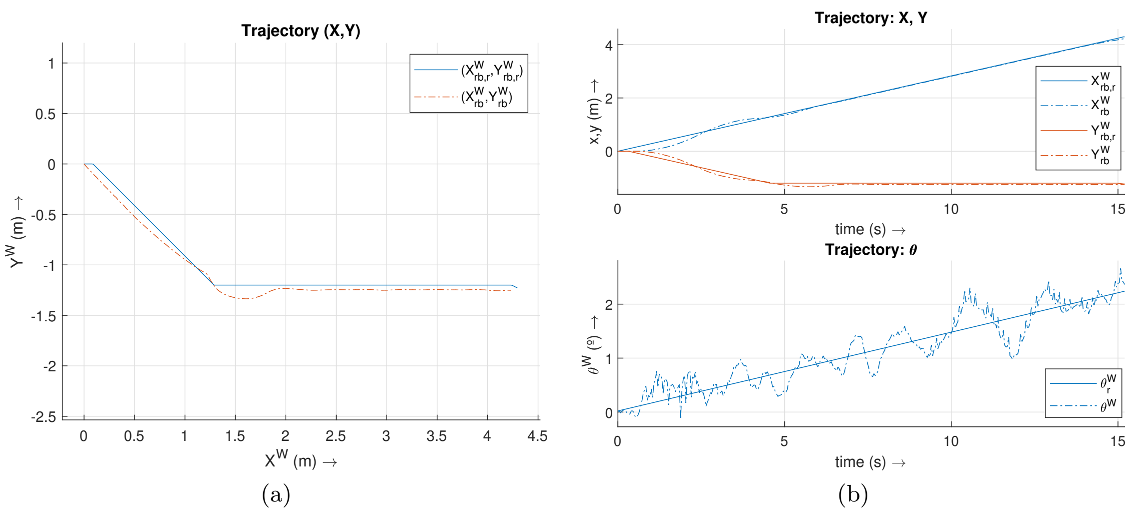


Fig. 11. Example 1 testing with the three-wheeled omnidirectional robot

algorithm, the diagonal generated in the examples happens always before the straight line. Due to the lack of available space, it was not possible to generate trajectories with longer diagonals. Also, an important note is that the position controller for the omnidirectional robot was formulated considering its velocity model when changing between two non-zero nominal velocities. So, the fact that the robot's time response of starting from a standstill pose is different from the one considered on the controllers' formulation could have contributed to worse performance in the standstill situation.

Second, let's analyze the first example (figure 11) on the straight-line part of the trajectory. For the time interval $t \in [7.5, 12.5]$ (the controller is stabilized), the average ($\varepsilon_{avg,d}$) and maximum ($\varepsilon_{max,d}$) distance errors over time are 0.0484m and 0.0532m, respectively. As for the trajectory error, the average ($\varepsilon_{avg,t}$) and maximum ($\varepsilon_{max,t}$) errors are 0.0467m and 0.0514m, respectively. However, these errors are mainly in the v_n direction on the robot's local frame. Note that the orientation of the robot remains approximately 0^{o} leading to approximately aligned robot and global coordinate frames. Even though these error results could be due to inaccuracies of the parameters of the robot's velocity model, the same happens when analyzing the time interval $[6, 12]$ in the second example (figure 12). Indeed, $\varepsilon_{avg,d}$ and $\varepsilon_{max,d}$ are 0.0624m and 0.0898m, respectively, and the $\varepsilon_{avg,t}$ and $\varepsilon_{max,t}$ errors are 0.0436m and 0.0523m. Note that the robot's orientation is changing over time in the 6–12s timeframe of the second example. So, the v, v_n, and ω components (assumed to be independent in the controller's formulation) could have coupled effects on an omnidirectional robot.

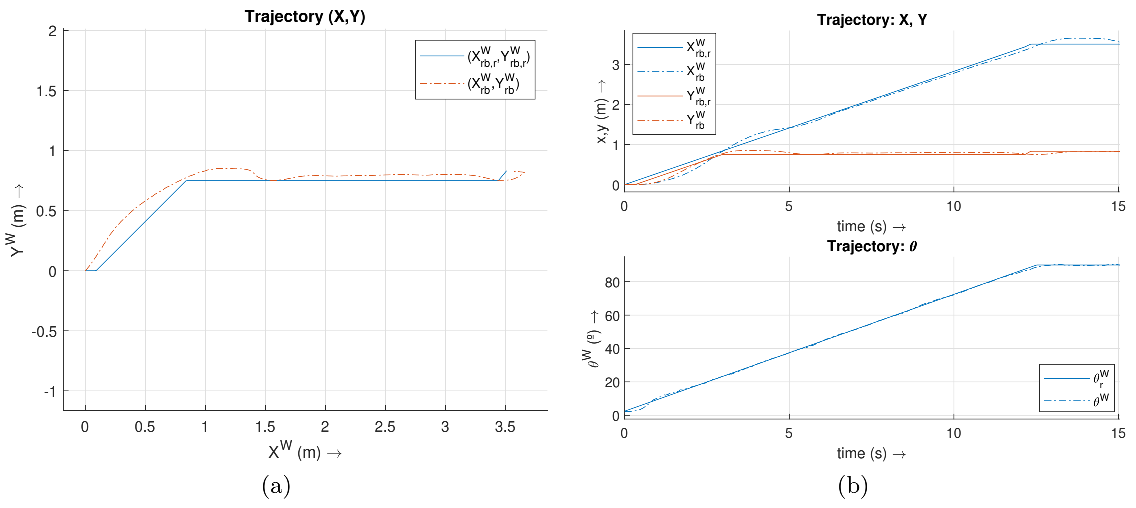


Fig. 12. Example 2 testing with the three-wheeled omnidirectional robot

Lastly, the approximation algorithm implemented (described in section 4) filled the future trajectory's buffer with the desired final goal when no more new poses were available. In the first example, this approach did not lead to an overshoot – 0.0562m that is approximately equal to the goal's distance tolerance of 0.05m defined for the controller – when achieving the desired goal. However, in the second example, it occurred overshoot of 0.1483m. The difference between the two examples is a final orientation difference from the starting one for the second example. So, the approximation algorithm could not eliminate the overshoot effect especially for orientation changing cases. This result indicates that a more precise approximation algorithm must be employed when the robot is near to the desired goal.

8 Conclusions and Future Work

In conclusion, the proposed controller was capable of following a trajectory composed of a set of points defined in both space and time. When compared to a PD-only controller algorithm, the proposed controller led to improvements in terms of eliminating the steady-state error for ramp pose references. This result was valid for both simulation and real environments. However, the experiments with real robots showed that the components of v, v_n, and ω (considered independent for the proposed controller) of the omnidirectional robot could have coupled effects on each other deteriorating the trajectory tracking. Even so, the pose errors noted in the experiments were lower than 0.055m. As for the size of the future trajectory used to compute the feedforward controller's derivatives, the choice of this size requires a tradeoff between predictive characteristics (allow the avoidance of overshoots for sudden changes in the references) and the pose errors along the trajectory: for sudden changes, lowering the value of M increases the overshoot and increasing M increases the pose errors after a certain point. The velocity of the reference also interferes with the most appropriate value of the future trajectory's buffer.

As future work, the proposed controller should consider the possible limitations of the robot in terms of the maximum angular speed of the wheels, and scaling their speed accordingly and the parameterization of the PI and PD controllers could be optimized (difference in the model between starting from a standstill pose versus the robot already in motion). In terms of performing comparisons with other types of approaches, the future work should also consider a comparison with Model Predictive Controllers (MPC) in terms of performance and computational requirements and a comparison with fuzzy-based approaches to evaluate the tracking performance with coupled effects on v, v_n, and ω.

Acknowledgments

I would like to thank professor António Paulo Moreira and researcher Héber Sobreira for all the help given to me throughout this work. Their help and collaboration were crucial for the successful implementation of the proposed control algorithm and integrating it in the `move_base` ROS package. Also, a special thanks to Tiago da Silva for the help in interpreting the existent documentation of `move_base` and configuring the package.

References

1. R. Siegwart, I.R. Nourbakhsh, D. Scaramuzza, *Introduction to autonomous mobile robots*, 2nd edn. (The MIT Press, Cambridge, Massachusetts, 2011)
2. R.H. Abiyev, I.S. Günsel, N. Akkaya, E. Aytac, A. Çağman, S. Abizada, *Fuzzy control of omnidirectional robot*, Procedia Computer Science **120** 608–616 (2017). DOI `10.1016/j.procs.2017.11.286`
3. W. Jianbin, C. Jianping, *An Adaptive Sliding Mode Controller for Four-wheeled Omnidirectional Mobile Robot with Input Constraints*, in *2019 Chinese Control And Decision Conference (CCDC)* (2019), pp. 5591–5596. DOI `10.1109/CCDC.2019.8832894`
4. J. Mu, X. Yan, B. Jiang, S.K. Spurgeon, Z. Mao, *Sliding mode control for a class of nonlinear systems with application to a wheeled mobile robot*, in *2015 54th IEEE Conference on Decision and Control (CDC)* (2015), pp. 4746–4751. DOI `10.1109/CDC.2015.7402959`
5. M.S. Masmoudi, N. Krichen, M. Masmoudi, N. Derbel, *Fuzzy logic controllers design for omnidirectional mobile robot navigation*, Applied Soft Computing **49** 901–919 (2016). DOI `10.1016/j.asoc.2016.08.057`
6. R.H. Abiyev, N. Akkaya, I. Gunsel, *Control of Omnidirectional Robot Using Z-Number-Based Fuzzy System*, IEEE Transactions on Systems, Man, and Cybernetics: Systems **49**(1) 238–252 (2019). DOI `10.1109/TSMC.2018.2834728`
7. Z. Li, Y. Wang, X. Song, Z. Liu, *Neural adaptive tracking control for wheeled mobile robots*, in *2015 International Conference on Fluid Power and Mechatronics (FPM)* (2015), pp. 610–617. DOI `10.1109/FPM.2015.7337188`
8. K. Watanabe, Y. Shiraishi, S.G. Tzafestas, J. Tang, T. Fukuda, *Feedback Control of an Omnidirectional Autonomous Platform for Mobile Service Robots*, Journal of Intelligent and Robotic Systems **22**(3) 315–330 (1998). DOI `10.1023/A:1008048307352`
9. F.G. Rossomando, C.M. Soria, *Identification and control of nonlinear dynamics of a mobile robot in discrete time using an adaptive technique based on neural PID*, Neural Computing and Applications **26**(5) 1179–1191 (2015). DOI `10.1007/s00521-014-1805-8`
10. E.W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1**(1) 269–271 (1959). DOI `10.1007/BF01386390`
11. P.E. Hart, N.J. Nilsson, B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics **4**(2) 100–107 (1968). DOI `10.1109/TSSC.1968.300136`
12. Open Robotics. ROS: powering the world's robots (2021). URL `https://www.ros.org/`. Accessed on June 16, 2021
13. E. Marder-Eppstein. move_base (2021). URL `http://wiki.ros.org/move_base`. Accessed on June 16, 2021
14. D. Lu. global_planner (2021). URL `https://wiki.ros.org/global_planner`. Accessed on June 16, 2021
15. R.B. Sousa, P.G. Costa, A.P. Moreira, *A Pose Control Algorithm for Omnidirectional Robots*, in *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)* (2021), pp. 91–96. DOI `10.1109/ICARSC52212.2021.9429803`
16. T.P. Nascimento, M.A. Pinto, H.M. Sobreira, F. Guedes, A. Castro, P. Malheiros, A. Pinto, H.P. Alves, M. Ferreira, P. Costa, P.G. Costa, A. Souza, L. Almeida, L.P. Reis, A.P. Moreira, *5DPO'2011: Team Description Paper*, in *no. Robocup* (2011)
17. I.L. Chien, *IMC-PID Controller Design - An Extension*, IFAC Proceedings Volumes **21**(7) 147–152 (1988). DOI `10.1016/S1474-6670(17)53816-1`
18. The Qt Company. Qt: cross-platform software development for embedded & desktop (2021). URL `https://www.qt.io/`. Accessed on June 17, 2021
19. LibSerial Development Team. Libserial (2021). URL `https://libserial.readthedocs.io/en/latest/index.html`. Accessed on June 17, 2021
20. P. Costa, J. Gonçalves, J. Lima, P. Malheiros, *SimTwo Realistic Simulator: A Tool for the Development and Validation of Robot Software*, Theory and Applications of Mathematics & Computer Science **1**(1) 17– (2011)