Article

A Novel Control Architecture Based on Behavior Trees for an Omni-Directional Mobile Robot

Rodrigo Bernardo [1,2], João M. C. Sousa [1,*], Miguel Ayala Botto [1] and Paulo J. S. Gonçalves [2]

1 IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal; rodrigo.f.bernardo@tecnico.ulisboa.pt (R.B.); ayalabotto@tecnico.ulisboa.pt (M.A.B.)
2 IDMEC, Instituto Politécnico de Castelo Branco, Av. do Empresário ,Campus da Talagueira, Zona do Lazer, 6000-767 Castelo Branco, Portugal; paulo.goncalves@ipcb.pt
* Correspondence: jmsousa@tecnico.ulisboa.pt

Abstract: Robotic systems are increasingly present in dynamic environments. This paper proposes a hierarchical control structure wherein a behavior tree (BT) is used to improve the flexibility and adaptability of an omni-directional mobile robot for point stabilization. Flexibility and adaptability are crucial at each level of the sense–plan–act loop to implement robust and effective robotic solutions in dynamic environments. The proposed BT combines high-level decision making and continuous execution monitoring while applying non-linear model predictive control (NMPC) for the point stabilization of an omni-directional mobile robot. The proposed control architecture can guide the mobile robot to any configuration within the workspace while satisfying state constraints (e.g., obstacle avoidance) and input constraints (e.g., motor limits). The effectiveness of the controller was validated through a set of realistic simulation scenarios and experiments in a real environment, where an industrial omni-directional mobile robot performed a point stabilization task with obstacle avoidance in a workspace.

Keywords: autonomous robotics; behavior tree; model predictive control; omni-directional systems

check for updates

Citation: Bernardo, R.; Sousa, J.M.C.; Botto, M.A.; Gonçalves, P.J.S. A Novel Control Architecture Based on Behavior Trees for an Omni-Directional Mobile Robot. *Robotics* **2023**, *12*, 170. https://doi.org/10.3390/robotics12060170

Academic Editor: Aman Behal

Received: 31 October 2023
Revised: 2 December 2023
Accepted: 13 December 2023
Published: 16 December 2023

1. Introduction

Mobile robots are the core components of industrial automation systems, especially mobile manipulation applications [1]. The way in which their motion planning is carried out is an important research topic [2]. Mobile robot motion studies can be classified into three distinctive problem sets: path-following control [3,4], trajectory tracking [5,6], and point stabilization [7]. This paper presents a solution for the point stabilization of an omni-directional mobile robot with four mechanical wheels (a holonomic robot). Several control strategies have been implemented and validated in simulation and real environments, but they have mostly been applied to differential robots [8–10]. Omni-directional mobile robots have gained increasing attention as they are more maneuverable than non-omni-directional mobile robots, thus being more suitable for traveling in complex environments; their ability to perform simultaneous and independent translation and rotation movements allows them to obtain relatively sensitive and accurate responses when executing tasks [11]. Today, there are many advanced control alternatives for omni-directional robots: PID control [12], Fuzzy Logic control [13,14], Adaptive Fuzzy Sliding-Mode control [15], Sliding-Mode control [16], the Linear Quadratic Regulator (LQR) [17], and model predictive control (MPC) [18]. Nevertheless, due to their highly non-linear behavior and the number of actuators required to govern each wheel independently, the control of omni-directional mobile robots is a complex task. The above control proposals are effective but have very complex structures that are hard to adjust. Thus, a large amount of theoretical and technical knowledge is required to implement these controllers in practice. Also, the computational cost of certain alternatives makes them less affordable for real-world applications. Thus, many of the works found in the literature only present results validated in a simulation environment.

MPC, also known as receding horizon control, is considered one of the most attractive control strategies for non-linear constrained multiple-input multiple-output (MIMO) systems [19]. In MPC, a future control sequence minimizing an objective function is computed over a finite prediction horizon. The MPC problem is an optimal control problem (OCP) case. The type of MPC controller depends on the OCP formulation, and there are many efficient solution techniques [20]. The direct optimal control method is used in this work, which applies Newton-type optimization schemes to convert the OCP into a non-linear program (NLP) in the form of multiple-shooting discretization [21].

With the growing complexity of robotic systems, using higher-level representations has become increasingly important to improving their software quality and maintainability. State machines have long been among the most common modeling notations for mission specification. Still, in recent years, behavior trees (BTs) have attracted roboticists' attention to express such high-level coordination [22]. A BT is a popular control architecture in the computer game industry and has more recently been applied in the robotics field [23–25]. A BT is a graphical representation of the control logic for autonomous agents. Thus, it provides a powerful and flexible tool for controlling robotic systems, allowing them to handle complex scenarios and adapt to changing situations while maintaining robustness and reliability [24]. By combining controller policies with BTs, relatively simple controllers can be combined to generate complex maneuvering capabilities [26]. BTs can offer a modular and robust framework for hybrid control, enabling goal-based and deliberative task execution [27].

Several control algorithms for mobile robots have been explored and developed over the last few years. However, approaches for implementing them in combination with higher-level representations to generate complex maneuvering capabilities have yet to be examined; ref. [26] was the only relevant work found in the literature in this regard. Other efforts have been made by the Nav2 (https://navigation.ros.org, accessed on 10 July 2023) developers, who used BTs to integrate navigation control logic.

This paper proposes a BT control architecture combined with a non-linear model predictive control (NMPC)-based motion planning scheme to control an omni-directional four-wheeled robotic system. The BT defines the high-level control structure determining the robot's behavior. Further, the NMPC algorithm is encapsulated inside a behavior node to generate low-level optimal control actions for the robotic agent to execute the desired behavior. The BT specifies the goal (e.g., reaching a specific object) and the constraints (e.g., avoiding obstacles). At the same time, the NMPC algorithm generates the optimal control action to achieve the goal while satisfying the constraints. The proposed behavior tree is scalable and extensible, i.e., as the complexity of the robotic system increases or the complexity of the environment in which the robotic system is present increases, additional behaviors or control strategies can be added to the tree without the need to reformulate the complete tree or the NMPC algorithm. We propose a flexible system that quickly switches to an alternative behavior in a fault state. This is particularly useful in real-world robotic applications where unexpected situations can arise. This framework is integrated into the robot operating system (ROS) and applied to a real environment.

The paper is organized as follows: Section 2 presents the robotic agent used in the study and its kinematics. Section 3 presents the controller design formulation. Section 4 presents the validation of the proposed controller in simulated and real environments; a discussion of the approach is also presented. Finally, Section 5 presents the paper's conclusions and future work.

2. Robot Modeling

An omni-directional mobile base with mecanum wheels (Figure 1) allowed the robot to move in any direction. The mechanical wheel mainly comprised a wheel hub and some passive rollers evenly distributed along the wheel's outer edge at a certain angle. Based on the mechanic wheel's characteristics and the system model analysis, the platform could conduct omni-directional movements with strong adaptability, high sensitivity, good stability, and flexible rotation.

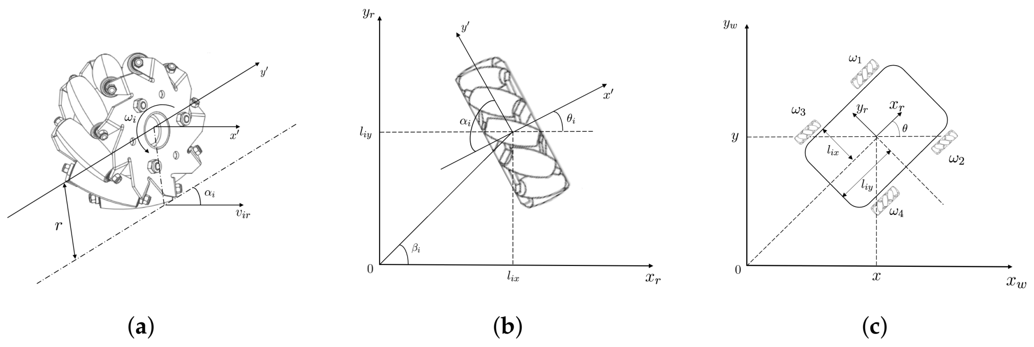


Figure 1. Configuration of the robot and its wheels. (**a**) Wheel *i* motion principle. (**b**) Wheel *i* in the robot coordinates. (**c**) System coordinate frames. Robot frame x_r, y_r, z_r and world frame x_w, y_w, z_w.

Kinematics Modeling

Figure 1 shows the configurations of the robot and its wheels.

According to Figure 1a, the velocity of wheel *i* in frame x', o', y' can be obtained by

$$\begin{bmatrix} v_{ix'} \\ v_{iy'} \end{bmatrix} = \begin{bmatrix} 0 & \sin\alpha_i \\ r_i & \cos\alpha_i \end{bmatrix} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = K_{i1} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix}, \tag{1}$$

where all parameters are considered in terms of the wheel coordinates, wheel speed $(v_{ix'}, v_{iy'})$, and angular velocity (ω_i). v_{ir} represents the norm velocity of the wheel, and r represents the radius of the wheel. It is necessary to translate $v_{ix'}, v_{iy'}$, written in terms of $x'o'y'$ (wheel coordinates) to v_{ix_r}, v_{iy_r}, written in terms of $x_r o_r y_r$ (robot coordinates). From Figure 1b, the equation in terms of the robot coordinates is as follows:

$$\begin{bmatrix} v_{ix_r} \\ v_{iy_r} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} v_{ix'} \\ v_{iy'} \end{bmatrix} = K_{i2}K_{i1} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix}, \tag{2}$$

where K_{i2} is the transition matrix.

According to the geometric properties, the transformation relationship of robot speed between different coordinate systems is obtained by

$$\begin{bmatrix} v_{ix_r} \\ v_{iy_r} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -l_{iy} \\ 0 & 1 & l_{ix} \end{bmatrix} \begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix} = K_{i3} \begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix}, \tag{3}$$

where l_{ix} and l_{iy} are the distances between the wheel coordinates and the robot coordinates (see Figure 1b), and l_i is the distance between the wheels and the base (the center of the robot).

The inverse kinematics model of the omni-directional four-wheel base can be obtained from (1) and (2):

$$K_{i2}K_{i1} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = K_{i3} \begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix}, \quad \mathrm{i} = 1,2,3,4. \tag{4}$$

Equations (1) and (2) relate the variables of the wheel structures and the center of the robot. With the inverse kinematics, the velocity of the system can be obtained by v_{ir} (the linear velocity) and w_i (the angular velocity) of wheel *i* in (5) and the inverse (forward kinematics) in (6).

$$\begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix} = K_i^{+} \begin{bmatrix} \omega_i \\ v_{i_r} \end{bmatrix}, \tag{5}$$

$$\begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = K_i \begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix}, \tag{6}$$

where

$$K_i = K_{i2}^{-1} K_{i1}^{-1} K_{i3}, \quad K_i^{+} = \left(K_i^T \cdot K_i\right)^{-1} K_i^{\top}. \tag{7}$$

Assuming $\phi_i = \theta_i - \alpha_i$ and considering the fact that $l_{ix} = l_i \cos\beta i$, $l_{iy} = l_i \sin\beta i$, and the wheels are the same size,

$$K_i = \frac{1}{-r\sin\alpha_i} \begin{bmatrix} \cos\phi_i & \sin\phi_i & l_i \sin(\phi_i - \beta_i) \\ r\cos\theta_i & -rsin\theta_i & l_i \sin(-\beta_i + \theta_i) r \end{bmatrix}. \tag{8}$$

The forward kinematics are used to determine the robot's velocity, and the inverse kinematics are used to determine the wheels' angular velocities $\boldsymbol{\omega_r} = [\omega_1, \omega_2, \omega_3, \omega_4]^{\top}$. Assuming that the wheels do not slip on the ground, the system inverse kinematics can be obtained by

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{-1}{r} \begin{bmatrix} \frac{\cos\phi_1}{\sin\alpha_1} & \frac{\sin\phi_1}{\sin\alpha_1} & \frac{l_1 \sin(\phi_1 - \beta_1)}{\sin\alpha_1} \\ \frac{\cos\phi_2}{\sin\alpha_2} & \frac{\sin\phi_2}{\sin\alpha_2} & \frac{l_2 \sin(\phi_2 - \beta_2)}{\text{sinff}_2} \\ \frac{\cos\phi_3}{\sin\alpha_3} & \frac{\sin\phi_3}{\sin\alpha_3} & \frac{l_3 \sin(\phi_3 - \beta_3)}{\sin\alpha_3} \\ \frac{\cos\phi_4}{\sin\alpha_4} & \frac{\sin\phi_4}{\sin\alpha_4} & \frac{l_4 \sin(\phi_4 - \beta_4)}{\sin\alpha_4} \end{bmatrix} \begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix}. \tag{9}$$

The Jacobian matrix for the inverse kinematics is given by

$$K_i = \frac{-1}{r} \begin{bmatrix} \frac{\cos\phi_1}{\sin\alpha_1} & \frac{\sin\phi_1}{\sin\alpha_1} & \frac{l_1 \sin(\phi_1 - \beta_1)}{\sin\alpha_1} \\ \frac{\cos\phi_2}{\sin\alpha_2} & \frac{\sin\phi_2}{\sin\alpha_2} & \frac{l_2 \sin(\phi_2 - \beta_2)}{\text{sinff}_2} \\ \frac{\cos\phi_3}{\sin\alpha_3} & \frac{\sin\phi_3}{\sin\alpha_3} & \frac{l_3 \sin(\phi_3 - \beta_3)}{\sin\alpha_3} \\ \frac{\cos\phi_4}{\sin\alpha_4} & \frac{\sin\phi_4}{\sin\alpha_4} & \frac{l_4 \sin(\phi_4 - \beta_4)}{\sin\alpha_4} \end{bmatrix}, \tag{10}$$

and the forward kinematics are given by

$$\begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix} = K_i^{+} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}. \tag{11}$$

The autonomous mobile manipulator robot (AMMR) used in this study comprised an omni-directional mobile base consisting of four mechanical wheels (Figure 1c), with the configuration parameters shown in Table 1.

Table 1. Robot parameters.

i	Wheel	β_i	θ_i	α_i	l_i	l_{ix}	l_{iy}
1	ω_1	$\frac{\pi}{4}$	$\frac{\pi}{2}$	$-\frac{\pi}{4}$	l	l_x	l_y
2	ω_2	$-\frac{\pi}{4}$	$-\frac{\pi}{2}$	$\frac{\pi}{4}$	l	l_x	l_y
3	ω_3	$\frac{3\pi}{4}$	$\frac{\pi}{2}$	$\frac{\pi}{4}$	l	l_x	l_y
4	ω_4	$-\frac{3\pi}{4}$	$-\frac{\pi}{2}$	$-\frac{\pi}{4}$	l	l_x	l_y

Inserting the parameters presented in Table 1 into (9), the inverse kinematics are

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{-1}{r} \begin{bmatrix} 1 & -1 & -(l_x + l_y) \\ 1 & 1 & (l_x + l_y) \\ 1 & 1 & -(l_x + l_y) \\ 1 & -1 & (l_x + l_y) \end{bmatrix} \begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix}. \tag{12}$$

Inserting the parameters presented in Table 1 into (11), the forward kinematics are

$$\begin{bmatrix} v_{x_r} \\ v_{y_r} \\ \omega_z \end{bmatrix} = \frac{r}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} \end{bmatrix}\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}. \tag{13}$$

3. Proposed Control Architecture

A behavior tree (BT) is a way of structuring the switching between different tasks of an autonomous agent. BTs are a very efficient means to create complex systems that are both modular and reactive. These properties are crucial in many applications. Figure 2 shows the control architecture that uses a BT to create a hierarchical control structure and an NMPC for the point stabilization of a generic autonomous mobile manipulator robot (AMMR). In short, it is an architecture that uses modularity, hierarchies, and feedback. Exploring the potential of BTs has already proved effective [26,27].

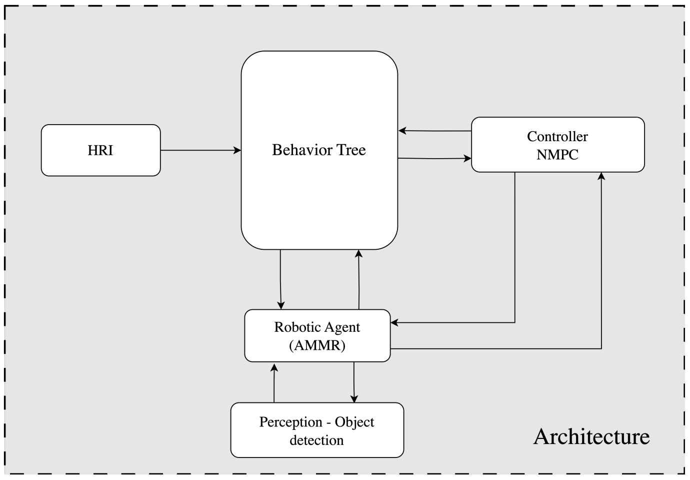


Figure 2. Proposed control architecture.

The architecture shown in Figure 2 comprises five main modules. The human–robot interface (HRI) module allows the human "operator" to send/update or cancel a goal for the robot. The low-level control block comprises the NMPC controller. A module for detecting environmental obstacles (*Perception—Object detection*) and the BT are responsible for the different modules' hierarchical management control. These modules are discussed in more detail in the following sections.

3.1. Detection of Obstacles for Collision Avoidance

As the mobile robot moves, obstacles (either static or moving) may appear in the environment. It is then necessary to check whether these could make the robot's trajectory unfeasible. A common simplification is to model moving and static obstacles as circles [28]. The following sections present the strategies used to update the properties of an obstacle if needed. At each sampling instant, the model verifies if new obstacles have appeared in the planned path. Whenever a new obstacle is detected, the control process is interrupted, and the controller constraints are updated with information regarding the new obstacles.

3.1.1. Autonomous Obstacle Detection

To automatically detect obstacles in the environment, the measurements made by the front and rear lasers equipping the mobile robot are processed before the controller's state constraints are updated. It is necessary to transform the laser's measurements into global coordinates. Figure 3 presents the proposed obstacle detection mechanism. The lasers measurements d_i , $i = 1, 2, \ldots, p$, are collected at each sampling time, while parameters

d_{max} and R are set as constant values. Parameters d_{max} and R represent the maximum allowed distance between the mobile robot and the obstacle detected and the safety region radius, respectively. The obstacle encircles the safety region, not allowing the robot to penetrate it.

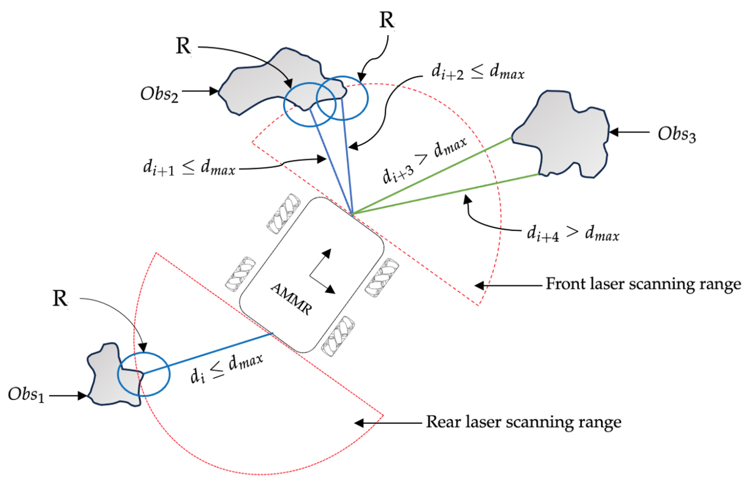


Figure 3. Obstacle detection mechanism.

If one of the measurements d_i satisfies $d_i \le d_{max}$, then the coordinates of the detected obstacle are stored and serve as input to Equation (14) (e.g., obstacles 1 and 2 in Figure 3), where p is the number of detected obstacles. In our approach, the safety regions are excluded from the optimization search. Furthermore, the state θ_w (robot orientation) is assumed to have no limitations. If the scan performed by the lasers detects relevant obstacles, then the set of inequalities of the state constraints are given by

$$\sqrt{\left(x_w - x_{i_{obs}}\right)^2 + \left(y_w - y_{i_{obs}}\right)^2} \ge R, \quad i = 0, \dots, p. \tag{14}$$

where (x_w, y_w) are the coordinates of the location of the mobile robot, and $x_{i_{obs}}, y_{i_{obs}}$ are the coordinates of the points of the objects detected. All coordinates are in the global coordinate system (world frame).

3.1.2. Previously Known Obstacles

Obstacles may be previously known (i.e., previously detected by other robotics agents, environmental sensors, etc.). We have the information of the position in the world frame and the dimensions of these obstacles. For obstacle avoidance, a Euclidean distance (see Equation (15)) is used between the predicted location of the robot (x_w, y_w) and the location of the obstacles $(x_{i_{obs}}, y_{i_{obs}})$ in the world frame. This distance must be greater than the sum of r_r and $r_{i_{obs}}$, representing the robot radius and obstacle radius, respectively.

$$-\sqrt{\left(x_w - x_{i_{obs}}\right)^2 + \left(y_w - y_{i_{obs}}\right)^2} + \left(r_r + r_{i_{obs}}\right) \le 0, \quad i = 0, \dots, . \tag{15}$$

3.2. Controller Formulation

A kinematic model can provide acceptable performance at a low speed and acceleration, and the effect of the robot mass can be neglected to reduce the computational effort [29]. Due to the specific requirements of the robot's practical applicability, it will only work at low speeds, making the system less sensitive to disturbances such as inertia and friction, reducing the need for a complex dynamic model to consider external forces. Low speeds have slower dynamics, which allows for more flexible control requirements and potentially simpler models [30]. We aimed to design a feedback controller using the kinematics model given in (13) to find the optimal control inputs—in this case, the independent angular velocities at each wheel—so that the robot can move to a predetermined point and avoid previously known obstacles.

The state vector describes the omni-directional robot state $\boldsymbol{x} = [x_w, y_w, \theta_w]^\top$, which consists of the (spatial) position (x_w, y_w) and the orientation angle θ_w of the robot in the world frame (Figure 1c). Additionally, the robot control input is defined by the vector $\boldsymbol{u} = [u_1, u_2, u_3, u_4]^\top$, where each input u_i, $i = 1, 2, 3, 4$ represents a corresponding wheel angular velocity. Note that $\boldsymbol{u} = \boldsymbol{\omega_r}$.

This paper addresses a robotic system described by non-linear and time-invariant differential equations with state $\boldsymbol{x} : \mathbb{R} \mapsto \mathcal{X}$ and control $\boldsymbol{u} : \mathbb{R} \mapsto \mathcal{U}$. The discrete-time kinematics model of an omni-directional mobile robot is given by

$$\boldsymbol{x}(k+1) = \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k)), \tag{16}$$

where $\boldsymbol{x}(k) \in \mathbb{R}^n$ and $\boldsymbol{u}(k) \in \mathbb{R}^m$ are the n-dimensional state and m-dimensional control vectors, respectively.

The control objective is to compute an admissible control input driving the system to move toward the equilibrium point defined by $(\boldsymbol{e}(k) = 0)$, where $\boldsymbol{e}$ is the difference between the reference value and the system state vector in the world frame. The state signal vector is denoted as $\boldsymbol{x} = [x_w, y_w, \theta_w]^\top$. The reference state vector $\boldsymbol{x_{ref}} = [x_{ref}, y_{ref}, \theta_{ref}]^\top$ maintains the same value over time, corresponding to the desired goal. The error state vector $\boldsymbol{e}$ is defined as

$$\boldsymbol{e}(k) = \boldsymbol{x_{ref}}(k) - \boldsymbol{x}(k). \tag{17}$$

The global control architecture is shown in Figure 4, where the control loop of an autonomous mobile manipulator robot using an NMPC law is represented. NMPC was developed for the point stabilization of an omni-directional mobile robot with four wheels. NMPC calculates the low-level control actions required at each step as a function of the error (see Equation (17)) using predictions of the system's future behavior to determine the optimal control actions. The aim is to move the robot from one point to another with minimum error and drift while satisfying the constraints on the control input $\boldsymbol{u}$. Each control input is subject to an inequality constraint as follows:

$$u_{\min} \leq u_i \leq u_{\max}; \quad i = 1, 2, 3, 4. \tag{18}$$

where $u_{\min}$ and $u_{\max}$ are the motor rated angular velocities or the required control limits.

We propose using an NMPC scheme to achieve the control objective. To this end, the objective of the control algorithm is to minimize a cost function given by [31]:

$$J(k, \boldsymbol{e}, \boldsymbol{u}) = F(\boldsymbol{e}(N)) + \sum_{j=0}^{N-1} l(\boldsymbol{e}(j), \boldsymbol{u}(j)), \tag{19}$$

Here,

$$\begin{aligned} F(\boldsymbol{e}(N)) &= \boldsymbol{e}(k+N)^\top R \boldsymbol{e}(k+N), \\ l(\boldsymbol{e}(j), \boldsymbol{u}(j)) &= \boldsymbol{e}(k+j)^\top Q \boldsymbol{e}(k+j) + \boldsymbol{u}(k+j)^\top R \boldsymbol{u}(k+j), \end{aligned} \tag{20}$$

where N is the prediction horizon, k is the sampling instant, and $Q \in \mathbb{R}^{n \times n}$ (penalizing deviation state) and $R \in \mathbb{R}^{m \times m}$ (penalizing deviation control) are positive definite symmetric weighting matrices with appropriate dimensions. The control horizon in this approach equals the prediction horizon. The computational efforts are not a problem due to the system characteristics; this way, we have more degrees of freedom to control. A control horizon shorter than the prediction horizon is typically used in constant-reference tracking problems. The optimal control solution is obtained as follows:

$$\boldsymbol{u}^* = \arg\min_{\boldsymbol{u}} J(k, \boldsymbol{e}, \boldsymbol{u}), \tag{21}$$

subject to

$$x(0) = x_0, \tag{22a}$$

$$x(k+j+1) = f(x(k+j), u(k+j)), \tag{22b}$$

$$u_{\min} \leq u \leq u_{\max}, \tag{22c}$$

$$\sqrt{\left(x_{w(k+j)} - x_{i_{obs(k+j)}}\right)^2 + \left(y_{w(k+j)} - y_{i_{obs(k+j)}}\right)^2} \geq R, \tag{22d}$$

$$-\sqrt{\left(x_{w(k+j)} - x_{i_{obs(k+j)}}\right)^2 + \left(y_{w(k+j)} - y_{i_{obs(k+j)}}\right)^2} + \left(r_r + r_{i_{obs(k+j)}}\right) \leq 0, \tag{22e}$$

$$x(k+N) \in x_f, \tag{22f}$$

where $i = 0, \ldots, p$, $j = 0, \ldots, N-1$, x_f is the terminal state constraint, and x_0 is an initial state. The solution of the optimal control problem represents the vector of optimal controls u^*. For state x at time k, the optimal control approach produces the vector of optimal controls $u^* = \left[u_0^*, u_1^*, \ldots, u_{N-1}^*\right]$.

The low-level control applied in this paper is equivalent to that deduced by Hsieh et al. in [31]. Hsieh et al. proved the controller's stability, and the proof of stability is valid in this paper.

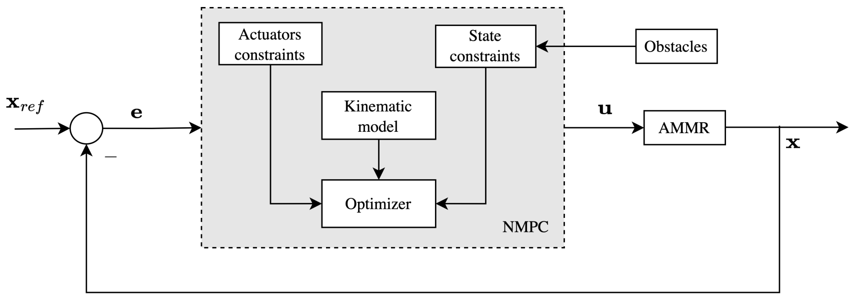


Figure 4. Control of autonomous mobile manipulator robot (AMMR) using NMPC law.

3.3. NMPC Implementation

The CasADi toolbox [32] was used to implement NMPC. CasADi is an open-source tool for non-linear optimization and algorithmic differentiation that sets up, solves, and performs forward and adjoint sensitivity analysis for systems of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) as well as formulating and solving non-linear programming (NLP) problems (https://web.casadi.org/docs, accessed on 12 July 2023). NLP problems are solved using a robust optimization solver, the IPOPT (Interior Point Optimizer) [33]. The interface for the IPOPT was provided by CasADi.

3.4. Implementation in the ROS Environment

The software system used to control the robot was entirely implemented under ROS Noetic (http://wiki.ros.org/noetic, accessed on 12 July 2023), running on Ubuntu 20.04.5 LTS. All ROS nodes were written in C++. The BT library BehaviorTree.CPP (https://github.com/BehaviorTree/BehaviorTree.CPP, accessed on 12 July 2023), an open-source C++ library supporting type-safe asynchronous actions, composable trees, and logging/profiling infrastructures for development, was used to implement a task-managing architecture (see Figure 5). BTs are a powerful and flexible tool for designing the control logic of autonomous agents in robotics, allowing for modular and scalable behavior design that can be easily adapted to different tasks and environments [34]. BTs are typically composed of nodes that determine the flow of execution and the outcome of actions. The primary statuses in a BT are *running*, *failure*, and *success*. There are four types of control flow nodes (fallback, sequence, parallel, and decorator) and two execution nodes (action and condition) [24]. The creators of the Naviagation2 [35] package for ROS2 [36] designed the recovery node (*"RecoveryNode"*), which was used in this work. This node links an action and a recovery action, as the name suggests. The first action is usually the "primary" behavior, and the second action is some action to be performed in the event of the failure of the main behavior. The ticking of the second child action often promotes the chance of the first action succeeding.

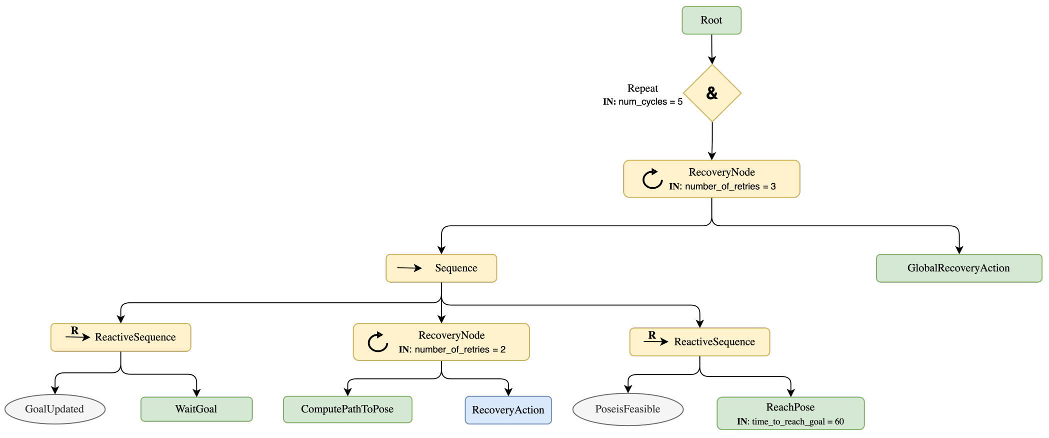


Figure 5. BT for navigating to the goal. Control flow nodes are shown in yellow. Execution nodes of the condition type are shown in grey, defining the conditions that need to be maintained. In green are the nodes of the action type, which comprise the different actions to be executed. In blue is a recovery action, which recovers the failure state of the "primary" action of the control node (RecoveryNode), respecting its established conditions.

The BT presented in Figure 5 was designed to navigate a robotic agent from a starting pose to a goal pose in an environment. The behavior tree (BT) contains custom recovery behaviors in specific sub-contexts and a global recovery node (*"GlobalRecoveryAction"*) for system-level failures. In this BT, all navigation tasks are tried up to three times before returning to the *failure* state. It also allows the user to send up to five goals that must be reached before it is restarted. Initially, the process waits for a goal pose to be published (*"WaitGoal"* node); as soon as the goal is sent, the *"GoalUpdate"* node checks if it is different from the goals sent previously. The *"ComputePathToPose"* and *"ReachPose"* nodes use the NMPC presented in Section 3.2. Based on the available environmental information, the *"ComputePathToPose"* node checks whether designing a safe, collision-free trajectory for the robot from the starting pose to the goal pose is possible. The planned trajectory depends on the environmental information. If the goal is attainable, it publishes the calculated path. Otherwise, it calls the recovery node to recover from the fault conditions or wait for transient problems to pass, such as temporary sensor failures. The *"ReachPose"* node has the function of executing the control actions and sending them to the robot so that it reaches the goal position. Together with this node, the *"PoseisFeasible"* node marks all iterations to ensure the responsiveness of new goal objectives and verify the presence of new obstacles that make the goal unfeasible. If these contextual recoveries fail, the BT enters the global recovery node (*"GlobalRecoveryAction"*). This node is reserved for system-level failures. After each recovery from this node, the main navigation subtree is executed again.

To estimate the robot pose (position and orientation) in the environment based on sensor measurements, *Adaptive Monte Carlo Localization* (AMCL (http://wiki.ros.org/amcl), accessed on 12 July 2023)) is used.

The exact tuning of weighting matrices Q and R to minimize the overall cost function with reliable controller performance is a challenge and an ongoing research problem [37]. The lack of a unified procedure and the setup/perturbation-specific nature of these weights are additional challenges that increase the complexity of the problem. In this paper, we opted to adjust these weights manually using the ROS package *"dynamic_reconfigure"* (http://wiki.ros.org/dynamic_reconfigure, accessed on 12 July 2023), which provides a means to update parameters during the runtime without having to restart the node.

Previously known environmental obstacles are published via a topic created in the ROS environment. All obstacle information can be obtained based on sensors in the robot and sensors present in the environment through a semantic knowledge base [38]. The semantic information provides the controller with richer information regarding the constraints of the environment [39].

4. Simulation and Experimental Results

The proposed BT control architecture was implemented in an *autonomous mobile manipulator robot* (AMMR). These robots combine autonomous navigation with autonomous grasping. An omni-directional mobile base with mecanum wheels allows the robot to move in any direction. In this work, only the mobile base was considered. The AMMR contains sensors, such as lasers and cameras, for autonomous navigation and obstacle recognition. A simulated environment was created to validate the proposed controller. After validating the controller in the simulation environment, a strategy was developed to implement it in the ROS environment. All simulations and experimental results were conducted using a laptop running an Ubuntu 20.04.5 LTS operating system with an Intel® Core^{TM} i7-7740X CPU @ 3.30 GHz × 8 processor, 16 GB RAM, and Quadro P2000/PCIe/SSE2 Graphics. A large number of experiments were performed that aimed to validate the potential of the proposed BT control architecture and define the best parameters for NMPC.

4.1. Simulation Results

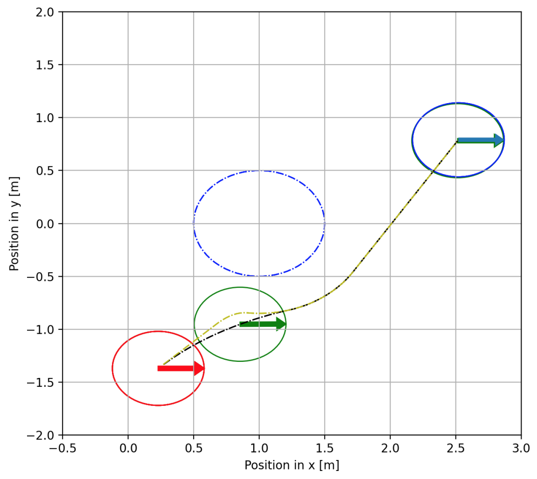
NMPC achieved the point stabilization task in the simulation with obstacle avoidance. The robot started from the initial pose $\boldsymbol{x} = [0.23, -1.37, 0]^\top$ (m, m, rad) and was commanded to stabilize at pose $\boldsymbol{x_{ref}} = [2.52, 0.79, 0]^\top$ (m, m, rad). An obstacle was placed at position $(x_{1_{obs}} = 1,\ y_{1_{obs}} = 0)$ (m,m) with an obstacle radius of $(r_{1_{obs}} = 1)$ (m). NMPC was tested with two prediction horizons, $N = 10$ and $N = 20$. Furthermore, the sampling time of the controller was set to 0.2 s, and several different values were tested for the weighting matrices Q and R to minimize the overall cost function with reliable controller performance. The selected values were the following:

$$Q = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 1000 \end{bmatrix}, \quad R = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}. \tag{23}$$

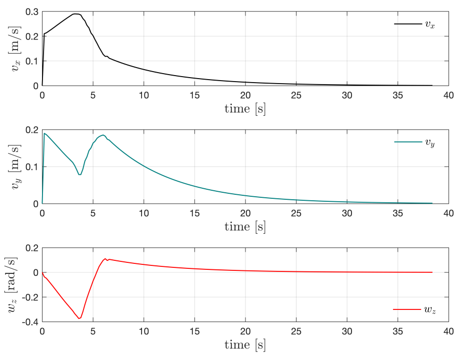
To achieve the accurate localization of the robot and satisfy its actuator saturation limits, the controller saturation limits for the angular velocity $\omega_i = u_i, i = 1,2,3,4$ were set as follows:

$$-4 \leq u_i \leq 4; \quad i = 1,2,3,4. \tag{24}$$

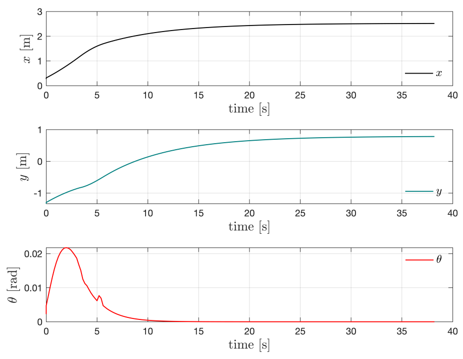
Figure 6 presents the simulation results (with $N = 10$ and $N = 20$) and the controller performance for the case of point stabilization while the robot deviated from an obstacle. In Figure 6a, the robot's position and orientation are represented by a circle with the center located at the robot's position and an arrow indicating the robot's orientation. The robot's starting position is highlighted with a red circle and arrow. The green circle and arrow represent the robot's actual position in the simulation. The blue circle and arrow represent the robot's goal position. The blue dotted circle represents the object. The yellow dashed line represents the trajectory performed by the robot with prediction horizon $N = 10$, and the dark dashed line represents the trajectory performed by the robot with prediction horizon $N = 20$.



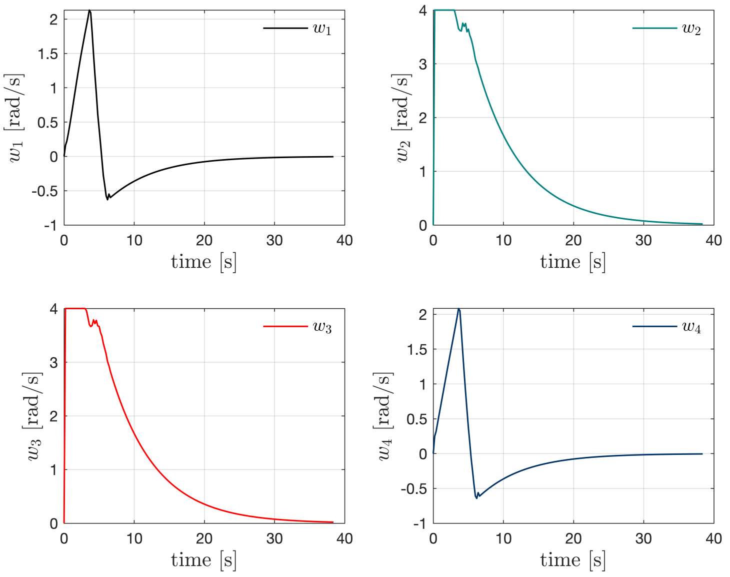
(**a**) Robot position.



(**b**) Velocities.



(**c**) State vectors.



(**d**) Control signals.

Figure 6. Simulation results: (**a**) robot position in the environment; (**b**) linear velocities (v_x and v_y) and angular velocity (w_z); (**c**) state vector elements ($\boldsymbol{x} = [x_w, y_w, \theta]^\top$); (**d**) control signal elements ($\omega_i = u_i$).

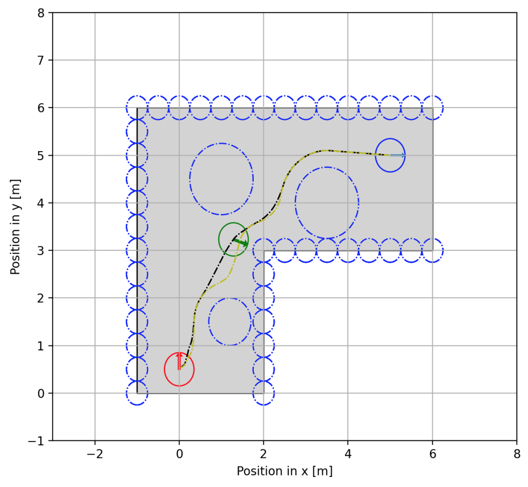
Figure 6a presents the trajectories performed by the AMMR with a prediction horizon of $N = 10$ and $N = 20$. NMPC with a prediction horizon of $N = 20$ achieved a smoother trajectory than with a prediction horizon of $N = 10$. This shows that a longer prediction horizon is necessary for the AMMR to encode more information about the environment, detecting obstacles earlier to ensure smoother navigation to the goal. The computational effort was exponential when increasing from N = 10 to N = 20. The computational time needed to determine the control action by optimizing the NMPC for N = 10 and N = 20 increased by ≈28.15% (from 5.72 ms to 7.33 ms); this was not a limitation in our case because the sampling time was 0.2 s, with $0.2 > 0.0073\,s$, and therefore there was enough time to deal with this increase in the complexity of the optimization problem associated with NMPC.

Figure 6b shows the linear velocities (v_x and v_y) and angular velocity (w_z) sent to the robot. As shown in Figure 6c, the controller stabilized the robot to the desired final position. At the same time, it deviated from the obstacle present in the environment. Figure 6d shows the control actions. The robot angular velocities satisfied the saturation limits given by (24).

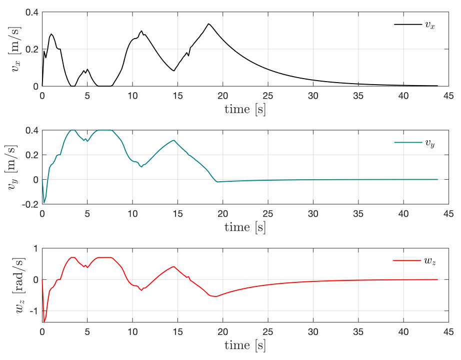
NMPC achieved the point stabilization task in a corridor (see Figure 7a). The robot started from the initial pose $\boldsymbol{x} = [0, 0.5, \frac{\pi}{2}]^\top$ (m, m, rad) and was commanded to stabilize at pose $\boldsymbol{x_{ref}} = [5, 5, 0]^\top$ (m, m, rad). Three obstacles were placed in the corridor to increase the complexity of the environment: obstacle_1 ($x_{1_{obs}} = 1$, $y_{1_{obs}} = 4.5$, $r_{1_{obs}} = 1.5$) (m, m, m); obstacle_2 ($x_{2_{obs}} = 1.2$, $y_{2_{obs}} = 1.5$, $r_{2_{obs}} = 1$) (m, m, m); and obstacle_3 ($x_{3_{obs}} = 3.5$, $y_{3_{obs}} = 4$, $r_{3_{obs}} = 1.5$) (m, m, m). The blue dotted circles represent the objects. The walls were detected by the laser sensors attached to the robot and modeled as explained in Section 3.1, and the set of state constraints was given by (14). This experiment defined 46 circles ("obstacles"), which modeled the corridor's walls.

The sampling time of the controller was set to 0.2 s. It was tested with two prediction horizons, $N = 20$ and $N = 35$, and the Q and R matrices were the same as those used in the simulation (see (23)). To achieve the accurate localization of the robot and satisfy its actuator saturation limits, the controller saturation limits for the angular velocity were set as in (24).

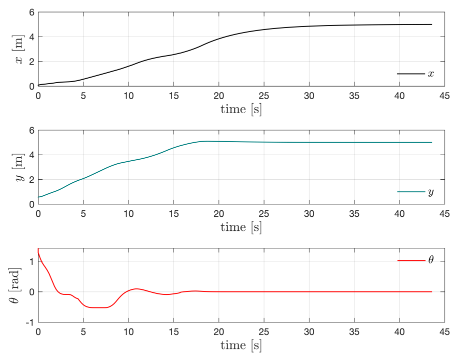
In Figure 7a, the robot's starting position is highlighted with a red circle and arrow. A green circle and arrow represent the robot's actual position in the simulation. A blue circle and arrow represent the robot's goal position. The yellow dashed line represents the trajectory performed by the robot with prediction horizon $N = 20$, and the dark dashed line represents the trajectory performed by the robot with prediction horizon $N = 35$. As the complexity of the environment increased, the $N = 20$ prediction horizon became insufficient to obtain a smooth trajectory, as can be seen in Figure 7a, with the $N = 35$ prediction horizon showing satisfactory results; the sampling period was not violated. Figure 7b shows the linear velocities (v_x and v_y) and angular velocity (w_z) sent to the robot. As shown in Figure 7c, the controller stabilized the robot to the desired final position. At the same time, the mobile robot moved along the corridor, avoiding collision with the obstacles and the walls of the corridor. Figure 7d shows the control actions, and the robot's angular velocities satisfied the saturation limits given by (24).



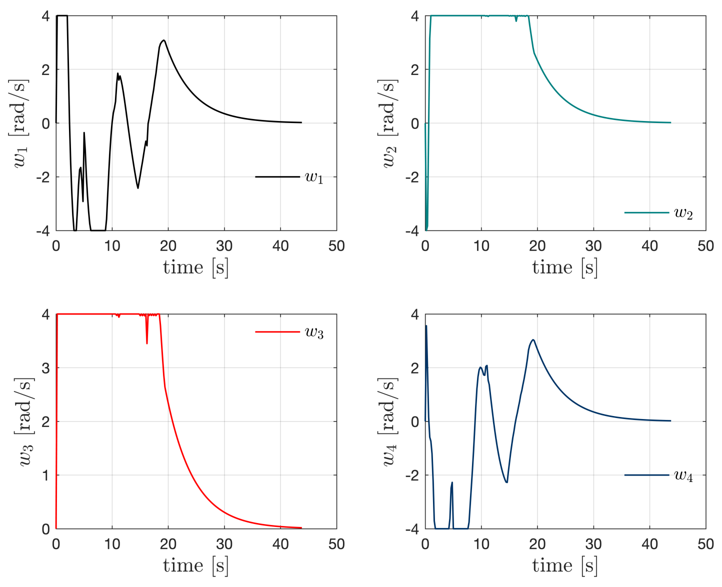
(**a**) Robot position.



(**b**) Velocities.



(**c**) State vectors.



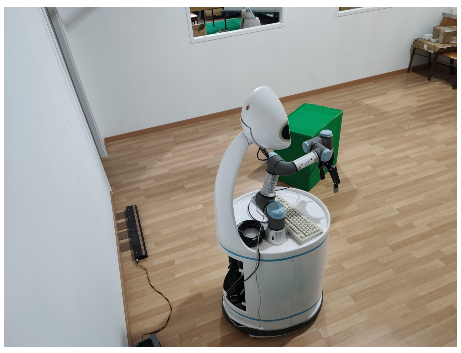
(**d**) Control signals.

Figure 7. Simulation in the corridor: (**a**) robot position in the environment; (**b**) linear velocities (v_x and v_y) and angular velocity (w_z); (**c**) state vector elements ($\boldsymbol{x} = [x_w, y_w, \theta]^\top$); (**d**) control signal elements ($\omega_i = u_i$).

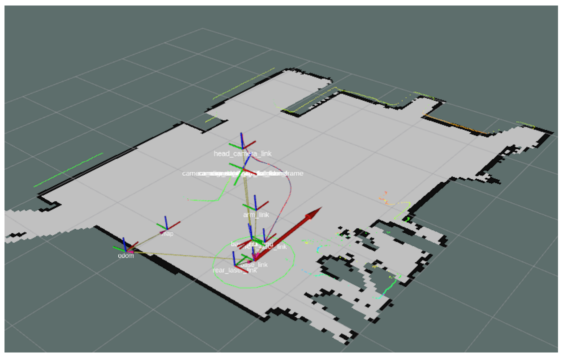
4.2. Experimental Validation

NMPC achieved the point stabilization task with obstacle avoidance in the experimental validation. The robot started from the initial pose $\boldsymbol{x} = [0.23, -1.37, 0]^\top$ (m, m, rad) and was commanded to stabilize at pose $\boldsymbol{x_{ref}} = [2.52, 0.79, 0]^\top$ (m, m, rad). The obstacle was placed at position ($x_{1_{obs}} = 1,\ y_{1_{obs}} = 0$) (m, m) with an obstacle radius of ($r_{1_{obs}} = 1$) (m). The sampling time of the controller was set to 0.2 s with a prediction horizon of $N = 20$, and the Q and R matrices were the same as those used in the simulation (see (23)). To achieve the accurate localization of the robot and satisfy its actuator saturation limits, the controller saturation limits for the angular velocity $\omega_i = u_i, i = 1, 2, 3, 4$ were set as in (24).

Figure 8 presents an overview of the experimental setup. Figure 8a presents the scenario created for controller validation, where the robot and the object (green cube) placed in the environment are visible. Figure 8b shows the environment where the robot was implemented, as well as its transform frame (TF (http://wiki.ros.org/tf, accessed on 12 July 2023)). The TF is a library used in ROS, an open-source framework that helps manage coordinate transformations between different frames of reference within a robot or a robotic system. This environment was obtained through the environment visualizer ROS RViz (http://wiki.ros.org/rviz, accessed on 12 July 2023). The two faces of the object present in the environment were delimited in green by the laser sensors attached to the robot, as can be seen in Figure 8b. The red arrow corresponds to the robot's current position and orientation. The robot was in the goal position ($\boldsymbol{x_{ref}} = [2.52, 0.79, 0]^\top$ (m, m, rad)). A circle representing the robot area can be seen in green. The red line represents the ideal trajectory generated initially, and the blue line represents the trajectory performed by the robot to reach the goal.



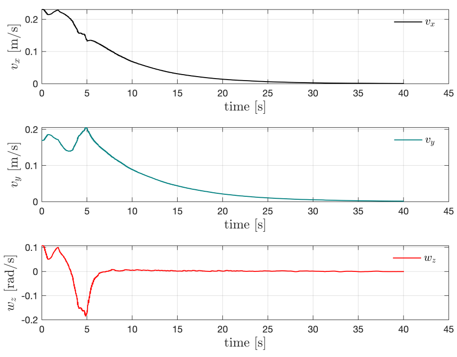
(**a**) Real scenario.



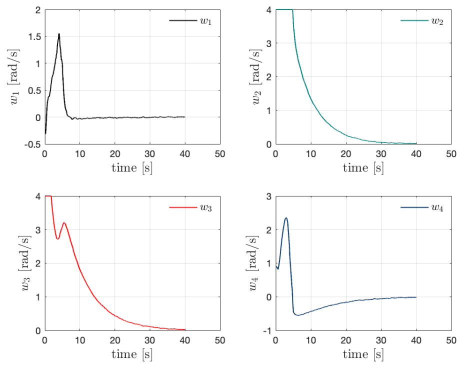
(**b**) Scenario in RViz.

Figure 8. Overview of the experimental set-up. (**a**) A panoramic picture of the scenario created for controller validation. (**b**) Representation of the environment in RViz based on the environment map and observations based on the robot sensors, including the robot footprint and its transform frame (TF).

Figure 9a presents the linear velocities v_x and v_y, as well as the angular velocity w_z, sent to the robot. Figure 9b presents the control actions and shows an acceptable control performance. The results confirmed that the constraints imposed on the maximal and minimal wheel speeds were not violated, i.e., (24) was satisfied for all time steps. The same behavior was observed for the remaining initial conditions.



(**a**) Velocities.



(**b**) Control signals.

Figure 9. Experimental results: (**a**) linear velocities (v_x and v_y) and angular velocity (w_z) at which the robot moved, (**b**) control signal elements ($\omega_i = u_i$).

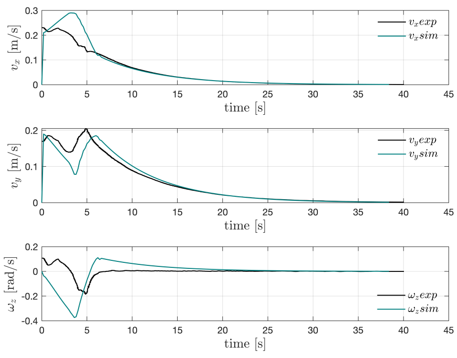
We quantified the controller's performance by calculating the root mean square (RMS) between the trajectory performed by the robot and the previously calculated trajectory ("optimal trajectory"). We used the RMS combined with the Euclidean distance to achieve this; see (25).

$$\text{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((x_{2i}-x_{1i})^2+(y_{2i}-y_{1i})^2\right)} \tag{25}$$

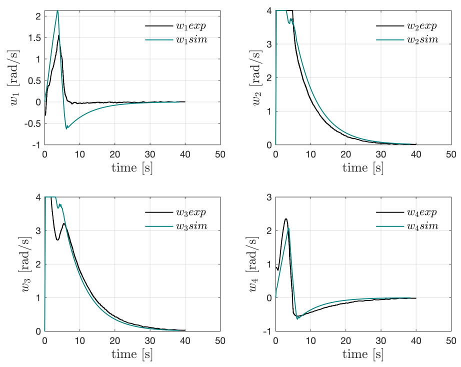
where n is the number of points in each trajectory, and (x_{1i}, y_{1i}), and (x_{2i}, y_{2i}) are the coordinates of corresponding points in the two trajectories.

We obtained an RMS of 0.11 m, a satisfactory value; since we were not following the previously calculated trajectory, it was only used to check that the goal sent was feasible. The experimental results showed the good performance of the proposed control strategy.

Figure 10 combines the simulated results with the experimental results of the two tests carried out under the same conditions to prove the developed model's quality. Figure 10a combines the speeds shown in Figures 6b and 9a; Figure 10b combines the control signal shown in Figure 6d with that in Figure 9b. Analyzing the results, we can see that the model developed in the simulation was very close to the experimental validation when a real robot was used.



(**a**) Velocities.



(**b**) Control signals.

Figure 10. Comparison of experimental and simulated results.

The controller proved to be effective in meeting the goal, even considering the sensors' errors.

5. Conclusions

Robotic systems are increasingly present in dynamic environments. This paper proposed a hierarchical control structure wherein a behavior tree (BT) is used to improve the flexibility and adaptability of an omni-directional mobile robot for point stabilization. Flexibility and adaptability are crucial at each level of the sense–plan–act loop to implement robust and effective robotic solutions in dynamic environments. The controller was built based on the kinematics model of the robot. The BT combines high-level decision making and continuous execution monitoring while applying non-linear model predictive control (NMPC) for the point stabilization of an omni-directional mobile robot. The proposed control architecture can guide the mobile robot to any configuration within the workspace while satisfying state constraints (e.g., obstacle avoidance) and input constraints (e.g., motor limits). This proved to be a powerful and flexible tool. The effectiveness of the controller was validated through a set of realistic simulation scenarios and experiments in a real environment, where an industrial omni-directional mobile robot performed a point stabilization task with obstacle avoidance in a workspace. The behavioral hierarchy defined by the BT was shown to be suitable for dealing with events and situations where the robot needed to make decisions quickly based on the conditions of the environment, proving to be a robust strategy with high flexibility, managing to change the agent's behavior based on unforeseen events such as the appearance of a new obstacle in the path.

Future work will extend the performed analysis to other challenging problems, such as trajectory tracking (i.e., following the trajectory generated by a global planner) [5,6] and path-following control [3,4].

Author Contributions: R.B.: Conceptualization, Methodology, Software, Investigation, Validation, Data curation, Writing—original draft, Visualization. M.A.B.: Conceptualization, Methodology, Writing—review & editing, Supervision. J.M.C.S.: Conceptualization, Writing—review & editing, Supervision, Funding acquisition. P.J.S.G.: Conceptualization, Methodology, Resources, Writing—review & editing, Supervision, Funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This work was financed by national funds from the FCT (Foundation for Science and Technology), I.P., through IDMEC under LAETA, project UIDB\50022\2020. The work of Rodrigo Bernardo was supported by the PhD Scholarship BD\6841\2020 from the FCT. This work indirectly received funding from the European Union's Horizon 2020 programme under StandICT.eu 2026 (Grant Agreement No. 101091933).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare that they do not have any competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Bernardo, R.; Sousa, J.M.; Gonçalves, P.J. Survey on robotic systems for internal logistics. *J. Manuf. Syst.* **2022**, *65*, 339–350. [CrossRef]
2. Elbanhawi, M.; Simic, M. Sampling-based robot motion planning: A review. *IEEE Access* **2014**, *2*, 56–77. [CrossRef]
3. Zhang, J.; Shao, X.; Zhang, W.; Na, J. Path-Following Control Capable of Reinforcing Transient Performances for Networked Mobile Robots Over a Single Curve. *IEEE Trans. Instrum. Meas.* **2023**, *72*, 1–12. [CrossRef]
4. Hung, N.; Rego, F.; Quintas, J.; Cruz, J.; Jacinto, M.; Souto, D.; Potes, A.; Sebastiao, L.; Pascoal, A. A review of path following control strategies for autonomous robotic vehicles: Theory, simulations, and experiments. *J. Field Robot.* **2023**, *40*, 747–779. [CrossRef]
5. Gu, D.; Hu, H. Receding horizon tracking control of wheeled mobile robots. *IEEE Trans. Control Syst. Technol.* **2006**, *14*, 743–749.
6. Ye, H.; Wang, S. Trajectory tracking control for nonholonomic wheeled mobile robots with external disturbances and parameter uncertainties. *Int. J. Control. Autom. Syst.* **2020**, *18*, 3015–3022. [CrossRef]
7. Azizi, M.R.; Keighobadi, J. Point stabilization of nonholonomic spherical mobile robot using nonlinear model predictive control. *Robot. Auton. Syst.* **2017**, *98*, 347–359. [CrossRef]
8. Rösmann, C.; Makarow, A.; Bertram, T. Online motion planning based on nonlinear model predictive control with non-euclidean rotation groups. In Proceedings of the 2021 European Control Conference (ECC), Delft, The Netherlands, 29 June–2 July 2021; pp. 1583–1590.

9. Zavala, V.M.; Biegler, L.T. The advanced-step NMPC controller: Optimality, stability and robustness. *Automatica* **2009**, *45*, 86–93. [CrossRef]
10. Mehrez, M.W.; Mann, G.K.; Gosine, R.G. Stabilizing nmpc of wheeled mobile robots using open-source real-time software. In Proceedings of the 2013 16th International Conference on Advanced Robotics (ICAR), Montevideo, Uruguay, 25–29 November 2013; pp. 1–6.
11. Siegwart, R.; Nourbakhsh, I.R.; Scaramuzza, D. *Introduction to Autonomous Mobile Robots*; MIT Press: Cambridge, MA, USA, 2011.
12. Rojas, R.; Förster, A.G. Holonomic control of a robot with an omnidirectional drive. *KI-Künstliche Intell.* **2006**, *20*, 12–17.
13. Masmoudi, M.S.; Krichen, N.; Masmoudi, M.; Derbel, N. Fuzzy logic controllers design for omnidirectional mobile robot navigation. *Appl. Soft Comput.* **2016**, *49*, 901–919. [CrossRef]
14. Zijie, N.; Qiang, L.; Yonjie, C.; Zhijun, S. Fuzzy control strategy for course correction of omnidirectional mobile robot. *Int. J. Control Autom. Syst.* **2019**, *17*, 2354–2364. [CrossRef]
15. Huang, J.T.; Chiu, C.K. Adaptive fuzzy sliding mode control of omnidirectional mobile robots with prescribed performance. *Processes* **2021**, *9*, 2211. [CrossRef]
16. Xie, Y.; Zhang, X.; Meng, W.; Xie, S.; Jiang, L.; Meng, J.; Wang, S. Coupled sliding mode control of an omnidirectional mobile robot with variable modes. In Proceedings of the 2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Boston, MA, USA, 6–9 July 2020; pp. 1792–1797.
17. Morales, S.; Magallanes, J.; Delgado, C.; Canahuire, R. LQR trajectory tracking control of an omnidirectional wheeled mobile robot. In Proceedings of the 2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA), Barranquilla, Colombia, 1–3 November 2018; pp. 1–5.
18. Kanjanawanishkul, K. MPC-Based path following control of an omnidirectional mobile robot with consideration of robot constraints. *Adv. Electr. Electron. Eng.* **2015**, *13*, 54–63. [CrossRef]
19. Schwenzer, M.; Ay, M.; Bergs, T.; Abel, D. Review on model predictive control: An engineering perspective. *Int. J. Adv. Manuf. Technol.* **2021**, *117*, 1327–1349. [CrossRef]
20. Rao, A.V. A survey of numerical methods for optimal control. *Adv. Astronaut. Sci.* **2009**, *135*, 497–528.
21. Bock, H.G.; Plitt, K.J. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proc. Vol.* **1984**, *17*, 1603–1608. [CrossRef]
22. Ghzouli, R.; Berger, T.; Johnsen, E.B.; Wasowski, A.; Dragule, S. Behavior Trees and State Machines in Robotics Applications. *IEEE Trans. Softw. Eng.* **2023**, *49*, 4243–4267. [CrossRef]
23. Banerjee, B. Autonomous acquisition of behavior trees for robot control. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 3460–3467.
24. Colledanchise, M.; Ögren, P. *Behavior Trees in Robotics and AI: An Introduction*; CRC Press: Boca Raton, FL, USA, 2018.
25. Colledanchise, M.; Almeida, D.; Ögren, P. Towards blended reactive planning and acting using behavior trees. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 8839–8845.
26. Bhat, S.; Stenius, I. Controlling an underactuated auv as an inverted pendulum using nonlinear model predictive control and behavior trees. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 12261–12267.
27. Marzinotto, A.; Colledanchise, M.; Smith, C.; Ögren, P. Towards a unified behavior trees framework for robot control. In Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 31 May–7 June 2014; pp. 5420–5427.
28. Breivik, M. MPC-based mid-level collision avoidance for ASVs using nonlinear programming. In Proceedings of the 2017 IEEE Conference on Control Technology and Applications (CCTA), Maui, HI, USA, 27–30 August 2017; pp. 766–772.
29. Raffo, G.V.; Gomes, G.K.; Normey-Rico, J.E.; Kelber, C.R.; Becker, L.B. A predictive controller for autonomous vehicle path tracking. *IEEE Trans. Intell. Transp. Syst.* **2009**, *10*, 92–102. [CrossRef]
30. Marques, F.; Flores, P.; Pimenta Claro, J.; Lankarani, H.M. A survey and comparison of several friction force models for dynamic analysis of multibody mechanical systems. *Nonlinear Dyn.* **2016**, *86*, 1407–1443. [CrossRef]
31. Hsieh, C.H.; Liu, J.S. Nonlinear model predictive control for wheeled mobile robot in dynamic environment. In Proceedings of the 2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Kaohsiung, Taiwan, 11–14 July 2012; pp. 363–368.
32. Andersson, J.A.; Gillis, J.; Horn, G.; Rawlings, J.B.; Diehl, M. CasADi: A software framework for nonlinear optimization and optimal control. *Math. Program. Comput.* **2019**, *11*, 1–36. [CrossRef]
33. Wächter, A.; Biegler, L.T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* **2006**, *106*, 25–57. [CrossRef]
34. Iovino, M.; Scukins, E.; Styrud, J.; Ögren, P.; Smith, C. A survey of Behavior Trees in robotics and AI. *Robot. Auton. Syst.* **2022**, *154*, 104096. [CrossRef]
35. Macenski, S.; Martín, F.; White, R.; Clavero, J.G. The marathon 2: A navigation system. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 2718–2725.
36. Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, architecture, and uses in the wild. *Sci. Robot.* **2022**, *7*, eabm6074. [CrossRef] [PubMed]

37. Forgione, M.; Piga, D.; Bemporad, A. Efficient calibration of embedded MPC. *IFAC-PapersOnLine* **2020**, *53*, 5189–5194. [CrossRef]
38. Kostavelis, I.; Gasteratos, A. Semantic mapping for mobile robotics tasks: A survey. *Robot. Auton. Syst.* **2015**, *66*, 86–103. [CrossRef]
39. Bernardo, R.; Sousa, J.; Gonçalves, P.J. Planning robotic agent actions using semantic knowledge for a home environment. *Intell. Robot.* **2021**, *1*, 116–130. [CrossRef]