Sensor-based Navigation of Omnidirectional Wheeled Robots Dealing with both Collisions and Occlusions

Abdellah Khelloufi, Nouara Achour, Robin Passama, Andrea Cherubini

Robotica () © Cambridge University Press
doi:

Sensor-based Navigation of Omnidirectional Wheeled Robots Dealing with both Collisions and Occlusions

Abdellah Khelloufi† ‡ §*, Nouara Achour‡, Robin Passama§ and Andrea Cherubini§

†*Center for Development of Advanced Technologies CDTA, 20 Aout 1956 City, Baba Hassen Algiers, Algeria*
‡*Faculty of Electronics and Computer Science, USTHB, BP32 EL-ALIA, 16111 Bab Ezzouar Algiers, Algeria*
§*LIRMM, Université de Montpellier, CNRS, Montpellier, France*

)

SUMMARY
Navigation tasks are often subject to several constraints that can be related to the sensors (visibility) or come from the environment (obstacles). In this paper, we propose a framework for autonomous omnidirectional wheeled robots, that takes into account both collision and occlusion risk, during sensor-based navigation. The task consists in driving the robot towards a visual target in the presence of static and moving obstacles. The target is acquired by fixed —*limited field of view*— on-board cameras, while the surrounding obstacles are detected by lidar scanners. To perform the task, the robot has not only to keep the target in view while avoiding the obstacles, but also to predict its location in the case of occlusion. The effectiveness of our approach is validated through several experiments.

KEYWORDS: Sensor-based navigation; omnidirectional wheeled robots; obstacle avoidance; visibility constraints; occlusions.

1. Introduction

Navigation strategies generally aim at endowing a robot with capacities of perception, decision, and action, that allow it to autonomously move in the environment. These strategies are traditionally divided in two main classes, depending on whether the problem is solved *locally* or *globally*. The *global* approach [1]-[3], usually consists in motion planning, hence relies on accurate knowledge of the robot pose and on a global map of the environment. On the other hand, *local* or reactive strategies are based on instantaneous information from on-board sensors. These strategies include: potential fields [4, 5], vector field histogram [6], dynamic window [7], obstacle-restriction method [8], and closest gap [9].

In contrast with static environments, where classical path planning approaches are suitable, dynamic environments are more challenging to deal with. Indeed, the decision about the robot motion must be related with the on-line perception of the workspace and the future behaviour of the moving obstacles. To address this issue, several motion planing algorithms have been developed [10]. Some of these works are based on the concept of velocity obstacle [11]-[13]. Other approaches employ probabilistic representation to take into account the environment uncertainty [14, 15]. These approaches are however computationally expensive. The combination of global and local planers is generally used to improve the computational cost [16, 17]. Learning-based methods are also applied by learning cost functions from human demonstration (teleoperation) [18, 19]. On the other hand, reactive strategies have proved more appropriate to use in unknown environments. In particular,

* Corresponding author. E-mail: akhelloufi@cdta.dz

the potential field method was extended to take into account unpredictably moving obstacles [20]-[23]. Biologically inspired algorithms are another class of paradigms that can be applied to avoid collisions [24, 25].

One of the advantages of reactive strategies is that they can be well combined with other sensor-based approaches such as visual servoing [26, 27]. In visual servoing, the task is defined in the sensor space, instead of configuration space, and it does not require neither a global model of the environment nor robot localization. Works in this area include [28], where image-based navigation is combined with obstacle avoidance, for a differential-drive robot equipped with a pan actuated camera. Moving obstacles may be considered in the same approach, as shown in [29]. Recently, another interesting framework for visual navigation with obstacle avoidance has been presented in [30], for a car-like robot equipped with an actuated camera and a lidar. The framework is based on *tentacles*, i.e., drivable paths used to predict possible collisions in the near future. The task realized in [30] consists in following a visual path represented by key images, without colliding with the ground obstacles. The authors show that obstacle avoidance does not affect visual navigation and that their approach outperforms potential fields [31]. The framework was later improved [32, 33], by using a Kalman filter to deal with moving obstacles.

A frequent problem in vision-based task control is the loss of image features, due to occlusions or to the sensor's limited field of view. Various prediction and correction methods have been proposed to solve this issue [34]-[39]. However, all these approaches are dedicated to manipulator arms. In the context of mobile robotics, the problem of keeping visual target or landmark visibility has been addressed from different viewpoints. The authors of [40] and [41] study the shortest paths in terms of distance in the image plane for a differential drive robot equipped with a limited field of view sensor. The objective is to obtain the globally optimal paths that allow the robot to maintain a landmark in sight in the absence of obstacles. Recently, these works have been extended to handle environments populated with static obstacles [42]-[44]. However, all the methods proposed in these works are based on motion planning and require a priori knowledge of the environment. On the other hand, some researchers have addressed visibility constraints in reactive, purely sensor-based navigation strategies. For example, in [45], a homography-based visual controller drives a mobile robot to a goal pose by following some of the optimal paths proposed in [40]. Another solution, proposed in [46], consists in predicting the location of features, when they are occluded by static obstacles.

Aside the need for environment knowledge, another limitation of the cited works is that none addresses omnidirectional autonomous ground vehicles (AGVs). These systems, however, are getting increasingly popular in industrial applications, since they can perform complex three-dimensional trajectories, that are indispensable in limited footprint environments. It is therefore surprising that no attention has been paid to incorporating collision and occlusion constraints in the navigation of such platforms.

In this paper, we address the problem of dealing with both collisions and occlusions caused by static and moving obstacles, during visual navigation of an omnidirectional AGV equipped with fixed —*limited field of view*— on-board sensors. Specifically, the navigation task consists in *making the robot safely and autonomously navigate towards a static or moving target, in an unknown environment*. To solve this problem, we propose a generalization of the framework designed in [30] for car-like robots, by introducing *omnidirectional tentacles*. These tentacles are characterized not only by curvature, but also by course angle, since the robot linear velocity is not necessarily aligned with its heading, as in traditional nonholonomic systems. We also consider the visibility constraints that are induced by the limitations of the fixed sensors field of view. Furthermore, if the target is lost or occluded by an obstacle, a fast and efficient strategy is designed to estimate its current pose, by using its previous pose along with the control inputs. Let us summarize the constraints that are satisfied by our approach: 1) safety is ensured, by avoiding collisions with static and moving obstacles, 2) the target is maintained as much as possible within the sensors field of view even in presence of obstacles, 3) the loss or occlusion of the target is handled by continuous online estimation of its pose.

To our knowledge, this is the first work addressing and validating experimentally safe sensor-based navigation of omnidirectional robots under these three constraints. A preliminary version of our framework was presented in [48] without considering the target occlusion problem nor real vision (the target pose was estimated via odometry alone). Our main contributions with regards to that work are indicated hereby.

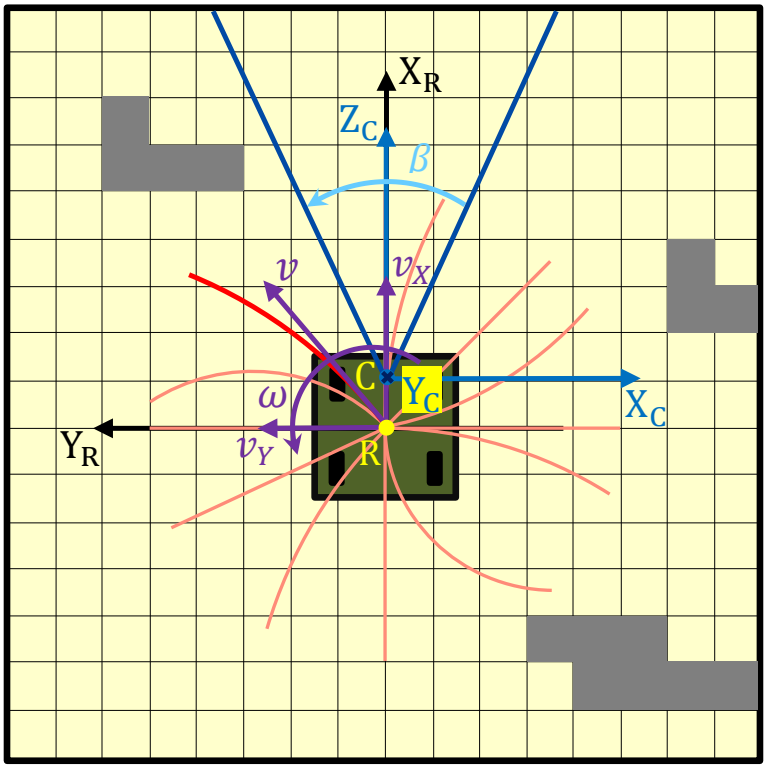

Fig. 1. Top view of the robot (green box with black wheels) with all the variables used in this work. We show the robot frame $\mathscr{F}_{\mathscr{R}}(R,X_R,Y_R)$, the occupancy grid (yellow), occupied cells (gray), some omnidirectional tentacles (red and orange), and the robot linear and angular velocities (purple). In this view, we assume the robot is following the red tentacle: this tentacle is tangent to $\mathbf{v}$, with radius $\|\mathbf{v}\|/\omega$. We also represent the visual sensors frame $\mathscr{F}_{\mathscr{C}}(C,X_C,Y_C,Z_C)$ (centered at C and blue) and the sensors' field of view of amplitude β (cyan).

- The development of an algorithm for detecting and tracking the target pose, using one or two cameras along with the lidar measures.
- The inclusion of target loss management to deal simultaneously with collision and occlusion.
- Analysis of the controller stability.

The remainder of the paper is organized as follows. In Sect. 2, the problem and relevant variables are defined. The omnidirectional generalization of tentacles is presented in Sect. 3, and we detail our controller in Sect. 4. In Sect. 5, we present the strategy for determining the best path (tentacle) to be tracked by the controller. Section 6 describes the algorithm used to detect and track the visual target. Experiments are presented in Sect. 7, and we finally conclude in Sect. 8.

2. General Definitions

2.1. Omnidirectional platform

We consider an omnidirectional wheeled robot, which can move in any direction on the ground plane. The reader is referred to Fig. 1. We define the robot frame $\mathscr{F}_{\mathscr{R}}(R,X_R,Y_R)$, which is rigidly constrained to the robot. It has center in R, the robot center of rotation, and axes X_R pointing forward and Y_R pointing left. The robot control inputs are:

$$\mathbf{u} = [v_X \; v_Y \; \omega]^\top, \tag{1}$$

where v_X and v_Y are the robot linear velocities in the direction of the axes X_R and Y_R respectively, and ω is the robot angular velocity (positive counterclockwise). The resultant linear velocity is noted $\mathbf{v} = [v_X \; v_Y]^\top$, and its norm is: $\|\mathbf{v}\| = \sqrt{v_X^2 + v_Y^2}$.

The robot is equipped with a set of forward looking visual sensors (e.g., one or more cameras) with a combined field of view β, centered at C. The visual sensors are fixed, so the robot heading also determines its viewing direction. We define the visual sensors frame $\mathscr{F}_{\mathscr{C}}(C, X_C, Y_C, Z_C)$. This frame is also rigidly linked to the robot chassis, with origin in C, Z_C pointing in the forward robot direction, X_C pointing right and Y_C downward[1]. The robot also has fixed on board distance sensors for building a local map of the obstacles surrounding it. This is the *occupancy grid*, shown in yellow in Fig. 1.

2.2. Visual navigation

The navigation task consists in driving towards a visible and possibly moving target, while avoiding the environment obstacles. When the environment is safe, the robot should progress forward, while visually tracking the target. In case avoidable obstacles are present, the robot should circumnavigate them, while maintaining the target in the field of view. If the target is lost or occluded by an obstacle, its current pose must be estimated by using its previous location along with the current control inputs. Finally, if collision is inevitable or if the target is not visible for a long time, the robot must stop.

The specifications of the navigation task are:

1. orient the robot so that it points the visual sensors towards the target,
2. make the robot go towards the target,
3. avoid collisions with the obstacles, while realizing 1 and 2.
4. estimate the current target pose at all times, including during sensor occlusions, in order to predict its pose in case of reappearance.

More formally, the navigation task (i.e., with the four specifications above) is defined in the robot frame $\mathscr{F}_{\mathscr{R}}$, instead of in a fixed world coordinate system. In particular, to fulfill specifications 1 and 2, the robot should move so that the visual target T (that can be static or moving) is displaced in $\mathscr{F}_{\mathscr{R}}$, from the current pose ${}^R\mathbf{p}_T = \left[{}^RX_T \; {}^RY_T \; {}^R\theta_T\right]^\top$ to a desired (final) *constant* pose, ${}^R\mathbf{p}_T^* = \left[{}^RX_T^* \; {}^RY_T^* \; {}^R\theta_T^*\right]^\top$. This final pose ${}^R\mathbf{p}_T^*$ is defined by the user depending on the application. For example, consider a mobile manipulator having to realize a manipulation on a table: the visual target T will be the table and ${}^R\mathbf{p}_T^*$ will be some table pose in the robot frame that makes the manipulator easily access the table. In summary, ${}^R\mathbf{p}_T^*$ is the desired target pose in the robot frame to be reached, to consider the task achieved.

We define (see Fig. 2 for a complete illustration of these variables):

1. the *distance between current and desired target positions*:

$$\rho^* = \sqrt{\left({}^RX_T - {}^RX_T^*\right)^2 + \left({}^RY_T - {}^RY_T^*\right)^2}, \tag{2}$$

2. the *orientation offset between current and desired target positions*, defined only when $\rho^* \neq 0$:

$$\alpha^* = \operatorname{atan2}\left({}^RY_T - {}^RY_T^*, {}^RX_T - {}^RX_T^*\right), \tag{3}$$

3. the *distance between robot and target position*:

$$\rho_T = \sqrt{{}^RX_T{}^2 + {}^RY_T{}^2}, \tag{4}$$

4. the *heading to the target*:

$$\alpha_T = \begin{cases} \operatorname{atan2}\left({}^RY_T, {}^RX_T\right) & \text{if } \left({}^RX_T, {}^RY_T\right) \neq (0,0), \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

[1] Note that if the robot is equipped only with one forward looking camera, this definition follows the pinhole camera frame convention.

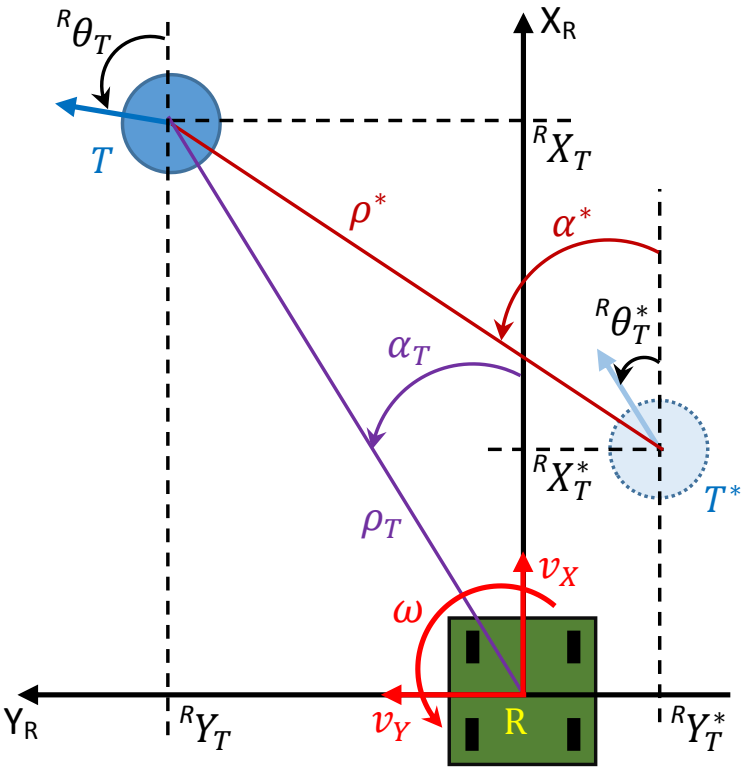

Fig. 2. Definition of the variables related to the visual target T: $\left[{}^R X_T \; {}^R Y_T \; {}^R\theta_T\right]^\top$ and $\left[{}^R X_T^* \; {}^R Y_T^* \; {}^R\theta_T^*\right]^\top$ are the current (variable) and desired (constant) poses of the target, (ρ^*, α^*) are the distance and orientation between the current and desired target positions, (ρ_T, α_T) are the polar coordinates of the current target position, (v_X, v_Y, ω) are the robot control inputs. All the variables are defined in the robot frame $\mathscr{F}_\mathscr{R}(R, X_R, Y_R)$.

2.3. Obstacle occupation times

For obstacle modeling, we use the occupancy grid shown in Fig. 1: it is linked to $\mathscr{F}_\mathscr{R}$, with cell sides parallel to X_R and Y_R. Its longitudinal and lateral extensions are limited ($X_m \leq X_R \leq X_M$ and $Y_m \leq Y_R \leq Y_M$), to ignore obstacles that are too far to be dangerous. Any grid cell $\mathbf{c}$ centered at (X, Y) is considered occupied if an obstacle has been sensed in $\mathbf{c}$.

The set of occupied cells with their estimated velocities, is denoted by state vector $\mathscr{O}$:

$$\mathscr{O} = \{\mathbf{c}_1, \ldots, \mathbf{c}_n, \dot{\mathbf{c}}_1, \ldots, \dot{\mathbf{c}}_n\}. \tag{6}$$

This is used to derive possible future collisions. Indeed, the estimations of the obstacles positions and velocities are updated at every iteration, by the Kalman observer designed in [33]. Then, for each $\mathbf{c}_i$ that may be occupied by an obstacle within time horizon T_h, we can predict initial $t_{i0}(\mathscr{O}) \in [0, T_h]$ and final $t_{if}(\mathscr{O}) \in [t_{i0}, T_h]$ *obstacle occupation times*, in function of the set of occupied cell states $\mathscr{O}$.

3. Omnidirectional Tentacles

We hereby present a generalization of the tentacles-based approach proposed in [30], to omnidirectional robots.

3.1. Definitions

We use a set of drivable paths (tentacles) both for perception and motion execution. Each tentacle j is a semicircle that starts in R and is tangent to the robot linear velocity $\mathbf{v}$. In contrast with the tentacles originally proposed in [30, 33], our omnidirectional tentacles are characterized not only by

their curvature (i.e., inverse radius)

$$\kappa_j = \begin{cases} \omega / \|\mathbf{v}\| & \text{if } \|\mathbf{v}\| \neq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

but also by their course angle

$$\alpha_j = \begin{cases} \text{atan2}\,(v_Y, v_X) & \text{if } \|\mathbf{v}\| \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Instead, on traditional nonholonomic systems, since the linear velocity is aligned with the heading $v_Y = 0$, thus $\alpha_j = 0$ for all control inputs (i.e., all tentacles are tangent to the heading).

Curvature κ_j belongs to $\mathscr{K}$, a uniformly sampled set[2]:

$$\kappa_j \in \mathscr{K} = \{-\kappa_M, \ldots, 0, \ldots, \kappa_M\} \tag{9}$$

and α_j belongs to $\mathscr{A}$, another uniformly sampled set:

$$\alpha_j \in \mathscr{A} = \{\alpha_{Min}, \ldots, 0, \ldots, \alpha_{Max}\} \subseteq [-\pi, \pi[. \tag{10}$$

The set of tentacles is $\mathscr{T} = \mathscr{K} \times \mathscr{A}$. The total number of tentacles in $\mathscr{T}$ is the product of the number of candidate curvatures by the number of candidate course angles. In Fig. 1, we show an example with 12 tentacles (5 linear and 7 circular tentacles) that could be followed by the robot. At each time instant, only one tentacle (drawn in red) is selected. As indicated in the figure, the robot linear velocity is tangent to this tentacle. Since tentacles are used both for perception and motion, a compromise between computational cost and control accuracy is needed to set the sizes of $\mathscr{K}$ and $\mathscr{A}$.

In Figure 3, a straight ($\alpha_j = 0$ and $\kappa_j = 0$), an oblique ($\alpha_j \neq 0$ and $\kappa_j = 0$) and a circular ($\kappa_j \neq 0$) tentacles are dashed. Each tentacle j is characterized by two classification areas (*collision* and *dangerous*), which are obtained by rigidly displacing, along the tentacle, two rectangular boxes, with increasing size. The boxes are overestimated with respect to the robot dimensions. For each tentacle j, the sets of cells belonging to the two classification areas (shown in Fig. 3) are noted $\mathscr{C}_j$ and $\mathscr{D}_j$[3]. The sets $\mathscr{O}$, $\mathscr{C}_j$ and $\mathscr{D}_j$ are used to calculate the variables required in our control law, as will be explained just below. In particular, the largest classification area $\mathscr{D}_j$ is used to select the safest tentacle and to assess its danger, while the thinnest one $\mathscr{C}_j$ determines the - eventually needed - deceleration.

3.2. Robot occupation times

For each dangerous collision cell in tentacle j (i.e., for each cell $\mathbf{c}_i \in \mathscr{D}_j$), we compute the *robot occupation time* t_{ij}. This is an estimate of the time at which the box will enter the cell, assuming the robot follows the tentacle at the current velocity. To calculate $t_{ij}\,(\mathbf{c}_i, v, \alpha_j, \kappa_j)$, we assume that the robot motion is uniform, and displace the box at its current velocities:

$$\begin{cases} v_X = \|\mathbf{v}\| \cos\alpha_j \\ v_Y = \|\mathbf{v}\| \sin\alpha_j \\ \omega_j = \kappa_j \|\mathbf{v}\|, \end{cases} \tag{11}$$

until the instant t_{ij} at which the box intersects cell $\mathbf{c}_i$.

3.3. Dangerous instants and collision instants

Once the obstacle and robot occupation times have been calculated for each cell, we can derive the earliest time instant at which a collision between obstacle and robot may occur on each tentacle j. By

[2] For algorithmic reasons, we bound the tentacle curvature to an arbitrarily large value κ_M, although omnidirectional robots can follow tentacles of infinite curvature (i.e., pivot in place) whenever $\|\mathbf{v}\| = 0$.
[3] For further details on the derivation of $\mathscr{C}_j$ and $\mathscr{D}_j$, refer to [33].

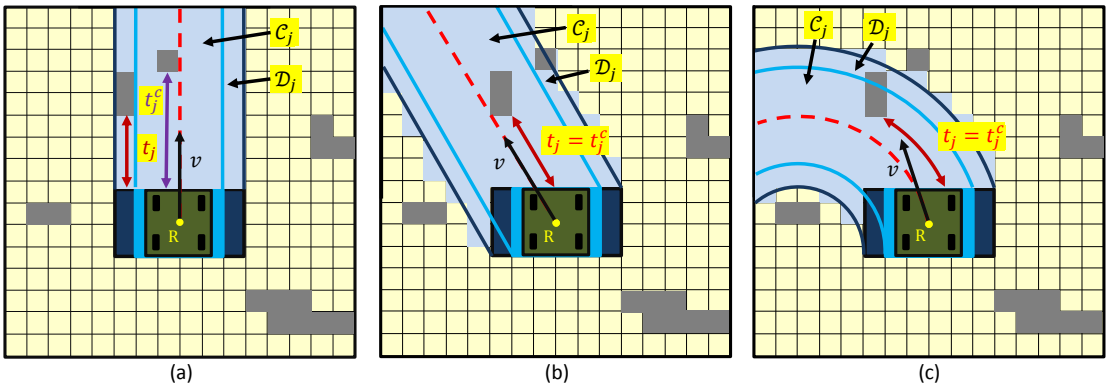

Fig. 3. Tentacles (dashed red), with classification areas, corresponding boxes and delimiting arcs of circle, and cells $c_i \in \mathscr{D}_j$ displayed in light blue. The robot linear velocity $\mathbf{v}$, the dangerous instant t_j and the collision instant t_j^c are also illustrated for three tentacles: (a) straight tentacle with $\alpha_j = 0$ and $\kappa_j = 0$, (b) oblique tentacle with $\alpha_j \neq 0$ and $\kappa_j = 0$, (c) circular tentacle with $\kappa_j \neq 0$.

either checking all cells in $\mathscr{D}_j$, or focusing just on $\mathscr{C}_j \subset \mathscr{D}_j$, and using variables t_{i0} and t_{if} introduced in Sect. 2.3, we discern between *dangerous instants* and *collision instants*. These are defined as:

$$t_j = \inf_{\mathbf{c}_i \in \mathscr{D}_j} \{t_{ij} : t_{i0} \leq t_{ij} \leq t_{if}\} \tag{12}$$

$$t_j^c = \inf_{\mathbf{c}_i \in \mathscr{C}_j} \{t_{ij} : t_{i0} \leq t_{ij} \leq t_{if}\} \tag{13}$$

respectively. In both cases, we seek the earliest time at which a cell is simultaneously occupied by the obstacle and by the robot. Computation of t_j and t_j^c is illustrated in the example of Fig. 3. These metrics give a good approximation of the time that the robot can travel along the tentacle without colliding. Further details are given in [33].

4. Control Scheme

4.1. Tentacle risk function

To assess the danger of each tentacle j, we design a *tentacle risk function*, by using t_j and tuned thresholds $t_d > 0$ and $t_s > t_d$ (d stands for dangerous, and s for safe):

$$H_j = \begin{cases} 0 & \text{if } t_j \geq t_s \\ \frac{1}{2}\left[1 + \tanh\left(\frac{1}{t_j - t_d} + \frac{1}{t_j - t_s}\right)\right] & \text{if } t_d < t_j < t_s \\ 1 & \text{if } t_j \leq t_d. \end{cases} \tag{14}$$

Note that H_j smoothly varies from 0, when possible collisions are in the far future, to 1, when they are forthcoming. If $H_j = 0$, tentacle j is tagged as *clear*.

To determine the best behaviour (among visual target tracking and obstacle avoidance) to adopt, we assess the danger of the environment via the risk function $H = H_v$ of the *visual task tentacle*, (κ_v, α_v). This is the tentacle that best approximates the visual path that the robot would follow to reach the target in absence of obstacles. Depending on the value of H (noted *situation risk function*), we distinguish the contexts (*safe* or *unsafe*) explained below. The derivation of the visual task tentacle is also explained hereby.

4.2. Safe context

In the *safe context* ($H = 0$), no dangerous obstacle is detected on the robot path. In this case, it is desirable that the robot realizes the task of driving ${}^R\mathbf{p}_T$ to ${}^R\mathbf{p}_T^*$, while keeping as much as possible T within its field of view.

Since the angular velocity ω determines both the convergence of ${}^R\theta_T$ to ${}^R\theta_T^*$ (for pose regulation) and that of α_T to 0 (for target visibility), a compromise must be reached. We weigh the two objectives respectively with a gain $\lambda_\omega \in [0,1]$ and with its complementary $1-\lambda_\omega$. Priority is given to target visibility when the desired position is farther than ρ_α, and to pose regulation when it is nearer than ρ_θ[4]. In between, we use a smoothing function, as in (14):

$$\lambda_\omega(\rho^*) = \begin{cases} 1 & \text{if } \rho^* \geq \rho_\alpha \\ \frac{1}{2}\left[1+\tanh\left(\frac{1}{\rho_\theta-\rho^*}+\frac{1}{\rho_\alpha-\rho^*}\right)\right] & \text{if } \rho_\theta < \rho^* < \rho_\alpha \\ 0 & \text{if } \rho^* \leq \rho_\theta. \end{cases} \tag{15}$$

The norm of the translation velocity $\mathbf{v}$ must be reduced, as the target is approached. We specify this through variable:

$$v_s(\rho^*) = \begin{cases} V & \text{if } \rho^* > \rho_v, \\ \frac{\rho^*}{\rho_v} V & \text{otherwise,} \end{cases} \tag{16}$$

with $V > 0$ the maximum desired value for v_s and $\rho_v > 0$ the distance at which the robot should slow down. Both V and ρ_v are easily tunable variables.

Finally, the translation velocity must be aligned with the heading towards the desired position α^*, when the target is far ($\lambda_\omega = 1$). As the target gets near ($\lambda_\omega < 1$), we also use the angular velocity ω to regulate ${}^R\theta_T$ to ${}^R\theta_T^*$. In summary, the robot control inputs in the safe context are:

$$\begin{cases} v_X = v_s \cos\alpha^* + (1-\lambda_\omega)\ \omega\ {}^RY_T \\ v_Y = v_s \sin\alpha^* - (1-\lambda_\omega)\ \omega\ {}^RX_T \\ \omega = \lambda_\omega\ \alpha_T + (1-\lambda_\omega)\left({}^R\theta_T - {}^R\theta_T^*\right). \end{cases} \tag{17}$$

The above *safe context* control law (17) is also used to define the *visual path curvature* κ_s and *course angle* α_s:

$$\begin{cases} \kappa_s = \left[\lambda_\omega\ \alpha_T + (1-\lambda_\omega)\left({}^R\theta_T - {}^R\theta_T^*\right)\right] / \|\mathbf{v}\| \\ \alpha_s = \operatorname{atan2}(v_Y, v_X). \end{cases} \tag{18}$$

Note that in general κ_s and α_s will not correspond to a tentacle belonging to set $\mathscr{T}$, but they will be used to find the nearest tentacle in $\mathscr{T}$, i.e., *the visual task tentacle* (κ_v, α_v) that characterizes the *risk function* H, as explained in Sect. 5.3.

4.3. Unsafe context

In the *unsafe context* ($H = 1$), dangerous obstacles are detected. The robot should circumnavigate them by following the best tentacle. The norm of the translation velocity must be reduced for safety reasons (i.e., to avoid collisions). We specify this by using a function $v_u \in [0, v_s]$:

$$\|\mathbf{v}\| = v_u = \begin{cases} v_s & \text{if } t_b^c \geq t_s^c \\ v_s\sqrt{t_b^c - t_d^c / t_s^c - t_d^c} & \text{if } t_d^c < t_b^c < t_s^c \\ 0 & \text{if } t_b^c \leq t_d^c \end{cases} \tag{19}$$

(with $t_d^c > 0$ and $t_s^c > t_d^c$ two thresholds corresponding to dangerous and safe collision times) to guarantee that the robot decelerates (and eventually stops) as the collision instant on the best tentacle t_b^c decreases. Then, the control inputs in the unsafe context are:

$$\begin{cases} v_X = v_u \cos\alpha_b \\ v_Y = v_u \sin\alpha_b \\ \omega = \kappa_b\ v_u, \end{cases} \tag{20}$$

[4] Both ρ_α and ρ_θ are pre-tuned, and $0 < \rho_\theta < \rho_\alpha$.

where α_b and κ_b are respectively the course angle and the curvature of the best tentacle. In Sect. 5, we will explain how we determine α_b and κ_b.

4.4. General control law

In *intermediate contexts* ($0 < H < 1$), the robot should navigate between the visual path, and the best tentacle. The transition between these two extremes will be driven by H. Using all the variables defined above, we can write our controller for visual navigation with obstacle avoidance:

$$\begin{cases} v_X = (1-H)\left(v_s\ \cos\alpha^* + (1-\lambda_\omega)\,\omega\ {}^R Y_T\right) + H\ v_u\ \cos\alpha_b \\ v_Y = (1-H)\left(v_s\ \sin\alpha^* - (1-\lambda_\omega)\,\omega\ {}^R X_T\right) + H\ v_u\ \sin\alpha_b \\ \omega = (1-H)\left(\lambda_\omega\ \alpha_T + (1-\lambda_\omega)\left({}^R\theta_T\ -\ {}^R\theta_T^*\right)\right) + H\ v_u\ \kappa_b. \end{cases} \tag{21}$$

When $H = 0$ this coincides with (17), and for $H = 1$, with (20).

4.5. Stability Analysis

Assuming that *the target is not moving in the environment*, we can write:

$$\begin{cases} {}^R\dot{X}_T = -v_X\ +\ \omega\ {}^R Y_T \\ {}^R\dot{Y}_T = -v_Y\ -\ \omega\ {}^R X_T \\ {}^R\dot{\theta}_T = -\omega. \end{cases} \tag{22}$$

Based on this hypothesis, we can derive three important stability properties of control law (21). Although this hypothesis is needed for the proofs, in the experiments we have also validated our framework in cases where the target is moving.

Property 1. In the safe context, if $\rho^* \geq \rho_\alpha$ (and therefore $\lambda_\omega = 1$) controller (21) guarantees asymptotic convergence of task vector $\mathbf{s} = [\rho^*\ \alpha_T]^\top$ to $\mathbf{s}^* = [\rho_\alpha\ 0]^\top$. Therefore, the robot will orient its heading towards the target while simultaneously approaching it until $\rho^* = \rho_\alpha$.

Proof. First, let us derive α_T with respect to time, using (5) (for $\rho_T \neq 0$), along with (22):

$$\dot{\alpha}_T = \frac{v_X\ {}^R Y_T\ -\ v_Y\ {}^R X_T}{{\rho_T}^2}\ -\ \omega. \tag{23}$$

Injecting the safe context controller (17) in (23) when $\lambda_\omega = 1$:

$$\dot{\alpha}_T = \frac{v_s}{{\rho_T}^2}\left({}^R Y_T\ \cos\alpha^*\ -\ {}^R X_T\ \sin\alpha^*\right)\ -\ \alpha_T. \tag{24}$$

Since ${}^R Y_T/\rho_T = \sin\alpha_T$ and ${}^R X_T/\rho_T = \cos\alpha_T$:

$$\dot{\alpha}_T = \frac{v_s}{\rho_T}\ \sin(\alpha_T - \alpha^*)\ -\ \alpha_T. \tag{25}$$

Note that when $\rho^* \geq \rho_\alpha$, $\alpha_T \approx \alpha^*$, since the Euclidean distance between points R and T^* becomes neglectable with respect to ρ^* (see Fig.2). Hence, replacing this in (25), we can consider the following approximation:

$$\dot{\alpha}_T = -\alpha_T. \tag{26}$$

Now, let us derive ρ^* with respect to time, using (4):

$$\dot{\rho}^* = \frac{\left({}^R X_T\ -\ {}^R X_T^*\right){}^R\dot{X}_T\ +\ \left({}^R Y_T\ -\ {}^R Y_T^*\right){}^R\dot{Y}_T}{\rho^*}. \tag{27}$$

From (3) if $\rho^* \neq 0$ (as is the case here):

$$\begin{cases} {}^R X_T \; - \; {}^R X_T^* = \rho^* \cos\alpha^* \\ {}^R Y_T \; - \; {}^R Y_T^* = \rho^* \sin\alpha^*. \end{cases} \tag{28}$$

Injecting this in (27) yields:

$$\dot{\rho}^* = -\left(v_X \; \cos\alpha^* + v_Y \; \sin\alpha^*\right) \; + \; \omega \; \rho_T \; \sin\left(\alpha_T - \alpha^*\right), \tag{29}$$

and with the safe context controller (17) since $\alpha_T \approx \alpha^*$, the previous equation can be approximated as:

$$\dot{\rho}^* = -v_S. \tag{30}$$

Finally, let us consider the Lyapounov function $\mathscr{V} = \mathbf{e}^\top \mathbf{e}/2$ where $\mathbf{e} = \mathbf{s} - \mathbf{s}^*$. Its time derivative is:

$$\dot{\mathscr{V}} = \dot{\mathbf{e}}^\top \mathbf{e} = \begin{bmatrix} \dot{\rho}^* & \dot{\alpha}_T \end{bmatrix} \begin{bmatrix} \rho^* - \rho_\alpha \\ \alpha_T \end{bmatrix}. \tag{31}$$

Injecting (26) and (30) in (31) yields:

$$\dot{\mathscr{V}} = -v_s \left(\rho^* - \rho_\alpha\right) - \alpha_T^2, \tag{32}$$

which is clearly negative definite for $\mathbf{e} \neq 0$ when $\rho^* \geq \rho_\alpha$. □

An alternative control strategy would have consisted in applying the linear velocity that cancels the effect of ω in (22):

$$\begin{cases} v_X = v_s \; \cos\alpha^* + \omega \; {}^R Y_T \\ v_Y = v_s \; \sin\alpha^* - \omega \; {}^R X_T. \end{cases} \tag{33}$$

Although this controller guarantees asymptotic convergence of both $\omega \; {}^R X_T$ and $\omega \; {}^R Y_T$ to 0, it makes the value of α_T constant throughout navigation. Indeed, plugging (33) in (23) yields:

$$\dot{\alpha}_T = \frac{v_s}{\rho} \sin\left(\alpha_T - \alpha^*\right), \tag{34}$$

which is neglectable, since $\alpha_T \approx \alpha^*$. In summary, controller (33) does not center the target in the field of view, as one would wish (particularly for far target).

Property 2. In the safe context, if $\rho^* \leq \rho_\theta$, controller (21) guarantees asymptotic convergence of task vector $\mathbf{s} = \left[\rho^* \; {}^R\theta_T\right]^\top$ to $\mathbf{s}^* = \left[0 \; {}^R\theta_T^*\right]^\top$. Hence, the robot will asymptotically converge to the desired pose with respect to the target.

Proof.
When $\lambda_\omega = 0$ (since $\rho^* \leq \rho_\theta$), injecting the safe context controller (17) in 29 yields:

$$\dot{\rho}^* = -v_s, \tag{35}$$

and from the last equation of (22):

$${}^R\dot{\theta}_T = \; {}^R\theta_T^* \; - \; {}^R\theta_T. \tag{36}$$

Consider the same Lyapounov function as above. Since ${}^R\mathbf{p}_T^*$ is constant, the Lyapounov function derivative is:

$$\dot{\mathscr{V}} = \dot{\mathbf{e}}^\top \mathbf{e} = \begin{bmatrix} \dot{\rho}^* & {}^R\dot{\theta}_T \end{bmatrix} \begin{bmatrix} \rho^* \\ {}^R\theta_T \; - \; {}^R\theta_T^* \end{bmatrix}. \tag{37}$$

If $\rho^* \neq 0$, injecting (35) and (36) in (37) yields:

$$\dot{\mathscr{V}} = -v_s \rho^* - \left({}^R\theta_T \; - \; {}^R\theta_T^*\right)^2, \tag{38}$$

which is clearly negative definite for $\mathbf{e} \neq 0$.
If $\rho^* = 0$, ${}^R X_T = {}^R X_T^*$ and ${}^R Y_T = {}^R Y_T^*$. Hence, $\dot{\mathscr{V}} = -\left({}^R\theta_T \; - \; {}^R\theta_T^*\right)^2$, which is also negative definite for $\mathbf{e} \neq 0$. □

Property 3. In the unsafe context, if moving ($\|\mathbf{v}\| \neq 0$), the robot will precisely follow the best tentacle (κ_b, α_b).

Proof.
When $H = 1$, (21) becomes (20), and it is trivial to see, by applying (7) and (8) that whenever $\|\mathbf{v}\| \neq 0$, the curvature and course angle will respectively coincide with κ_b and α_b. □

These three properties outline some nice features of our framework. For instance, consider the safe context. Property 1 shows that from far, the robot will orient its heading towards the target (to keep it within its field of view), while approaching it. When the robot is near enough to the target, Property 2 is triggered. Then, the robot will point its heading towards the desired relative orientation, while converging to the desired position with respect to the target.

It is noteworthy to mention that the properties are valid in given operating situations (H and λ_ω either null or unitary). It is very difficult (and out of scope here) to study analytically the behaviour of the system in the intermediate situations, i.e., when H and λ_ω are different from 0 or 1. However, the smooth design of these two functions, as well the extensive experimental validation of (21) (see Sect. 7) make us very confident about the global performance of our framework.

In the next section, we explain the strategy that is adopted to select the best tentacle to be followed in the unsafe context.

5. Tentacles Selection Strategy

Here, we describe our strategy for selecting the best tentacle (κ_b, α_b), that allows the robot to avoid the obstacles while simultaneously performing the visual task. To this end, we first define a metric for sorting the tentacles. This is explained in Sect. 5.1. The visibility constraints to be satisfied by the best tentacle are given in Sect. 5.2. These criteria are finally used to select the best tentacle, as explained in Sect. 5.3.

5.1. Sorting tentacles

To apply any search algorithm to set $\mathscr{T}$, in order to find *the best tentacle* (κ_b, α_b), the set must be somehow sorted. To this end, we must find a metric characterizing similarity (or difference) among tentacles. This is a non-trivial task, since $\mathscr{T}$ is defined in a discrete non-Euclidean space:

$$\mathscr{T} = \mathscr{K} \times \mathscr{A} \subset \mathbb{R} \times \mathbb{SO}(2), \tag{39}$$

where a distance metric cannot be defined, without weighing – arbitrarily – the relative importance of a scalar (κ_j) and an angle (α_j). To solve this, we rely on the following property.

Property 4. Let an omnidirectional robot be modeled as a point. Its Cartesian position, after having followed tentacle (κ_j, α_j) at velocity $\|\mathbf{v}\|$ for an iteration $\Delta t << 1$, depends only on the scalar function: $\phi_R = \alpha_j + \frac{\|\mathbf{v}\| \, \Delta t}{2} \kappa_j$.

Proof. Since we model the robot as a point, only changes in the robot position (not orientation) are considered. We start by calculating the position change after Δt, in function of (κ_j, α_j). To this end, we define a fixed frame $\mathscr{F}_\mathscr{O}(O, X_O, Y_O)$ that corresponds to the robot frame at the beginning of the iteration (see Fig. 4). For each tentacle j, we can calculate the robot position ${}^O\mathbf{p}_R = \left[{}^O X_R \; {}^O Y_R\right]^\top$ in $\mathscr{F}_\mathscr{O}$ after it has followed the tentacle for time Δt. We can relate the derivative of the robot pose in

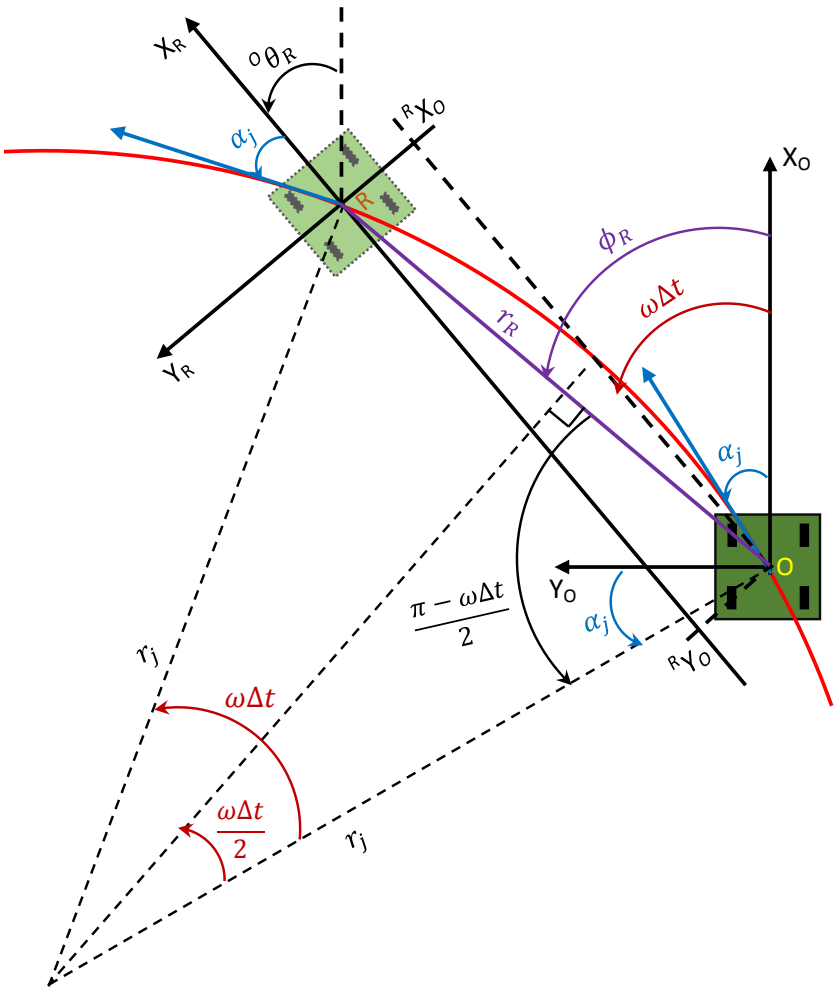

Fig. 4. Variables needed for tentacle selection.

$\mathscr{F}_{\mathcal{O}}$ and the control inputs:

$$\begin{cases} {}^O\dot{X}_R = v_X \cos\left({}^O\theta_R\right) - v_Y \sin\left({}^O\theta_R\right) \\ {}^O\dot{Y}_R = v_X \sin\left({}^O\theta_R\right) + v_Y \cos\left({}^O\theta_R\right) \\ {}^O\dot{\theta}_R = \omega. \end{cases} \tag{40}$$

Equation (40) can be derived from the kinematic model of omnidirectional mobile robots [47]. This model is generally used to calculate the robot pose with respect to a fixed reference frame by using odometry. In this paper, we employ this model to predict the future robot position, after one sampling time, with respect to the robot pose at the beginning of the sample time. Using (11), we can rewrite (40) in function of the followed tentacle, i.e., in function of κ_j and α_j:

$$\begin{cases} {}^O\dot{X}_R = \|\mathbf{v}\| \left(\cos\alpha_j \cos\left({}^O\theta_R\right) - \sin\alpha_j \sin\left({}^O\theta_R\right)\right) \\ {}^O\dot{Y}_R = \|\mathbf{v}\| \left(\cos\alpha_j \sin\left({}^O\theta_R\right) + \sin\alpha_j \cos\left({}^O\theta_R\right)\right) \\ {}^O\dot{\theta}_R = \|\mathbf{v}\| \kappa_j. \end{cases} \tag{41}$$

By integrating this expression, we can obtain the robot position at Δt. We distinguish two cases according to κ_j:

- if $\kappa_j = 0$ (i.e. the tentacle is a straight line), the robot position will be:

$$\begin{cases} {}^O X_R = \|\mathbf{v}\| \Delta t \cos\alpha_j \\ {}^O Y_R = \|\mathbf{v}\| \Delta t \sin\alpha_j; \end{cases} \tag{42}$$

- if $\kappa_j \neq 0$, the robot position will be:

$$\begin{cases} {}^{O}X_R = 2/\kappa_j \, \sin(\kappa_j \, \|\mathbf{v}\| \, \Delta t/2) \, \cos(\alpha_j + \kappa_j \, \|\mathbf{v}\| \, \Delta t/2) \\ {}^{O}Y_R = 2/\kappa_j \, \sin(\kappa_j \, \|\mathbf{v}\| \, \Delta t/2) \, \sin(\alpha_j + \kappa_j \, \|\mathbf{v}\| \, \Delta t/2). \end{cases} \tag{43}$$

Converting the position to polar coordinates (r_R, ϕ_R), with $r_R = \sqrt{{}^{O}X_R^2 + {}^{O}Y_R^2}$ and $\phi_R = \text{atan2}({}^{O}Y_R, {}^{O}X_R)$ (see Fig. 4) yields

$$r_R(\alpha_j, \kappa_j) = \frac{2}{\kappa_j} \, \sin\left(\frac{\kappa_j}{2} \, \|\mathbf{v}\| \, \Delta t\right), \tag{44}$$

$$\phi_R(\alpha_j, \kappa_j) = \alpha_j + \frac{\|\mathbf{v}\| \, \Delta t}{2} \, \kappa_j. \tag{45}$$

Since the robot movement is small during an iteration (as $\Delta t << 1$), $\sin\left(\frac{\kappa_j}{2} \, \|\mathbf{v}\| \, \Delta t\right) \approx \frac{\kappa_j}{2} \, \|\mathbf{v}\| \, \Delta t$, and therefore r_R will not vary much from a tentacle to the other ($r_R \approx \|\mathbf{v}\| \, \Delta t$). Hence, the robot position after having followed the tentacle at velocity $\|\mathbf{v}\|$ for Δt will depend only on ϕ_R. □

As a consequence of Property 4, we have decided to sort the tentacles according to their values of ϕ_R, calculated using (45), and noted $\phi_{R,j}$ for each tentacle j.

5.2. Tentacles guaranteeing visibility

In this section, we introduce the visibility constraints in our obstacles avoidance strategy. To ensure that the target is not lost during navigation, the best tentacle should keep the target in sight. However, this is not possible for all tentacles, since the robot is constrained to look in its heading direction. Here, we consider as tentacles that guarantee visibility, those that will keep the target in the field of view after an iteration Δt.

For a given tentacle j, we define (again, refer to Fig. 4):

1. $\left({}^{O}X_T, {}^{O}Y_T, {}^{O}\theta_T\right)$: the target pose (at $t = 0$) in the initial robot frame $\mathscr{F}_{\mathscr{O}}(R_O, X_O, Y_O)$
2. $\left({}^{R}X_T, {}^{R}Y_T, {}^{R}\theta_T\right)$: the target pose (at $t = \Delta t$) in the robot frame $\mathscr{F}_{\mathscr{R}}(R_R, X_R, Y_R)$.

The relation between these two poses is given by :

$$\begin{bmatrix} {}^{R}X_T \\ {}^{R}Y_T \\ {}^{R}\theta_T \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\omega\Delta t) & \sin(\omega\Delta t) & 0 & {}^{R}X_O \\ -\sin(\omega\Delta t) & \cos(\omega\Delta t) & 0 & {}^{R}Y_O \\ 0 & 0 & 1 & -\omega\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{O}X_T \\ {}^{O}Y_T \\ {}^{O}\theta_T \\ 1 \end{bmatrix} \tag{46}$$

where $\left({}^{R}X_O, {}^{R}Y_O\right)$ are the coordinates of the origin of $\mathscr{F}_{\mathscr{O}}$ in $\mathscr{F}_{\mathscr{R}}$. After Δt these can be expressed in polar coordinates as:

$$\begin{cases} {}^{R}X_O = -r_R \cos(\phi_R - \omega \, \Delta t) \\ {}^{R}Y_O = -r_R \sin(\phi_R - \omega \, \Delta t). \end{cases} \tag{47}$$

Injecting (47) in (46) and using (45), we derive the target position $\left({}^{R}X_T, {}^{R}Y_T\right)$ in $\mathscr{F}_{\mathscr{R}}$ after Δt:

$$\begin{cases} {}^{R}X_T = \cos(\kappa_j \, \|\mathbf{v}\| \, \Delta t) \, {}^{O}X_T + \sin(\kappa_j \, \|\mathbf{v}\| \, \Delta t) \, {}^{O}Y_T \\ \qquad -2/\kappa_j \, \sin(\kappa_j \, \|\mathbf{v}\| \, \Delta t/2) \, \cos(\alpha_j - \kappa_j \, \|\mathbf{v}\| \, \Delta t/2) \\ {}^{R}Y_T = -\sin(\kappa_j \, \|\mathbf{v}\| \, \Delta t) \, {}^{O}X_T + \cos(\kappa_j \, \|\mathbf{v}\| \, \Delta t) \, {}^{O}Y_T \\ \qquad -2/\kappa_j \, \sin(\kappa_j \, \|\mathbf{v}\| \, \Delta t/2) \, \sin(\alpha_j - \kappa_j \, \|\mathbf{v}\| \, \Delta t/2), \end{cases} \tag{48}$$

For the target to be visible, its position in the robot frame must satisfy the following constraints:

$$\begin{cases} {}^{R}X_T \; > \; {}^{R}X_C \\ {}^{R}Y_T > {}^{R}Y_C - ({}^{R}X_T - {}^{R}X_C)\tan\left(\frac{\beta}{2}\right) \\ {}^{R}Y_T < {}^{R}Y_C + ({}^{R}X_T - {}^{R}X_C)\tan\left(\frac{\beta}{2}\right), \end{cases} \tag{49}$$

with $\left({}^{R}X_C, {}^{R}Y_C\right)$ the sensors center coordinates in $\mathscr{F}_{\mathscr{R}}$ and β their field of view (see Fig. 1). In summary, to determine whether tentacle j guarantees target visibility, it is sufficient to inject its (κ_j, α_j) into (48) and then check if the resulting ${}^{R}X_T$ and ${}^{R}Y_T$ verify constraints (49).

5.3. Selecting the best tentacle

Algorithm 1 Selecting the best tentacle

input: Sorted set of tentacles $\mathscr{T}$, visual path $(\alpha_s,\ \kappa_s)$, previous best tentacle $(\alpha_p,\ \kappa_p)$, and target pose ${}^{O}\mathbf{p}_T$.
output: Risk function H and best tentacle $(\alpha_b,\ \kappa_b)$ (if $H \neq 0$).

$N \leftarrow$ FindVisibilityGuaranteeingTentacles($\mathscr{T}$, ${}^{O}\mathbf{p}_T$);
if $N > 5$ **then**
  $\mathscr{T} \leftarrow$ RemoveOtherTentacles($\mathscr{T}$);
end if
$\phi_{R,s} \leftarrow$ CalculateSortingAngle($\alpha_s,\ \kappa_s$);
$\phi_{R,v} \leftarrow$ FindNearestSortingAngle($\mathscr{T}$, $\phi_{R,s}$);
$(\alpha_v, \kappa_v) \leftarrow$ CalculateTentacleParameters($\phi_{R,v}$);
$H \leftarrow$ CalculateRiskFunction($\alpha_v,\ \kappa_v$);
$(\alpha_b, \kappa_b) \leftarrow null$;
if $H \neq 0$ **then**
  $\phi_{R,p} \leftarrow$ CalculateSortingAngle($\alpha_p,\ \kappa_p$);
  if $\phi_{R,v} < \phi_{R,p}$ **then**
    $(\phi_i,\ \phi_f) \leftarrow (\phi_{R,v},\ \phi_{R,p})$;
  else
    $(\phi_i,\ \phi_f) \leftarrow (\phi_{R,p},\ \phi_{R,v})$;
  end if
  $(\alpha_b, \kappa_b) \leftarrow$ FindNearestClearTentacle($\mathscr{T}$, $[\phi_i,\ \phi_f]$);
  if $(\alpha_b, \kappa_b) = null$ **then**
    $(\alpha_b, \kappa_b) \leftarrow$ FindNearestClearTentacle($\mathscr{T}$,
    $]\phi_{min}, \phi_i[\cup]\phi_f,\ \phi_{max}[$);
  end if
end if
return $(H,\ (\alpha_b, \kappa_b))$;

Let us describe the best tentacle selection strategy.
At *initialization*, all tentacles in set $\mathscr{T} = \mathscr{K} \times \mathscr{A}$ are sorted according to their $\phi_{R,j}$, calculated with (45). Then, *at each iteration* Δt we proceed as follows (see Algorithm 1).

1. The subset of all tentacles that guarantee the visibility constraints is derived as explained in Sect. 5.2. If the number of tentacles in this subset is sufficient (at least 5 in this work), all other tentacles are removed from $\mathscr{T}$. Otherwise, the entire set $\mathscr{T}$ is kept, to privilege obstacle avoidance over occlusion avoidance. This choice is motivated on one hand by obvious safety reasons, and on the other by the possibility – offered by the method that will be explained in the next Section – of estimating the target pose even when it is not visible.
2. For the path that the robot would perform if there were no obstacles, i.e., if the safe context controller (17) was applied, we compute: the course angle α_s , curvature κ_s using (18), and polar angular coordinate $\phi_{R,s}$.

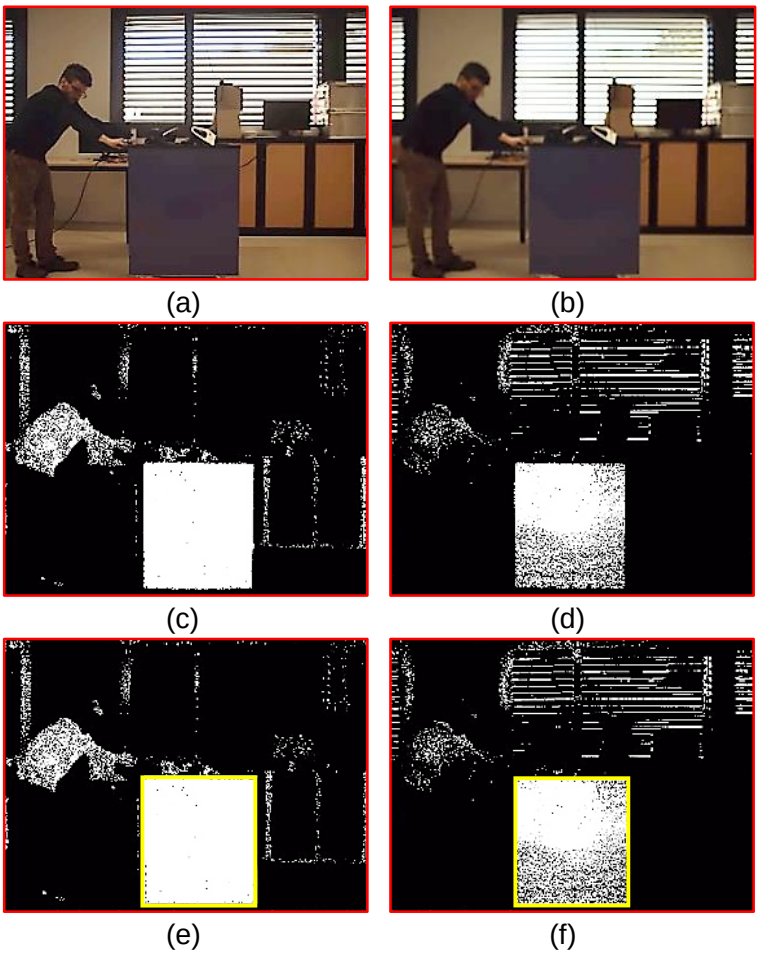

Fig. 5. Target detection and tracking in the left and right images.(a-b) Raw images. (c-d) Blob detection. (e-f) Cart contour extraction.

3. The tentacle that best approximates the visual path in $\mathscr{T}$ (i.e., the nearest in terms of ϕ_R) is computed; we denote it as the *visual task tentacle* (κ_v, α_v) and calculate its situation risk function as H_v using (14). This is also the value that will be used in control law (21): $H = H_v$.
4. If $H = 0$, the visual task tentacle is clear and the safe context controller (17) can be applied.
5. Instead, if $H \neq 0$, we seek a clear tentacle ($H_j = 0$).
   a) First, we search among the tentacles with $\phi_{R,j}$ between the one of the visual task tentacle and that of the best tentacle at the previous iteration. b) If there are many clear tentacles, the nearest to the visual task tentacle is chosen. c) If none is clear, we search among the others (those that are not between the visual task and the previous best tentacles). Again, the best tentacle will be the clear one closest to the visual task tentacle.

6. Target Detection and Tracking

In this section, we describe our approach for detecting and tracking over time the target pose in the robot frame, ${}^R\mathbf{p}_T$. This is done by the robot visual sensor/s with an image processing algorithm described in Sect. 6.1. If the target is not visible, because it exits the field of view or is occluded, an alternative method is applied, as explained in Sect. 6.2.

6.1. Vision-based target detection

Our target detection algorithm relies on the following assumptions:

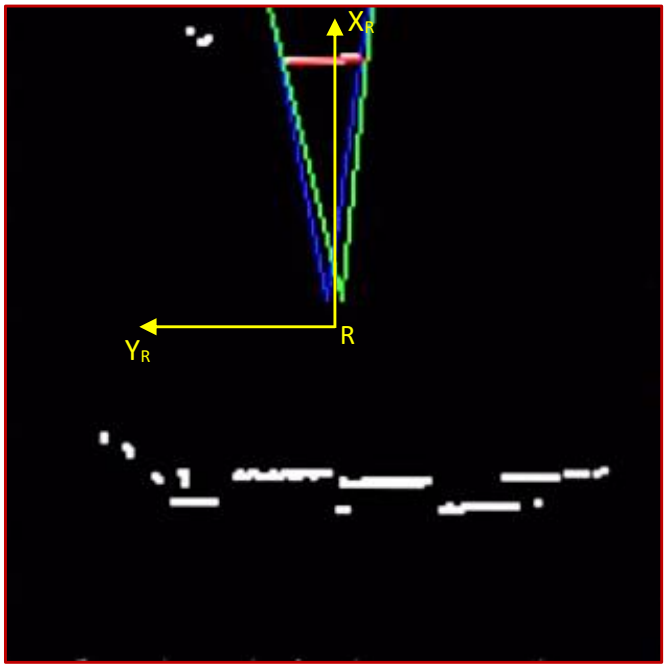

Fig. 6. Detecting the cart face segment using lidar measurements.

- the target is characterized by a predominant color,
- the visible part of the target is planar,
- the target height is known,
- as position of the target $({}^{R}X_T, {}^{R}Y_T)$, we take the Cartesian coordinates of the visible plane in $\mathscr{F}_R$,
- as target orientation ${}^{R}\theta_T$, we take the angular offset from X_R to the normal vector entering the visible plane.

The described target detection and tracking algorithm has been developed for the sensor suite available on our self-made robotic platform BAZAR [49]. This is composed of a forward looking fixed pair of 55.8° field of view cameras operating at 30 Hz, and two 270° Hokuyo lidars placed at opposite corners of the base, and operating at 40 Hz. The algorithm relies on both cameras, and on the forward looking lidar. The lidar however, is used only when the target is near, to obtain a precise estimation of ${}^{R}\mathbf{p}_T$. The reason is that, although lidar measurements are more precise on a short range, they are biased by inaccuracy and false positives on the long range. On the other hand, from far, when only vision is used, orientation ${}^{R}\theta_T$ cannot be properly estimated. Note that controller (21) does not require ${}^{R}\theta_T$ as long as $\rho^* \geq \rho_\alpha$. Hence, the most intuitive design choice is to start using the lidar only when $\rho^* < \rho_\alpha$. While $\rho^* \geq \rho_\alpha$, only the target position $({}^{R}X_T, {}^{R}X_T)$ is calculated from vision without using lidar data. Even if this estimation is imprecise, it will make the robot approach the target, while orienting its heading towards it until, for $\rho^* < \rho_\alpha$, the lidar will be activated and ${}^{R}\theta_T$ will also be estimated, to drive the robot to the desired pose with respect to the target.

For image processing, the same steps (developed with OpenCV[5]) are applied to the left and right cameras. These steps are detailed below and illustrated in Fig. 5.

1. The images are binarized with a logic conjunction (*and* operator) of predefined ranges in the Hue, Saturation and Value color space. This way, pixels of a certain color (blue in Figures 5c and 5d) can be isolated.
2. A dilation and an erosion are applied to the binary image to form blobs of pixels of the target's color.
3. The blob with largest contour is extracted (Fig. 5e, 5f).
4. Knowing the real target height and the camera intrinsic and extrinsic parameters, the position of the front face centroid is projected in $\mathscr{F}_R$ to obtain $({}^{R}X_T, {}^{R}Y_T)$.

[5] `http://opencv.org/`

5. If the blob is detected by both cameras, the average of the two positions is used.

As the robot approaches the target and $\rho^* < \rho_\alpha$, the front lidar is also utilized, to improve the position, and add the orientation estimates. We proceed as follows (see Fig. 6):

1. A high resolution occupancy grid (cell size 3×3 cm) is built from lidar measurements.
2. The two side borders of the blob are projected in the grid for both the left (blue cone in Fig. 6) and right (green) cameras.
3. The Hough transform is applied to detect all line segments in the two cones.
4. The extreme (nearer and farther) corners among all these line segments are used to define a region of interest, where linear regression on the raw lidar data is used to find the visible target projection (red segment in Fig. 6).
5. From this segment, the target pose ${}^R\mathbf{p}_T$ is derived.
6. A Kalman Filter (with state vector composed of ${}^R\mathbf{p}_T$ and ${}^R\dot{\mathbf{p}}_T$) is finally used to ensure continuity and robustness of the target pose estimation.

6.2. Dealing with total loss of the target

If none of the tentacles that guarantee robot safety can satisfy the visibility constraints, the target will exit the field of view and the above approach cannot be applied. The same applies if obstacles on the robot path provoke a partial or total target occlusion. To overcome both problems, we estimate the target pose $\left({}^RX_T, {}^RY_T, {}^R\theta_T\right)$ in the robot frame by using its previous pose $\left({}^OX_T, {}^OY_T, {}^O\theta_T\right)$ and control inputs $(v_X,\ v_Y,\ \omega)$. This is done by integrating the target pose over time interval Δt, using (46) and (47): if $\omega \neq 0$:

$$\begin{cases} {}^RX_T = \cos(\omega\Delta t)\,{}^OX_T \ + \ \sin(\omega\Delta t)\,{}^OY_T - 2\|\mathbf{v}\|/\omega\ \sin(\omega\Delta t/2)\ \cos(\alpha - \omega\Delta t/2) \\ {}^RY_T = -\sin(\omega\Delta t)\,{}^OX_T + \cos(\omega\Delta t)\,{}^OY_T - 2\|\mathbf{v}\|/\omega\ \sin(\omega\Delta t/2)\ \sin(\alpha - \omega\Delta t/2) \\ {}^R\theta_T = {}^O\theta_T \ - \ \omega\ \Delta t, \end{cases} \tag{50}$$

with α defined as in (8), but in the general case (not just $\alpha \in \mathscr{A}$).

Injecting these equations in the same Kalman Filter as above (with state vector composed of ${}^R\mathbf{p}_T$ and ${}^R\dot{\mathbf{p}}_T$), it is possible to deal with the target total loss, by predicting its location without affecting the robot behaviour. In practice, we use (50) in control law (21).

7. Experimental Validation

Here, we report the simulated and real experiments that we performed to validate our approach. These are also shown in the video attached to this paper[6]. In all tests, we use the same setup and parameters, the only difference being the wheeled platform: KUKA youBot in simulation and Neobotix MPO700 (the base of our BAZAR platform) in real experiments. The target is a blue wheeled cart, similar to the one used for kitting in automotive manufacturing by our partner PSA Peugeot Citroën. It is automatically moved in simulations, and manually pushed in the real experiments. For simplicity, we position the target in the sensors field of view at the beginning of each simulation/experiment. A "target searching" navigation procedure, out of scope here, could be devised as future work.

Both the simulated KUKA youBot and real Neobotix MPO700 are equipped with the BAZAR sensor suite described in Sect. 6.1. In the visibility constraint equation (49), we use as C the midpoint of the 2 optical centers and as β (conservatively) the field of view of the cameras (55.8°). The algorithm is implemented on a computer with an Intel Xeon E5 2620 V3 2.4 Ghz Hexacore processor with integrated graphics card (without GPU acceleration) and 32 Gb of RAM. We set to $\Delta t = 200$ ms the sampling time, that includes visual processing (Sect. 6), tentacle processing (Sections 3 and 5) and control (Sect. 4) in a unique thread. This choice of Δt is motivated by the limitations of the graphics card, since the average computational times of the algorithms are: 100 ms for visual processing, 60 ms for tentacle processing and less than 1 ms for control. One way of reducing Δt would consist in optimizing the computing time of the vision and tentacle processing pipelines by using a NVidia GeForce GTX 1080 graphics card along with the Opencv GPU API (instead of CPU API). Another improvement would consist in computing the omnidirectional tentacles (from laser scanners) and

[6] Also visible online at `https://youtu.be/K9yoNlqkqSI`

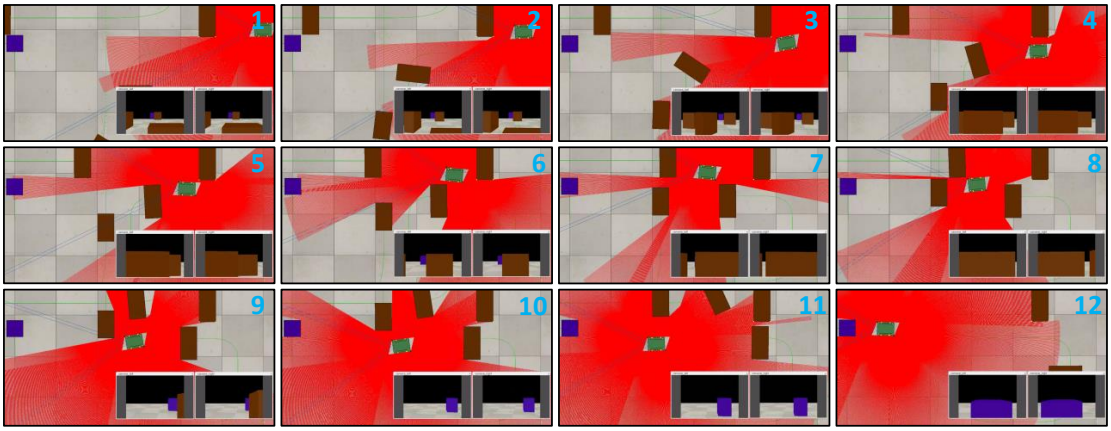

Fig. 7. V-REP Scenario A: the robot (green box), equipped with two cameras (blue cones) and lidar scanners (red lines) navigates towards a static target (blue) while avoiding collisions and dealing with occlusions caused by one static and three moving obstacles (all brown).

visual target tracking (from RGB camera) in parallel threads that would be synchronized before the last control steps.

The obstacle detecting occupancy grid (with $X_M = Y_M = 3$ m, $X_m = Y_m = -3$ m, and cell size 10×10 cm) is built by projecting the lidar readings from the two lidars. We use 147 tentacles (7 curvatures $\times$ 21 course angles), with $\kappa_M = 0.4\ \text{m}^{-1}$, $\alpha_{Min} = -170°$, and $\alpha_{Max} = 170°$ The maximum translational velocity, that is applied in the safe environment, and far from the target using (16) is $V = 0.4\ \text{ms}^{-1}$. For the situation risk function we use $t_s = 3.5$ s and $t_d = 3$ s, for the unsafe translational velocity $t_s^c = 2.25$ s and $t_d^c = 1.5$ s, and for the visual task $\rho_\alpha = 3.5$ m $\rho_\theta = 1$ m and $\rho_v = 2$ m.

7.1. Simulation

Let us recall that the main goal is to perform vision-based omnidirectional navigation in the presence of static or moving – and possibly occluding – obstacles. To this end, preliminary simulations were carried out in V-REP[7], with the KUKA youBot, an omnidirectional robot with four omni wheels. In V-REP, we have designed three scenarios A, B, and C shown respectively in Fig. 7, Fig. 10, and Fig. 11, along with the images acquired by the robot cameras during the simulations. In these figures, the robot, target and obstacles are represented respectively by green, blue and brown boxes. In scenario A, the visual target is static, whereas in scenarios B and C, it is moving. In all scenarios, the robot is able to perform the visual task and to reach the desired relative pose ${}^R\mathbf{p}_T^*$, while avoiding collisions and managing occlusions provoked by the obstacles. We hereby describe the robot behaviour for each scenario.

7.1.1. Scenario A. The environment is cluttered, with four obstacles that may partially or totally occlude the target. The closest obstacle is static, while the three others move. We have plotted, in Fig. 8, the robot control inputs during this navigation task. The plot shows that after approximately 2 s, the robot is deviated and oriented so that the camera is pointing at the target. When it approaches the first obstacle (at $t = 3$ s), it starts avoiding it while keeping the target in the camera field of view. Then, the robot progresses toward the desired pose before avoiding the second obstacle which induces a total occlusion of the target. Nevertheless, the robot performs its task thanks to the target pose estimation module described in Sect. 6.2. Afterwards, the robot succeeds in avoiding the last two obstacles and reaches the target despite the occlusions they cause (see images 7 and 8 in Fig. 7). Then (at $t = 33$ s), the environment is free again, and the visual task can be performed for the rest of the experiment. At the end, the robot decelerates and stops when the desired pose relative to the target has been reached. The target position error $\left({}^RX_T - {}^RX_T{}^*, {}^RY_T - {}^RY_T{}^*\right)$ and orientation error $\left({}^R\theta_T - {}^R\theta_T{}^*\right)$ are plotted in Fig. 9. Notice that the target orientation ${}^R\theta_T$ is not estimated until $t \approx 27$ s since before that, $\rho^* \geq \rho_\alpha$, as explained in Sect. 6.1. As we can see in these plots, all the errors

[7] `http://www.coppeliarobotics.com`

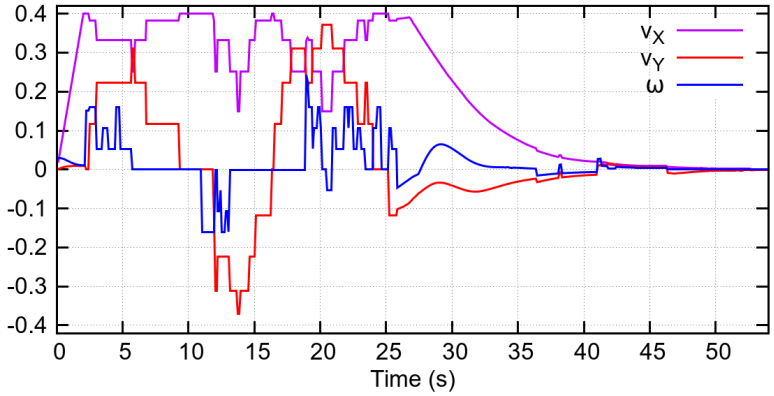

Fig. 8. V-REP Scenario A: robot control inputs (linear velocities v_X and v_Y in m/s and angular velocity ω in rad/s) during the simulation.

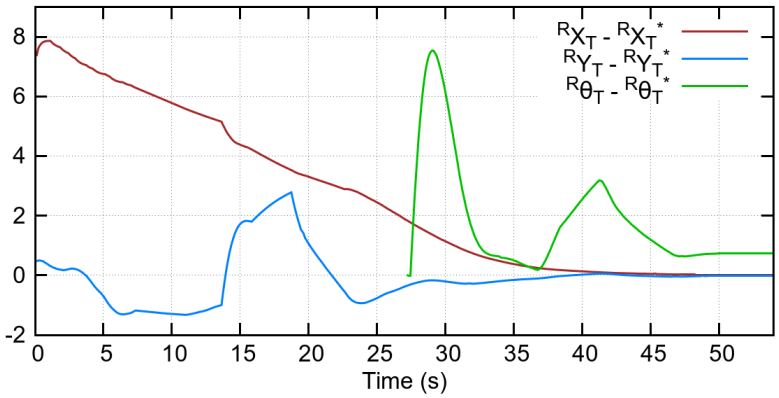

Fig. 9. V-REP Scenario A: evolution of position (in meters) and orientation (in degrees) errors of the target in the robot frame.

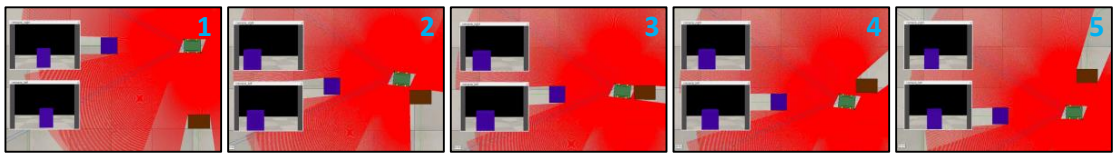

Fig. 10. V-REP Scenario B: the robot performs lateral avoidance of a moving obstacle while maintaining visibility of a moving target.

converge to zero while the robot is driving to the target. This result is coherent, as the visual task consists in moving the target from the initial to the desired pose in $\mathcal{F}_R$.

7.1.2. Scenario B. This scenario (shown in Fig. 10) is designed to test lateral collision avoidance by exploiting the omnidirectional characteristic of the robot. The target is moving in the lateral direction with regards to the robot (roughly in the direction parallel to the Y_R axis) and an obstacle approaches the robot in the same direction, but opposite verse. After reaching the desired pose relative to the moving target, the controller deviates the robot to maintain its relative heading (hence to guarantee target visibility) while avoiding the moving obstacle. When the environment is clear again, the robot returns to the desired relative pose.

7.1.3. Scenario C. Here, we consider navigation in a corridor (light purple in Fig. 11), common in indoor environments. The target is moving in the corridor, and the robot must follow it, while avoiding the walls as well as two brown boxes. The first one moves perpendicularly to the path

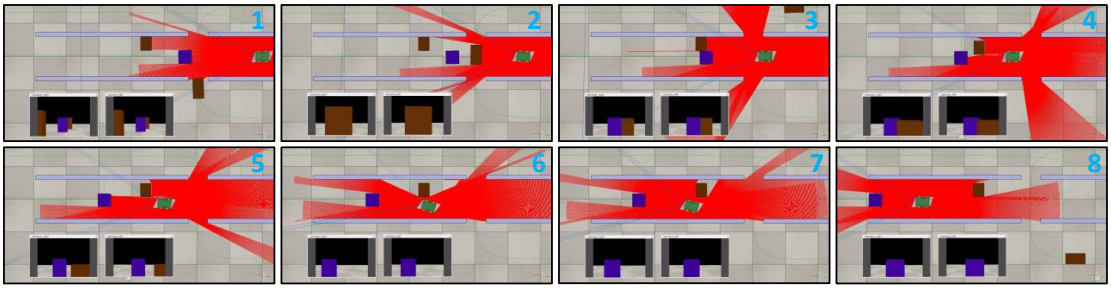

Fig. 11. V-REP Scenario C: the robot follows a moving target in a cluttered corridor with two obstacles (one moving, one static).

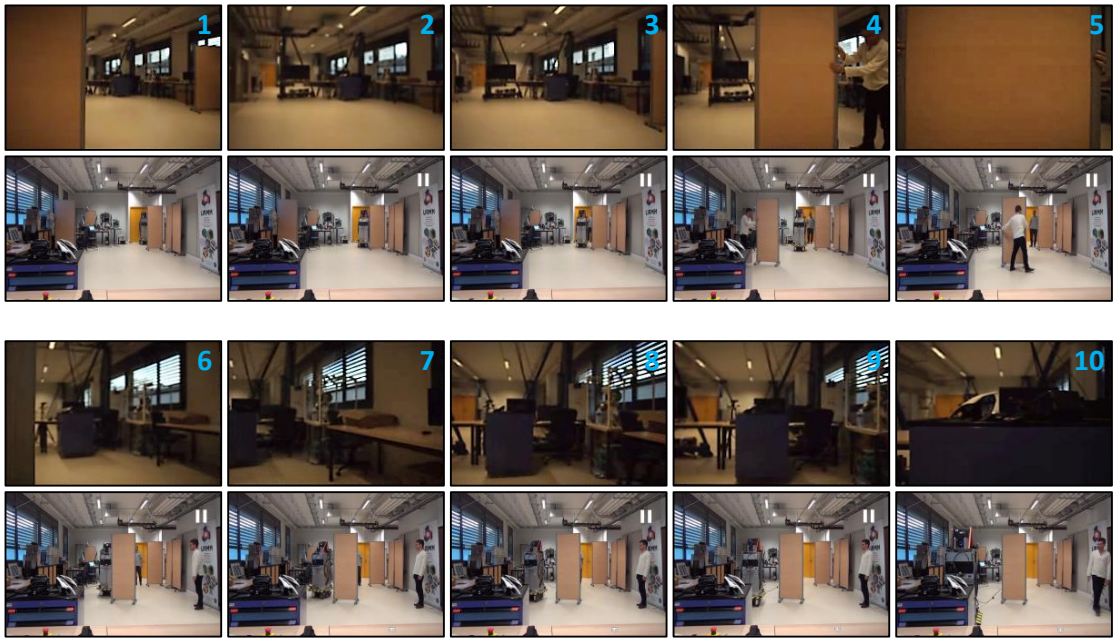

Fig. 12. Real Scenario A: the MPO700 platform navigates safely towards a static blue chariot. Top: images from the left camera, bottom: the MPO700 (grey) avoids two obstacles while dealing with total target occlusion.

and induces an occlusion, while the second one is static and presents a risk of collision. Despite the limited mobility within this environment, the robot is capable of dealing with the occlusion and avoiding both obstacles, while successfully following the target.

7.2. Real experiments

After the simulations, we have validated our approach in extensive real experiments, carried out on our Neobotix MPO700 platform. This is an omnidirectional robot with 4 steerable wheels, hence with a reduced mobility as compared to the omni wheels platform used in V-REP. The MPO700 wheels are driven by a low level controller that runs on the embedded PC at a rate of 40 Hz. Instead, our controller and sensor processing algorithms run on an on-board PC that sends the velocity commands (v_X, v_Y, ω) to the MPO700 PC, via ethernet. We have designed three scenarios:

7.2.1. Scenario A. The MPO700 navigates towards the – static – blue cart, with two obstacles (brown panels) present in the environment. The purpose of the experiment is to validate our framework when the robot has to handle simultaneously collisions and occlusions. We show in Fig. 12 the scenario as well as the images acquired by the left camera during navigation. The control inputs are plotted in Fig. 13. At the beginning (snapshots 1-3), since there is no risk of collision or occlusion, the robot is deviated (v_Y) and oriented (ω) so that the cameras points at the blue chariot. The first obstacle is then circumnavigated without provoking occlusions nor target loss. After that, we

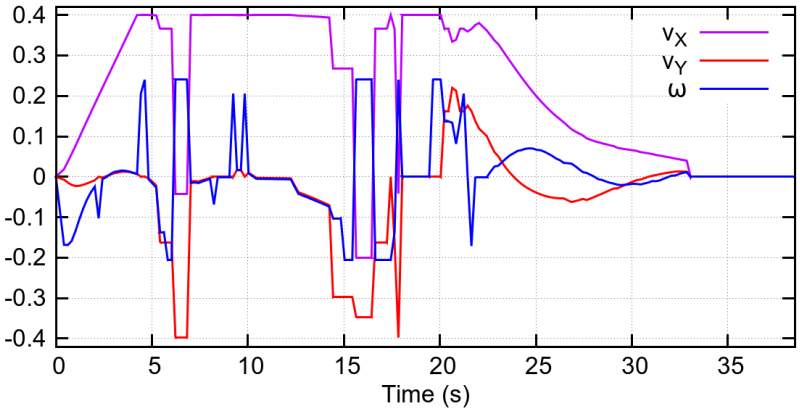

Fig. 13. Real Scenario A: MPO700 Control inputs (linear velocities v_X and v_Y in m/s and angular velocity ω in rad/s).

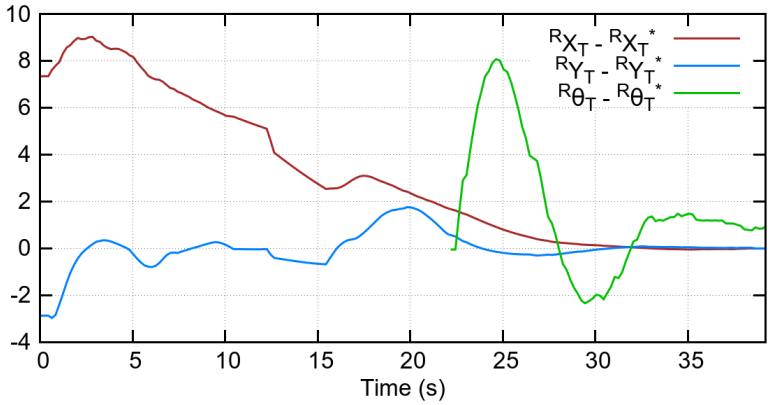

Fig. 14. Real Scenario A: evolution of position (in meters) and orientation (in degrees) errors of the target in the robot frame.

Fig. 15. Real Scenario B: lateral obstacle avoidance while maintaining the blue chariot in sight.

manually move the second obstacle to totally occlude the target (snapshots 4 and 5). As can be seen, the robot successfully predicts the target location, while avoiding the obstacle. Finally, when there is no more risk of collisions or occlusions, the MPO700 converges to the blue cart (snapshots 6-10). The evolution of the target position and orientation errors, during the navigation task, are plotted in Fig. 14. It can be seen that all three errors converge to zero as desired. As shown in Fig. 13 , the control inputs generated by our controller are not smooth. This is due to the nature of our approach, that is based on sampling a set of drivable paths. Since the sample time of our controller is higher than that of the steering wheels controller, it could be possible to filter this signal at low level. This will be done as future work.

7.2.2. Scenario B. In the second experiment, the navigation task consists in exploiting the omnidirectional characteristic of the MPO700, to follow and keep in sight a moving target, while avoiding a lateral moving obstacle (a pedestrian). As shown in Fig. 15, the robot successfully performs the visual task in spite of the walking person.

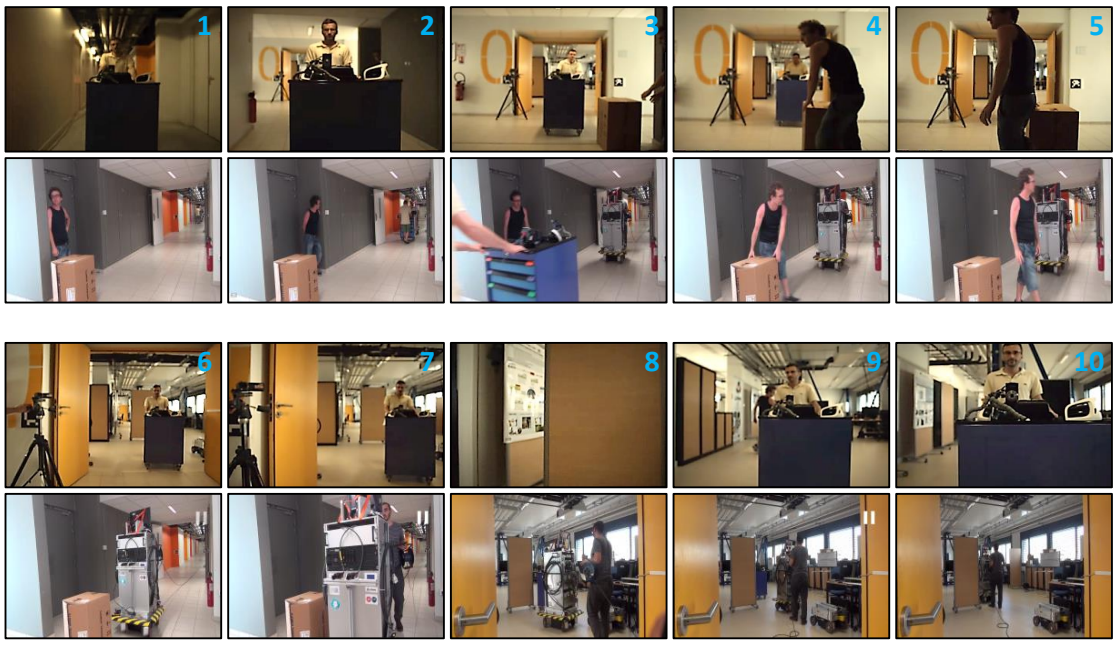

Fig. 16. Real Scenario C: the MPO700 follows a moving target in a cluttered environment, composed first of a corridor and then of a crowded hall.

7.2.3. Scenario C. A final very challenging experiment was carried out to assess the performance in a complex environment, including first a corridor and then a cluttered hall (see Fig. 16). The MPO700 starts by following the blue chariot that is moving along the corridor (snapshots 1-4). A first moving obstacle (human) crosses the robot path and occludes the target without affecting its behaviour (snapshot 5). The robot also succeeds in avoiding a second obstacle, while following and keeping the target in its field of view until reaching the hall (snapshot 6 and 7). Then, the MPO700 is in a difficult situation where an obstacle crosses its way, while the target has slightly changed direction (snapshot 8). This leads to a complete loss of target visibility. Once again, the robot manages to avoid the obstacle and to recover the chariot, thanks to the target pose estimation module. Finally, it reaches the desired pose with respect to the target (snapshots 9 and 10), after having avoided all obstacles while either keeping the target visibility or predicting its location.

8. Conclusions

We have presented a framework that guarantees obstacle avoidance during visual navigation of an omnidirectional wheeled robot that has to deal with visibility constraints. Additionally, the proposed framework can deal with partial and total visual occlusions provoked by the obstacles. For both perception and motion execution, we design *omnidirectional tentacles* that exploit the kinematics of the platform. Simulated and real experiments show that the robot is able to perform the task, with safety and smoothness, in spite of occlusions. Future work will investigate how to extend the proposed framework to Multi-Target Tracking (i.e., looking at multiple targets) in the presence of static and moving obstacles.

Acknowledgements

This work was funded by the EU Horizon 2020 research and innovation program under grant 731330 (Versatile Project), by the Algerian Ministry of Higher Education and Scientific Research, and by PSA Peugeot Citroën.

References

1. S. M. LaValle, “Planning Algorithms”(Cambridge, U.K.: Cambridge Univ. Press, 2006).

2. T.T, Mac, C. Copot, D.T Tran, R. De Keyser, "Heuristic approaches in robot path planning: A survey," it Robot. Auton. Syst., 86, pp. 13–28 (2016).
3. C. Sprunk, B. Lau, P. Pfaff, W. Burgard, "An accurate and efficient navigation system for omnidirectional robots in industrial environments", *Autonomous Robots* (2016), pp. 1–21.
4. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", *IEEE Int. Conf. on Robotics and Automation* (1985).
5. A. Cherubini and F. Chaumette, "Visual Navigation with a time-independent varying reference", in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS* (2009).
6. J. Borenstein and Y. Koren, "The Vector Field Histogram - Fast obstacle avoidance for mobile robots", *IEEE Trans. on Robotics and Automation*, vol. 7, no. 3 (1991), pp. 278-288.
7. D. Fox, W. Burgard and S. Thrun, "The Dynamic Window approach to obstacle avoidance", in *IEEE Robotics and Automation Magazine*, vol. 4, no. 1 (1997), pp. 23–33.
8. J. Minguez, "The Obstacle-Restriction Method (ORM) for robot obstacle avoidance in difficult environments", *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS* (2005).
9. M. Mujahad, D. Fischer, B. Mertsching and H. Jaddu "Closest Gap based (CG) reactive obstacle avoidance navigation for highly cluttered environments", *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS* (2010).
10. M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, pp. 463–497 (2015).
11. P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.* 17(7) (1998), pp. 760–772.
12. F. Large, C. Lauger and Z. Shiller, "Navigation among moving obstacles using the NLVO: Principles and applications to intelligent vehicles," *Auton. Robots* 19(2) (2005), 159–171.
13. W. Zhang, S. Wei, Y.Teng, J. Zhang, X. Wang and Z. Yan "Dynamic Obstacle Avoidance for Unmanned Underwater Vehicles Based on an Improved Velocity Obstacle Method," it Sensors, 17(12), 2742 (2017).
14. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. IEEE ICRA* (2006), pp. 2366–2371
15. N. E. Du Toit and J. W. Burdick, "Robot motion planning in dynamic, uncertain environments," *IEEE Trans. Robot.* 28(1) (2012) pp. 101–115.
16. C. Fulgenzi, A. Spalanzani, C. Laugier, "Probabilistic motion planning among moving obstacles following typical motion patterns", in *IEEE/RSJ Int. Conf. on Intelligent RObots and Systems* (2009).
17. A. Foka and P. Trahanias, "Probabilistic autonomous robot navigation in dynamic environments with human motion prediction," *Int. J. Soc. Robot.* 2(1), pp. 79–94 (2010).
18. B. Kim and J. Pineau, "Socially adaptive path planning in human environments using inverse reinforcement learning," *International Journal of Social Robotics* , vol. 8, no. 1, pp. 51–66 (2015).
19. Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS* (2017).
20. S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Auton. Robots* 13(3), pp. 207–222 (2002).
21. J. Ren, K A. McIsaac and R. V. Patel, "Modified Newtons method applied to potential field-based navigation for nonholonomic robots in dynamic environments," *Robotica* 26(1), pp. 117–127 (2008).
22. L. Xin, Y. Yin, C.J. Lin, "A new potential field method for mobile robot path planning in the dynamic environment," *Asian J. Control* 11 (2), pp. 214–225 (2009).
23. Q. Zhang, S. Yue, Q. Yin, Y. Zha, "Dynamic obstacle-avoiding path planning for robots based on modified potential field method," *Intell. Comput. Theor. Technol.* 7996, pp. 332–342 (2013).
24. A. V. Savkin and C. Wang, "A simple biologically inspired algorithm for collision-free navigation of a unicycle-like robot in dynamic environments with moving obstacles," *Robotica* 31(6), pp. 993–1001 (2013).
25. O. Montiel, R.U. Orozco, R. Sepulveda, "Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles," *Expert Syst. Appl.*, 42, pp. 51775191 (2015).
26. F. Chaumette and S. Hutchinson, "Visual servo control, Part I: Basic approaches", *IEEE Robotics and Automation Magazine*, vol. 13 , no. 4, pp. 82–90 (2006).
27. F. Chaumette and S. Hutchinson, "Visual servo control, Part II : Advanced approaches", *IEEE Robotics and Automation Magazine*, vol. 14 , no. 1, pp. 109–118 (2007).
28. D. Folio and V. Cadenat, "A redundancy-based scheme to perform safe vision-based tasks amidst obstacles", *IEEE Int. Conf. on Robotics and Biomimetics*, Kunming, China (2006).
29. M. Futterlieb, V. Cadenat, T. Sentenac, "A Navigational Framework Combining Visual Servoing and Spiral Obstacle Avoidance Techniques", *Int. Conf. on Informatics in Control, Automation and Robotics* (2014).
30. A. Cherubini and F. Chaumette. "Visual navigation of a mobile robot with laser-based collision avoidance", *Int. Journal of Robotics Research*, vol. 32 no. 2 (2013), pp. 189–205.
31. A. Cherubini, F. Spindler and F. Chaumette, "A New Tentacles-based Technique for Avoiding Obstacles during Visual Navigation", *IEEE Int. Conf. on Robotics and Automation, ICRA* (2012).
32. A. Cherubini, B. Grechanichenko, F. Spindler and F. Chaumette. "Avoiding moving obstacles during Visual Navigation", *IEEE Int. Conf. on Robotics and Automation, ICRA* (2013).
33. A. Cherubini, F. Spindler and F. Chaumette. "Autonomous Visual navigation and Laser-based moving obstacle avoidance", *IEEE Trans. on Int. Transportation Systems*, vol. 15, no. 5, pp. 2101–2110 (2014).
34. Y. Mezouar and F. Chaumette, "Avoiding self-occlusions and preserving visibility by path planning in the image", *Robotics and Autonomous Systems*, vol. 41, no. 2, p. 77–87 (2002).

35. G. Chesi and K. Hashimoto, "Keeping features in the field of view in eye-in-hand visual servoing: a switching approach", *IEEE Trans. On Robotics*, vol. 20, no. 5, pp. 908–914 (2004).
36. A. Remazeilles, N. Mansard, and F. Chaumette, "Qualitative visual servoing: application to the visibility constraint", in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS* (2006).
37. M. Kazemi, K. Gupta, and M. Mehrandezh, "Randomized kinody- namic planning for robust visual servoing", *IEEE Trans. on Robotics*, vol. 29, no. 5, pp. 1197–1211 (2013).
38. O. Kermorgant and F. Chaumette, "Dealing with constraints in sensor- based robot control", *IEEE Trans. on Robotics*, vol. 30, no. 1, pp. 244–257 (2014).
39. N. Cazy, P.-B. Wieber, P. R. Giordano, F. Chaumette, "Visual Servoing when Visual Information is Missing: Experimental Comparison of Visual Feature Prediction Schemes", *IEEE Int. Conf. on Robotics and Automation, ICRA* (2015).
40. S. Bhattacharya, R. Murrieta-Cid, and S. Hutchinson, "Optimal paths for landmark-based navigation by differential-drive robots with field-of-view constraints", *IEEE Trans. on Robotics*, 23 (1), pp. 47–59 (2007).
41. P. Salaris, D. Fontanelli, L. Pallottino, and A. Bicchi, "Shortest paths for a robot with nonholonomic and field-of view constraints", *IEEE Trans. on Robotics*, vol. 26, no. 2, pp. 269–280 (2010).
42. J. Hayet, C. Esteves, and R. Murrieta-Cid, "A motion planner for maintaining landmark visibility with a differential drive robot", in *Algorithmic Foundations of Robotics VIII*. Berlin, Germany: Springer (2009).
43. J.-B. Hayet, H. Carlos, C. Esteves, and R. Murrieta-Cid, "Motion planning for maintaining landmarks visibility with a differential drive robot", *Robotics and Autonomous Systems*, 62 (4), pp. 456–473 (2014).
44. D. Panagou, V. Kumar, "Cooperative visibility maintenance for leader-follower formations in obstacle environments", *IEEE Trans. on Robotics*, vol. 30, no. 4, pp. 831–844 (2014).
45. G. Lopez-Nicolas, N. R. Gans, S. Bhattacharya, C. Sags, J. J. Guerrero, and S. Hutchinson, "An optimal homography-based control scheme for mobile robots with nonholonomic and field-of-view constraints", *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, 40 (4), pp. 1115–1127 (2010).
46. D. Folio and V. Cadenat, "A sensor-based controller able to treat total image loss and to guarantee non-collision during a vision-based navigation task", in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS* (2008).
47. C. Canudas de Wit, B. Siciliano, G. Bastin, "Theory of Robot Control, Berlin", Springer-Verlag, 1996.
48. A. Khelloufi, N. Achour, R. Passama, and A. Cherubini, "Tentacle-based moving obstacle avoidance for omnidirectional robots with visibility constraints", in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS* (2017).
49. A. Cherubini, A. Crosnier, P. Fraisse, B. Navarro, R. Passama and M. Sorour, "Research on cobotics at the LIRMM IDH group", in *ICRA (2017) Workshop IC3 Industry of the future*.