Article

The International Journal of Robotics Research
2018, Vol. 37(8) 867–889
© The Author(s) 2018
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/0278364918775520
journals.sagepub.com/home/ijr
SAGE

Model-based robocentric planning and navigation for dynamic environments

María-Teresa Lorente, Eduardo Owen and Luis Montano

Abstract
This work addresses a new technique of motion planning and navigation for differential-drive robots in dynamic environments. Static and dynamic objects are represented directly on the control space of the robot, where decisions on the best motion are made. A new model representing the dynamism and the prediction of the future behavior of the environment is defined, the dynamic object velocity space (DOVS). A formal definition of this model is provided, establishing the properties for its characterization. An analysis of its complexity, compared with other methods, is performed. The model contains information about the future behavior of obstacles, mapped on the robot control space. It allows planning of near-time-optimal safe motions within the visibility space horizon, not only for the current sampling period. Navigation strategies are developed based on the identification of situations in the model. The planned strategy is applied and updated for each sampling time, adapting to changes occurring in the scenario. The technique is evaluated in randomly generated simulated scenarios, based on metrics defined using safety and time-to-goal criteria. An evaluation in real-world experiments is also presented.

Keywords
Motion planning, dynamic environments, robot navigation strategies, velocity space

1. Introduction

Applications of autonomous robotic systems are a recognized research priority (e.g. in contexts relating rescue, surveillance, households, guidance, museums, and factories). Robots in these contexts have to coexist or cooperate with humans or other moving vehicles in constantly changing scenarios. Planners for static scenarios and purely reactive avoidance techniques are no longer valid. Motion planning in dynamic scenarios requires being able to predict the future evolution of obstacles, to plan and execute the safest and quickest feasible movements for the robot. Furthermore, kinodynamic constraints must be taken into account to obtain feasible trajectories. In this work we develop a robocentric planning and navigation technique for such environments, which considers safety, manoeuvrability, and constraints on the robot and the environment. The paper offers two main contributions: a new dynamic environment model for non-holonomic robots, which maps the environment on the velocity space of the robot; and a set of planning and navigation strategies, based on the model and on safety and time-to-goal criteria.

In Section 2 the state of the art is presented. In Section 3 the contributions are highlighted and explained. Section 4 establishes the problem to be solved. Section 5 formally defines the dynamic environment model, the dynamic object velocity space (*DOVS*), and its properties. In Section 6, the decision making, planning and navigation strategies are developed. Section 7 evaluates the method, based on metrics by means of simulations and in real-world experiments. In Section 8 some conclusions and future work are presented.

2. Related work

Motion planning and reactive navigation in static and dynamic environments have been addressed extensively in the literature. Reactive approaches combined with global planning techniques lead to so-called iterative motion planners (e.g. Brock and Khatib, 2000; Fraichard, 1998; Frazzoli et al., 2001; Hsu et al., 2002; Minguez and Montano, 2005; Stachniss and Burgard, 2002). These techniques calculate several steps ahead depending on the time available. The planner evaluates different branches in a tree within a horizon and works out a partial trajectory. This process may be

Instituto de Investigación en Ingeniería de Aragón, University of Zaragoza, Spain

Corresponding author:
María-Teresa Lorente, Instituto de Investigación en Ingeniería de Aragón, C/Mariano Esquillor s/n 50018, Zaragoza, Spain.
Email: mlorente@unizar.es

interrupted at any time or is time-bounded to maintain the robot's reactivity.

Modeling the dynamic environment is a major problem to solve that requires representing the dynamics of the robotic system and the motion evolution of the moving obstacles. Two of the most commonly used and referenced models are the *velocity obstacle* (*VO*) (Fiorini and Shiller, 1998), expanded in Shiller et al. (2001), and the *inevitable collision states* (*ICS*) introduced by Fraichard and Asama (2003). In *VO* the set of velocities from the current state that would provoke a collision between a holonomic robot and an obstacle at a time in the future is computed. *ICS* is directly related to safety.

The *non-linear velocity obstacle* (*NLVO*), which is the *VO* concept applied to obstacles moving along arbitrary trajectories, is used in Large et al. (2002) to compute a risk function from the time to collision to select the best velocity for the robot. An evolved technique that employs the 3D information obtained by the *NLVO* is presented in Large et al. (2005). The authors developed an iterative approach based on the *A** algorithm to incrementally build a solution towards the goal, where real-time issues are also considered. Recently, Kim and Oh (2016) presented a new *VO*-based approach (*OVVO*), which improves the velocity selection compared with the *VO* approach. The method defines a cost function which weighs the similarity to a desired velocity and the level of obstacle avoidance (to move around obstacles to a greater or lesser extent) to decide on the best velocity for the robot in crowded or less-dense scenarios. However, the results are very dependent on weight heuristic parameters.

Regarding safety, LaValle and Kuffner (2001) introduced the concept of *region of inevitable collision* to identify the states in which there is no control input the robot can apply to avoid a collision. Frazzoli et al. (2001) formulated the concept of τ-safety as a guarantee of no collision during τ seconds for each node of a tree created with a sampling-based algorithm. However, the *ICS* model focuses on safety by computing the robot states that lead to inevitable collision in a time horizon, for a set of *evasive manoeuvres*, including the braking manoeuvre.

Fraichard (2007) introduced three safety criteria that a robotic system should consider to compute its future motion, and *ICS* is presented as a suitable framework to deal with motion safety for motion planning and navigation. Petti and Fraichard (2005), Kalisiak and van de Panne (2007), Chan et al. (2008), and Bekris and Kavraki (2007) exploited an approximation of *ICS* in different ways. They perform a tree-based search and check for inevitable collision states to provide safety for a time horizon. Martinez-Gomez and Fraichard (2009) presented *ICS*-Avoid, a reactive navigation approach that selects controls for the robot to move through *ICS*-free states. Probabilistic versions of *ICS* were defined in Althoff et al. (2010) and Bautin et al. (2010) to capture the uncertainty about the future behavior of moving objects in real-world situations. Althoff et al. (2012) considered interactions among objects in terms of collision avoidance, reasoning beyond the planning horizon. Bouraine et al. (2012) defined *Braking-ICS*, a version of *ICS* that guarantees that if a collision takes place, the robot will be at rest. It was later applied to the development of a partial motion planner by Bouraine et al. (2014).

Shiller et al. (2010) addressed the motion safety issue in the *VO* framework using the *ICS* concept for holonomic robots. Calculating the proper time horizon as the minimum time for the robot velocity to exit the velocity obstacle yields a representation of the environment that ensures robot safety as long as the robot velocity does not lie in the velocity obstacle. Seder and Petrovic (2007) adapted the *DWA* approach (Fox et al., 1997) to consider moving obstacles, which are represented as moving cells in a grid map. Safety is thus considered by computing intersection points between simulated obstacle and robot trajectories in the short term. The cells resulting in collision are marked as forbidden, and a command optimizing a weighted navigation criterion is selected for every sampling time.

Pallottino et al. (2007), Van den Berg and Overmars (2008), Bekris et al. (2009), and Bareiss and Van den Berg (2013), addressed multi-robot systems in which all the agents are capable of making avoidance decisions. They developed coordination schemes to distribute the avoidance actions among the robots. Bareiss and Van den Berg (2015) extended this work, generalizing the reciprocal collision avoidance for robots with different kinematic constraints.

The computation of time-optimal trajectories formulated as a problem of computing time-optimal motions has been explored by several works. Reister and Pin (1994) addressed this problem for a non-holonomic robot with bounded acceleration, in an obstacle-free environment. The authors demonstrated that time-optimal motions are obtained from bang–bang controls (maximum acceleration or deceleration), and parametrize the set of time-optimal trajectories using the switching times. Renaud and Fourquet (1997) computed sequences of extremal controls for acceleration-driven robots to reach time-optimal trajectories in free-space. Balkcom and Mason (2002) focused on differential drive and car-like robots with bounded linear and angular velocities but without acceleration limitations. Shiller et al. (2013) presented an on-line motion planner, where robot dynamics and actuator constraints were considered. A near-time-optimal trajectory that avoids stationary obstacles one at a time is incrementally computed, obtaining comparable results with respect to the global optimal solution. Park et al. (2009) dealt with one moving obstacle, and solved an optimization problem with inequality constraints for a holonomic robot with infinite acceleration. Mercy et al. (2016) formulated the optimization problem in a scenario with moving and static obstacles for a holonomic robot with fixed orientation. Real-time optimization was achieved by a B-spline parametrization of the trajectory and exploiting B-splines properties to limit the set of constraints. A different approach to obtain near-time-optimal trajectories was developed in Gal et al. (2009) and Shiller et al. (2010), which adapt time-optimal solutions based on extremal controls for static or free space environments.

In this paper we develop and evaluate a robot motion planner based on a model which represents the dynamism of the environment. This work extends and formalizes the model of a preliminary work (Owen and Montano, 2005). That work presented the dynamic object velocity (*DOV*) to model the moving obstacles in the environment, reasoning on it for robot navigation. The *DOVS* space herein developed explicitly takes into account the future evolution of the obstacles and the dynamics of the robot to compute a set of velocities leading to collision in the future. *DOVS* is defined upon the concepts of *estimated arriving time* of the obstacles, used to compute the times to potential collision or escape, and on the *maximum* and *minimum velocities* for the robot not to collide. As a consequence, a set of safe and feasible robot velocities is obtained for further planning.

The *DOVS* model exhibits several differences with respect to the discussed previous works. *ICS* calculates the states of inevitable collision for a set of escaping manoeuvres, but it does not explicitly compute motion controls for planning motions among the obstacles towards the goals. In addition, the computational cost for several manoeuvres and several objects is high. The *VO* technique computes states for the next sampling period, including velocity, for which there are collisions with the obstacles; but the planning of collision-free trajectories using this model requires looking ahead, simulating several steps forward. Instead, *DOVS* explicitly computes safe linear and angular robot velocities (control variables for the robot) which can be applied without collision in a given but not temporally limited horizon, for instance the field of view of the navigation sensors, without requiring several steps of lookahead simulation. In such a way, building *DOVS* entails an implicit computation of safe velocities (trajectories) without collision within the space horizon established. Moreover, as discussed in Section 5.6, *DOVS* is not affected by some complexity issues that do apply to those previous techniques.

3. Contributions

The *DOVS* model implicitly computes a set of collision-free commands for long-term planning within the sensor's field of view, providing information about collision-free trajectories. A precomputed set of trajectories in the configuration space is represented as sequences of velocity commands in the control space of the robot, which will contain free and forbidden velocities leading to collision, within the space horizon available. Having this greater visibility horizon at the current instant allows the decision process to select the best next motion command from an improved informed search, without needing to simulate the motion several steps ahead. However, only one of these velocity commands is applied for the next sampling period, and a new plan is computed to adapt to new situations in case hypotheses are no longer satisfied.

The contributions are related to two main issues.

- *Environment modeling mapped on the robot velocity space*. The *DOVS* model is formally defined for different kinds of obstacle trajectories and static obstacles, and its properties regarding safety are established. A complexity analysis of the model and a comparison with respect to other well-known techniques in this field is carried out.

- *Planning navigation strategies*. Using the *DOVS* model, a strategies-based approach to compute safe motion commands (velocities) for the robot is developed. Navigation decisions are made based on identifying a *situation* among a discrete set, evaluating the *decision variables* computed from the model of the environment.

The method is assessed by means of simulations and experiments in real-world scenarios. Randomly selected scenarios and dynamic conditions are used for evaluation. The properties of the model are verified, and metrics for performance evaluation of the navigation strategies are provided.

4. Problem statement

The *DOVS* model for dynamic environments exploits the concept of *estimated arriving time* of an object by computing the times of future potential collisions. Then the set of collision-free and collision-causing commands are represented explicitly in the control space of the robot. In such a way, any motion planner based on this model that selects collision-free commands will generate trajectories without collision. To compute the free control space, circular robot trajectories for non-holonomic robots have been selected for the long-term planning. Recomputing the plan every sampling time yields clothoid and anti-clothoid trajectories as the robot moves towards the goal, providing continuous curvature trajectories.

The motion planner applies different strategies of movement depending on the *situation* of the robot with respect to its environment. The situations are identified from the *DOVS* model. The *decision variables* are defined and valued so that a *situation* is identified and thus a navigation command is computed for that situation.

Robot and obstacle states

We consider a non-holonomic robot $\mathcal{R}$ moving in a dynamic environment, where it has to safely reach its goal avoiding collisions with the static and moving objects $\mathcal{O}_i$ around it. They all share workspace $\mathcal{WS} = \mathbb{R}^2$. Let $\mathcal{O}(t)$ be the set of points in $\mathcal{WS}$ occupied by all the obstacles $\mathcal{O}_i$ at instant t, i.e. $\mathcal{O}(t) = \cup \mathcal{O}_i(t)$.

The state of the robot is defined by its location and velocity at instant t, $\mathcal{R}_t = (x\, y\, \theta\, \omega\, v)$. Let $\mathcal{R}(\mathbf{v}, t)$ denote the state reached by the system $\mathcal{R}$ under the action of control $\mathbf{v} = (\omega, v)$ applied at time t. In our case, the controls are the angular (ω) and linear (v) velocity for a differential-drive robot. The motion model for the robot can be expressed by the well-known equations:

$$\dot{x} = v * \cos(\theta)\,; \quad \dot{y} = v * \sin(\theta)\,; \quad \dot{\theta} = \omega \tag{1}$$

From these states and the motion model we describe the *DOVS* environment model in the next section.

5. Modeling the environment

The model is constructed by mapping the information from the workspace to the control space of the robot, denominated *velocity space* ($\mathcal{V}$), which is the set of velocities (ω, v) reachable by the robot limited by the maximum and minimum velocity constraints. This space also contains sets of velocities leading to future collision if the robot executes them. In some way these collision velocities represent obstacles mapped into $\mathcal{V}$, so we name them *DOV*. The space is then used to compute commands without collision for the robot.

The basic concept to build the model is the *time of collision* between robot and obstacle trajectories. The intersection points are computed from a robocentric representation, resulting in associated information in $\mathcal{V}$ of time and velocity that indicate the instants at which an obstacle reaches them, and the velocities for the robot to pass before colliding or after the object passes, avoiding collision. These time–velocity data are then transferred to the control space of the robot to represent the *DOV* obstacles in the environment, conforming the *velocity–time space* ($\mathcal{VT}$) of the robot. In this work, for the sake of clarity, the method is presented for linear and circular obstacle motion, but other kinds of obstacle trajectories could be used, without loss of generality.

One of the advantages of this technique is that the model maps the safe robot velocities for a space horizon only limited by the field of view of the onboard sensors. This allows motions to be planned at every instant for the whole time horizon corresponding to the space horizon and not only for the next sampling period in a purely reactive way. Hence, this approach does not need to simulate future trajectories (states) to obtain a long-term plan, as most of the techniques previously referenced do. In addition, owing to computational burden or to practical navigation issues, this space horizon could be artificially limited, enabling reasoning about the behavior of only the closest obstacles, which are those that will impose the immediate manoeuvring decisions.

5.1. The DOVS

In this section the *DOVS* is defined and its properties described. Appendix B contains the equations for computing the model for linear and circular obstacle trajectories together with a more detailed explanation of the computation process. Next, a brief description is provided.

The model results from computing the times of collision between different feasible robot trajectories and obstacle trajectories in a robocentric reference. Figure 1(a)–(c) demonstrate this idea. In Figure 1a the workspace at a specific instant is depicted. The information in the workspace is mapped to a robocentric representation (Figure 1(b)) to $\mathcal{V}$ (Figure 2(b)). The robot is reduced to a point and the obstacles are enlarged with the radius of the robot. They are modeled as wrapping squares. It is also assumed that the future trajectory of each obstacle $\mathcal{O}_i$ is known or estimated.

In Figure 1(b) the $\mathcal{CB}_i$ (*collision band*) swept by an enlarged obstacle $\mathcal{O}_i$ moving in a straight line is depicted. A circular trajectory γ_j for the robot is also shown. Let Γ be a discretized set of feasible circular trajectories covering the whole range in $\mathcal{WS}$ (see Figure 1(c)). Then $\gamma_j \in \Gamma$.

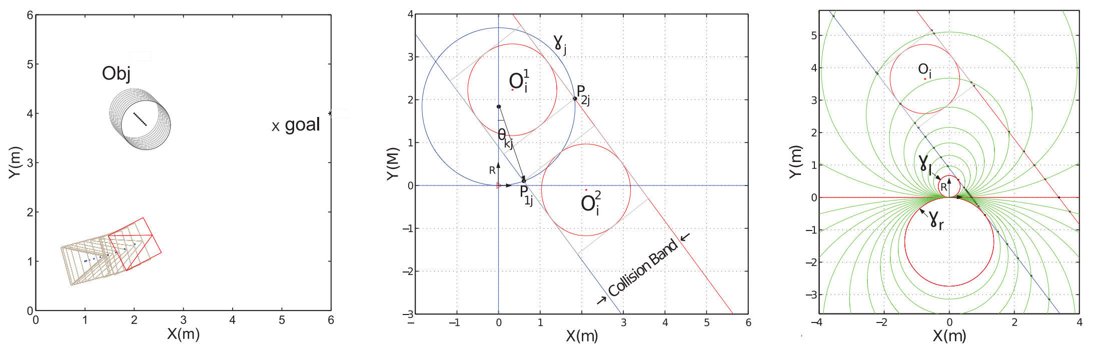
Let P_{1j} and P_{2j} be the intersection points between γ_j and the outline of $\mathcal{CB}_i$. These points represent the collision points in $\mathcal{WS}$ between $\mathcal{R}$ and $\mathcal{O}_i$, and will determine the opportunities for the robot to pass before (P_{2j}) or behind the obstacle (P_{1j}) by following trajectory γ_j. Then, the times at which the obstacle reaches those points, $\mathcal{O}_i^1$ and $\mathcal{O}_i^2$, respectively, are computed. These times are the *estimated arriving times* of the object, which indicate *times of collision*. Thus, the minimum and maximum velocities that allow the robot to pass before or after the obstacle, respectively, are

$$\begin{aligned} \omega_{kj} &= \frac{\theta_{kj}}{t_{kj}} = \frac{\operatorname{atan2}\left(2\, x_{kj}\, y_{kj},\; x_{kj}^2 - y_{kj}^2\right)}{t_{kj}} \\ v_{kj} &= r_j\, \omega_{kj}, \quad k = 1, 2 \end{aligned} \tag{2}$$

where θ_{kj} is the angular displacement on γ_j for the robot to reach $P_{kj} = (x_{kj}, y_{kj})$, $k = 1..2$. See Appendix B for details.

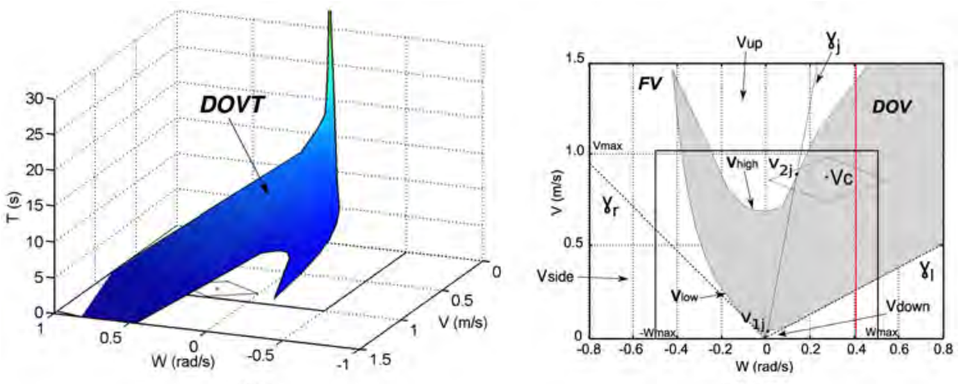
These calculations are extended to the whole set Γ (Figure 1(c)). The computed velocities and times to collision for Γ are mapped in the *velocity–time space*. Figure 2(a) shows the $\mathcal{VT}$ space of the robot including the resultant velocities and times of collision. These sets correspond to the limits of *dynamic object velocity–time* (*DOVT*), and the corresponding *DOV* in $\mathcal{V}$ (Figure 2(b)), which will be defined formally later. The velocities in between are forbidden because they lead to a collision with the obstacle at some time, so any velocity outside *DOVT* can be chosen as a safe command. In Figure 3, the robocentric representation of an obstacle following a circular trajectory and its corresponding *DOV* are plotted.

As a first approach, to simplify the model management, we propose to use the projection of the 3D space $\mathcal{VT}$ into $\mathcal{V}$, represented in Figures 2(b) and 3(b). Note that the circular paths are transformed in this plane into straight lines, with slope $\gamma_j = v_{m_j}/w_{m_j}$, $\gamma_j \in \Gamma$, as shown in Figure 2(b). In this figure, the extreme velocities correspond to $\mathbf{v}_{1j} = (\omega_{1j}, v_{1j})$, maximum robot velocity leading to collision, and $\mathbf{v}_{2j} = (\omega_{2j}, v_{2j})$, minimum velocity to escape from collision, for the path γ_j.



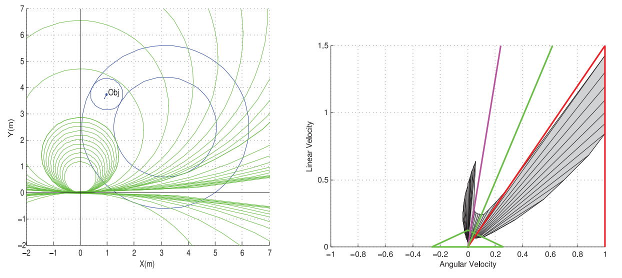
(a) (b) (c)

Fig. 1. (a) Workspace; (b) collision band swept by object $\mathcal{O}_i$, object $\mathcal{O}_i$ in positions $\mathcal{O}_i^1$ and $\mathcal{O}_i^2$, path γ_j, and collision points P_{1j} and P_{2j} in the robocentric (R) configuration space; (c) multiple paths Γ through the collision band.



(a) (b)

Fig. 2. (a) Velocity–time space *DOVTS* and velocity–time obstacle *DOVT*; (b) projection of *DOVTS* on the plane (w, v), *DOVS*, and *DOV* (projection of *DOVT*).



(a) (b)

Fig. 3. (a) Robocentric representation of an obstacle circular trajectory (blue) and a set of circular trajectories for the robot (green). (b) The corresponding model in *DOVS*.

We now formally define the space for the robot and its characteristic variables.

Definition 1 (**Dynamic object velocity–time**). *The dynamic object velocity–time (DOVT) for a particular moving object $\mathcal{O}_i$ with respect to the set of feasible trajectories of the robot Γ is defined as the set of velocities that produce a collision with $\mathcal{O}_i$ at time t,*

$$DOVT(O_i, \Gamma) = \{(\omega, v, t) \in \mathcal{V} \times \mathbb{R} \mid \exists \gamma_j \in \Gamma, \\ \frac{v}{\omega} = \gamma_j,\ \mathbf{v}_j = (\omega, v),\ \mathcal{R}(\mathbf{v}_j, t) \cap \mathcal{O}_i(t) \neq \emptyset\} \quad (3)$$

Definition 2 (**Dynamic object velocity**). *The dynamic object velocity (DOV) for a particular moving object $\mathcal{O}_i$ with respect to the set of feasible trajectories Γ is defined as the projection of its DOVT, i.e. the set of velocities that produce a collision with $\mathcal{O}_i$ at any time,*

$$DOV(O_i, \Gamma) = \{(\omega, v) \in \mathcal{V} \mid \exists \gamma_j \in \Gamma,\ \frac{v}{\omega} = \gamma_j, \\ \exists t \in \mathbb{R},\ \mathbf{v}_j = (\omega, v),\ \mathcal{R}(\mathbf{v}_j, t) \cap \mathcal{O}_i(t) \neq \emptyset\} \quad (4)$$

Accordingly, we define $DOVT(\mathcal{O}, \Gamma)$ and $DOV(\mathcal{O}, \Gamma)$ (henceforth $DOVT$, DOV) with respect to all the obstacles in the environment,

$$DOVT(\mathcal{O}, \Gamma) = \cup_{\mathcal{O}_i} DOVT(\mathcal{O}_i, \Gamma) \quad (5)$$

$$DOV(\mathcal{O}, \Gamma) = \cup_{\mathcal{O}_i} DOV(\mathcal{O}_i, \Gamma) \quad (6)$$

Definition 3 (**Free velocity**). *The free velocity (FV) is the set of velocities outside DOV,*

$$FV = \{(\omega, v) \in \mathcal{V} \mid \boldsymbol{v}_j = (\omega, v) \notin DOV, \ \forall t \in \mathbb{R}, \ \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) = \emptyset\} \quad (7)$$

Velocities belonging to *DOV* are unsafe, leading to collision at some future time. Any velocity inside *FV* will be safe in the long term, given that the initial conditions remain.

Definition 4 (**Dynamic object velocity space**). *The dynamic object velocity space (DOVS) is defined as the control space of the robot which contains controls belonging to DOV and FV,*

$$DOVS = \{DOV \cup FV\} \quad (8)$$

Given that the navigation method presented in this paper focuses on projection *DOVS*, the definitions which follow are provided only with respect to *DOVS*. However, they are straightforwardly applicable to *DOVTS*. Now, we introduce characteristic sets from *DOVS*.

Definition 5. *Let $\boldsymbol{V}_{low}$ and $\boldsymbol{V}_{high}$ respectively be the set of maximum and minimum velocities computed for all paths in Γ (Figure 2b),*

$$\boldsymbol{V}_{low}(O_i, \Gamma) = \{(\omega, v) \in DOV(O_i, \Gamma) \mid \boldsymbol{v}_j = (\omega, v) = \boldsymbol{v}_{1j}, \ t = \boldsymbol{t}_{1j}, \ \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}_i(t) \neq \emptyset\} \quad (9)$$

$$\boldsymbol{V}_{high}(O_i, \Gamma) = \{(\omega, v) \in DOV(O_i, \Gamma) \mid \boldsymbol{v}_j = (\omega, v) = \boldsymbol{v}_{2j}, \ t = \boldsymbol{t}_{2j}, \ \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}_i(t) \neq \emptyset\} \quad (10)$$

Note that $\mathbf{V}_{low}$ and $\mathbf{V}_{high}$ represent the contour of the *DOV* for a particular object, i.e.

$$\begin{aligned}(\omega, v) \in \mathbf{V}_{low}(\mathcal{O}_i, \Gamma) &\leq (\omega, v) \in DOV(O_i, \Gamma) \\ &\leq (\omega, v) \in \mathbf{V}_{high}(\mathcal{O}_i, \Gamma)\end{aligned} \quad (11)$$

Definition 6. *Let $\boldsymbol{V}_{down}$, $\boldsymbol{V}_{up}$ and $\boldsymbol{V}_{side}$ be the sets of velocities in FV eligible for the robot to move safely. In Figure 2(b), they represent the velocity commands under $\boldsymbol{V}_{low}$, above $\boldsymbol{V}_{high}$, and below the bounds of DOV, respectively. Let γ_l and γ_r be the left most and right most radius shaping DOV in* DOVS*:*

$$\boldsymbol{V}_{down} = \{(\omega, v) \in FV \mid \forall \gamma_j \in \Gamma, \ \frac{v}{\omega} = \gamma_j, \ \boldsymbol{v}_j = (\omega, v) < \boldsymbol{v}_{1j}, \ \forall t \in \mathbb{R}, \ \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) = \emptyset\} \quad (12)$$

$$\boldsymbol{V}_{up} = \{(\omega, v) \in FV \mid \forall \gamma_j \in \Gamma, \ \frac{v}{\omega} = \gamma_j, \ \boldsymbol{v}_j = (\omega, v) > \boldsymbol{v}_{2j}, \ \forall t \in \mathbb{R}, \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) = \emptyset\} \quad (13)$$

$$\boldsymbol{V}_{side} = \{(\omega, v) \in FV \mid \forall \gamma_j \in \Gamma, \ \frac{v}{\omega} = \gamma_j \notin [\gamma_l, \gamma_r], \ \boldsymbol{v}_j = (\omega, v), \ \forall t \in \mathbb{R}, \ \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) = \emptyset\} \quad (14)$$

From these definitions we can obtain some important properties that must be used by any planner based on this model to select safe robot commands ensuring collision avoidance in a dynamic environment.

Property 1. *Selecting velocity commands in $\mathbf{V}_{down}$ yields no collision trajectories. Formally,*

$$\forall \boldsymbol{v}_j \in \boldsymbol{V}_{down} \Rightarrow \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) = \emptyset, \forall t \in \mathbb{R}$$

Proof. By the definition of $\mathbf{V}_{down}$, if $\mathbf{v}_j \in \mathbf{V}_{down}$, then $\mathbf{v}_j < \mathbf{v}_{1j}$, where $\gamma_j = \frac{v_j}{w_j} = \frac{v_{1j}}{w_{1j}}$. Then, $\exists t \in \mathbb{R}$ such that $\mathcal{R}(\mathbf{v}_j, t) = P_{1j}$, the intersection point with the collision band. Given that $\mathbf{v}_j < \mathbf{v}_{1j}$, then $t > t_{1j}$. At time t the object has passed P_{1j}, and a collision cannot be produced on γ_j. □

Property 2. *Selecting velocity commands in $\mathbf{V}_{up}$ yields no collision trajectories. Formally,*

$$\forall \boldsymbol{v}_j \in \boldsymbol{V}_{up} \Rightarrow \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) = \emptyset, \forall t \in \mathbb{R}$$

Proof. By the definition of $\mathbf{V}_{up}$, if $\mathbf{v}_j \in \mathbf{V}_{up}$, then $\mathbf{v}_j > \mathbf{v}_{2j}$, where $\gamma_j = \frac{v_j}{w_j} = \frac{v_{2j}}{w_{2j}}$. Then, $\exists t \in \mathbb{R}$ such that $\mathcal{R}(\mathbf{v}_j, t) = P_{2j}$, the intersection point with the collision band. Given that $\mathbf{v}_j > \mathbf{v}_{2j}$, then $t < t_{2j}$. At any time before t_{2j} the object has not yet arrived at P_{2j}, and a collision cannot be produced on γ_j. □

Property 3. *Selecting velocity commands in $\mathbf{V}_{side}$ yields no collision trajectories. Formally,*

$$\forall \boldsymbol{v}_j \in \boldsymbol{V}_{side} \Rightarrow \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) = \emptyset, \forall t \in \mathbb{R}$$

Proof. If $\mathbf{v}_j \in \mathbf{V}_{side}$, then $\frac{v_j}{w_j} = \gamma_j \notin [\gamma_l, \gamma_r]$, by the definition of $\mathbf{V}_{side}$. Thus, no collision could be produced as the resultant trajectory γ_j does not intersect the bounds of the $\mathcal{CB}_i$ at any time. □

Property 4. *Selecting velocity commands in DOV yields collision trajectories. Formally,*

$$\forall \boldsymbol{v}_j \in DOV \Rightarrow \exists t \in \mathbb{R}, \ \mathcal{R}(\boldsymbol{v}_j, t) \cap \mathcal{O}(t) \neq \emptyset$$

Proof. If $\mathbf{v}_j \in DOV$, then $\mathbf{v}_j > \mathbf{v}_{1j}$ and $\mathbf{v}_j < \mathbf{v}_{2j}$, by the definition of *DOV*. Then, $\exists t_1, t_2 \in \mathbb{R}$ such that $\mathcal{R}(\mathbf{v}_j, t_1) = P_{1j}$ and $\mathcal{R}(\mathbf{v}_j, t_2) = P_{2j}$. Given that $\mathbf{v}_j > \mathbf{v}_{1j}$, then $t_1 < t_{1j}$, which means that at time t_1 the object is still at P_{1j} and a collision is produced. Likewise, given that $\mathbf{v}_j < \mathbf{v}_{2j}$, then $t_2 > t_{2j}$, which means that at time t_2 the object has already reached P_{2j} and a collision is produced. Then, a collision could be produced sometime within the interval $t \in [t_{2j}, t_{1j}]$. □

Summarizing the properties, when a velocity command is chosen out of *DOV*, collision avoidance is guaranteed as long as the command is maintained and the object motion hypotheses are met. Thus, commands can be safely changed every sampling control period, as long as they remain outside *DOV*. In the event that a sudden unforeseen change

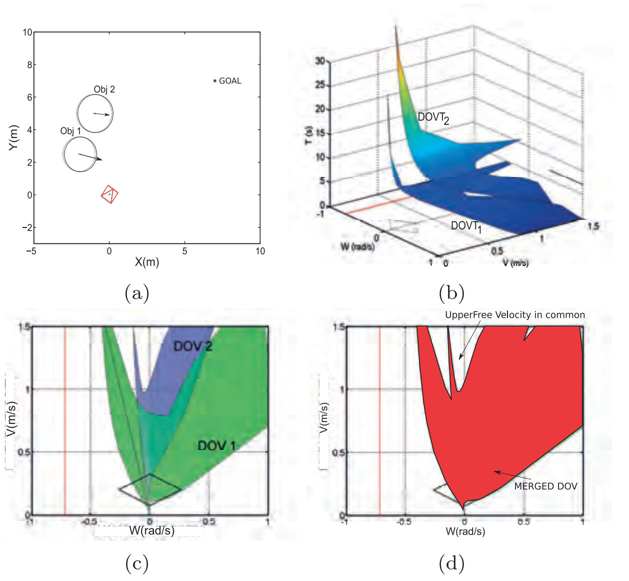


Fig. 4. (a) A situation with two moving objects near to each other in $\mathcal{WS}$. (b) The situation represented in *DOVTS*. (c) Projection of both moving objects in *DOVS*. (d) The merged *DOV* object.

of an object motion results in the current velocity being inside a *DOV*, it still could be possible, time permitting, to escape collision by correctly choosing subsequent velocity commands. This is illustrated in examples in Section 6.

5.2. Dealing with multiple objects in DOVS

Figure 4 shows a situation with two objects in $\mathcal{WS}$, *DOVTS* and *DOVS*. In the *DOVTS* space, moving objects are represented as their corresponding *DOVT* surfaces. Clearly, the highest surface in Figure 4(b) corresponds to the farthest or the slowest object with respect to the robot, and the lowest surface to the nearest or the quickest obstacle. Working directly in *DOVTS* would allow to utilize the velocity–time room between both surfaces for manoeuvring among the objects. Figure 4(c) shows the projection of both surfaces in *DOVS*. When this space is used to plan trajectories, the *DOV* objects can be geometrically merged to obtain one compound *DOV* object (Figure 4(d)). Reasoning in this space to compute the motion commands in the free velocity space (*FV*) is now made using the merged object, directly exploiting the same properties introduced above. This leads to more conservative navigation decisions than if the whole *DOVTS* was used, which is one of the objectives for future work.

5.3. Static objects in DOVS

Static obstacles can also be represented in *DOVS*. Let D_{safe} be the displacement the robot requires on a path to reach zero speed from its current velocity. If the distance to a static object is greater than D_{safe}, then the circular path in

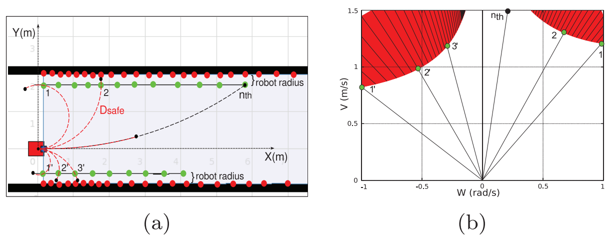


Fig. 5. (a) Mapping a corridor into *DOVS*. The red points correspond to scan laser points for example and the green points are the points after extending the walls with the robot radius. (b) The corridor mapped into *DOVS* showing the maximum velocity reachable for the robot on each circular path considered.

DOVS is mapped as free of collision. Otherwise, the maximum velocity that the robot should have to stop before a collision, $\mathbf{v}_{stop}$, is calculated and mapped into *DOVS*. Any higher velocity is mapped as a collision one. Figure 5 depicts the result of mapping a static corridor-like object into *DOVS*.

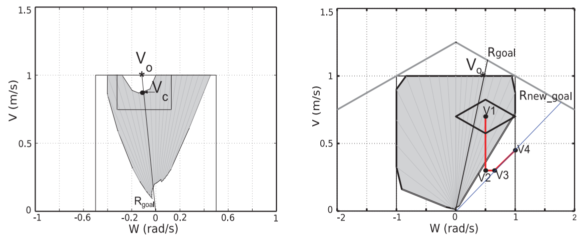
5.4. The kinodynamic constraints in the model

The maximum linear and angular velocity constraints are explicitly represented in *DOVS* by the external square (omnidirectional robot) or the external rhombus (differential drive robot) in Figure 6. So far, it has been assumed that any $\mathbf{v}_o$ could be reachable in one sampling step from the current velocity, which depends on the robot's maximum acceleration. The dynamic constraints of the robot define the velocity window (*VW*) centered in its current velocity $\mathbf{v}_c$ and bounded by the acceleration constraints (a_m, α_m). That is, $(v_c \pm a_m\Delta t, w_c \pm \alpha_m\Delta t)$, the inner rectangle and rhombus in Figure 6(a) and (b), respectively. The shape of this window also represents the kinematic constraints corresponding to the type of the robot model.

Figure 6(a) shows that $\mathbf{v}_o$ can only be reached in one step from the velocities inside *VW*, due to the dynamic constraints. In Figure 6(b) it can be seen that $\mathbf{v}_o$ cannot be reached in one step from $\mathbf{v}_1$. To deal with this problem, we compute first the number of sampling periods needed to reach the limits $\mathbf{V}_{low}$ and $\mathbf{V}_{high}$ from the current velocity in *DOV*, adding them to $\mathbf{t}_{low}$ and $\mathbf{t}_{high}$ to obtain $\mathbf{t}'_{low}$ and $\mathbf{t}'_{high}$, respectively. These times are used to recompute the new velocity limits ($\mathbf{V}'_{low}$, $\mathbf{V}'_{high}$) using Equations (2).

5.5. Trajectories in DOVS

Different kinds of trajectories can be selected to execute the motion plan and manoeuvre the robot. In this work we consider bounded velocity and acceleration for real robots, and apply extremal actions (maximum acceleration or deceleration) motivated by results presented in Reister and Pin (1994), introduced in Section 2. As described in previous work by Owen and Montano (2005), this type of control



(a) (b)

Fig. 6. Kinodynamic constraints in *DOVS*: (a) holonomic robot, $\mathbf{v}_o$ is reachable only inside *VW*, a square window; (b) non-holonomic robot, $\mathbf{v}_o$ is not reachable within *VW*, a rhombus window. Circular, clothoid, and anti-clothoid trajectories in *DOVS* are shown in red. Here R_{new_goal} represents a new radius, computed in the free zone, to be followed until the true R_{goal} becomes free.

action yields rectilinear, clothoid (only angular acceleration), and anti-clothoid (only linear acceleration) paths, allowing a curvature continuity. Moreover, as shown in Section 5.1, circular and linear trajectories are also used for motion planning and execution.

These mentioned trajectories are directly mapped on *DOVS* (see Figure 6b). Linear trajectories in the workspace appear as a straight line in the v axis ($\omega = 0$). Circular paths are mapped as straight lines with a slope, which represent different radii ($\gamma = v/\omega$) (v_3–v_4 red line). Clothoids are depicted as straight horizontal lines ($v = constant$) (v_2–v_3 red line), and anti-clothoids as vertical lines ($\omega = constant$) (v_1–v_2 red line).

5.6. *Complexity of modeling and planning algorithms*

A comparison is performed to analyze the complexity of the algorithms presented for modeling the dynamic environment and its use for planning in other techniques proposed in the literature, specifically those based on *VO* and *ICS*, with respect to our proposal.

Computing the set of *ICS* of a robot for different control trajectories and several obstacles involves a complexity of $\mathcal{O}(mnt_{max}o)$, where m is the number of obstacles, n the number of evasive manoeuvres, t_{max} a lookahead for practical use of the approach and o the sum of vertices of the robot and the most complex object, which are involved in the Minkowski difference operation required between each robot–obstacle pair. Parameter t_{max} is the number of sampling periods, with a duration of Δt each, considered for computing the *ICS* set, and thus the total time of such states being *ICS*. It is assumed that the workspace is bounded, and the robot and obstacles will exit it at t_{max}. By using graphics processing unit (GPU) the complexity can be reduced to $\mathcal{O}(mnt_{max})$ (see Martinez-Gomez, 2010).

The *VO* approach concerns mainly three operations: computing each individual VO_j for $j = 1..m$ obstacles, the multiple velocity obstacle *MVO*, and the set of safe velocities. The most complex operation is the second one, because it requires updating the points of each VO_i, $i = j + 1..m$, with the intersection points between VO_j and each VO_i, and ordering them clockwise, which leads to a complexity of $\mathcal{O}(m^2)$. This can be reduced to $\mathcal{O}(m \log m)$ by storing the points within a more efficient structure (see Fiorini, 1995). However, this approach provides information only for computing the next control command as constrained to the acceleration window. If a further plan has to be obtained, a tree search would be performed expanding the nodes for the lookahead considered t_{max}, which has the same meaning as in *ICS*. Then, the complexity would be $\mathcal{O}(q^{t_{max}/\Delta t} m \log m)$, where q is the fixed number of velocities that can be reached within the next sampling period, and $t_{max}/\Delta t$ represents the number of levels achieved during tree expansion.

In our technique, the complexity of modeling each moving object in its DOV_j, $j = 1..m$, is $\mathcal{O}(mn)$, where m is the number of obstacles and n the maximum number of the trajectories considered (Γ). The set of trajectories selected for computing a DOV_j depends on the position of the obstacle and its motion with respect to the robot, and on the desired discretization. The calculation for collision times and forbidden velocities is therefore focused on the area where the obstacle will be moving. Thus, several robot trajectories are considered for each obstacle. Furthermore, the complexity for mapping objects moving in linear and non-linear trajectories (in particular, circular paths have been developed in this work) is the same, because only intersection points with the collision band are needed, not altering the computation burden. The approach presented in this paper deals with multiple objects by merging all DOV_j. This operation requires for an object to compute the collision times and velocities for the robot trajectories considered for the other objects, and to repeat this for each object. The complexity is then $\mathcal{O}(nm^2)$. This approach provides enough information for computing plans in the defined workspace horizon, so the complexity is not affected by the parameter t_{max} of the other methods, since a lookahead simulation is not needed for planning safe motions. In Section 6.2 the criteria to select near time-optimal actions are explained.

6. Decision making for planning and navigation

This section explains the decision-making process for a robot when dealing with moving obstacles. The problem is similar to a situation in which a pedestrian intends to cross a road while cars are passing. The pedestrian must decide how to do this safely. In such a situation, the main decision the robot must take is whether to pass before or after the obstacle.

Most of the proposed techniques for navigation in dynamic environments limit the number of possible robot trajectories or manoeuvres, or simply slow down or stop the robot to avoid a collision by allowing the moving obstacle to

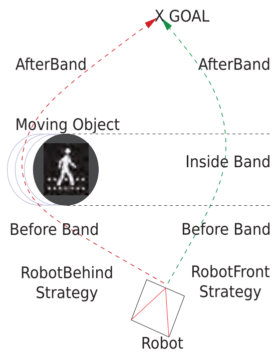


Fig. 7. Trajectory in green when the robot passes before the moving obstacle and in red when the robot passes after the moving obstacle.

pass first. The method presented in this work enables lighter computation of long-term safe command sequences in the free-velocity space, and thus the corresponding trajectories to the goal. These planned trajectories are not time-optimal, but are used as a first seed to guide the robot towards the goal in the free space. In the following steps, the recomputed extremal control commands will lead the robot to move in near time-optimal trajectories to the goal.

The design is based on the paradigm of situated activity (Arkin, 1998). The robot decides the motion plan within a set of finite but general situations identified from the perception system. Each situation has an associated motion strategy, which exploits the information provided by the model of the environment. Two main criteria are considered to define the strategy: (i) the trajectories generated have to be safe; (ii) near-time-optimal trajectories to the goal have to be generated. The first is ensured by selecting velocity commands inside the free velocity *FV* of the control space, as explained in the previous section. The second criterion is accomplished by the strategies defined for each situation, computing velocity commands complying with the robot kinodynamic constraints. Each motion leads to a stretch of trajectory, which combined during navigation drive the robot to the goal maintaining the highest velocities possible.

Figure 7 reflects this idea. Essentially, there are two main planning decisions: passing before the object (*RobotFront*) and passing after the object (*RobotBehind*). In turn, each is decomposed into different sub-decisions depending on whether the robot is before, inside or after the collision band. The execution of each motion strategy depends, in turn, on several *decision variables*, which drive the hierarchical decision process implemented in a decision tree. These variables are formally defined in the next subsection.

6.1. Decision variables

Table 1 summarizes the *decision variables*, some in the workspace $\mathcal{WS}$ and others in *DOVS*, including a brief explanation about their meaning and their use in the different situations. An important decision variable is *GD* (goal direction), which maps a circular trajectory from the current robot location towards the goal. If this trajectory is followed, reaching the goal is ensured. Commands always have to be selected in *FV* (free velocity) space. Then, when *GD* lies inside a *DOV*, it has to be moved to *FV* (GD_{new} in Figure 8(a)). To that end, a nearby velocity sub-goal that avoids the obstacles, not only in the next sampling period but also within the visibility horizon, is chosen. Figure 8(b) shows a situation in which there are not *UpperFree* velocities, so only $\mathbf{V}_{down}$ and $\mathbf{V}_{side}$ velocities can be selected.

6.2. Optimization based on decision variables

The decision-making process described above is implemented by means of navigation strategies, which use the defined decision variables. The objective of the strategies is to navigate towards the goals in dynamic scenarios, executing near-time-optimal trajectories whilst ensuring safety (no collisions). Unlike in static environments, in dynamic environments where obstacle motion is uncertain or it might unexpectedly change, computing global time-optimal trajectories is not feasible.

To apply the decision-making strategies based on *DOVS* model, the direction to the goal is projected in *DOVS* as the decision variable *GD*, representing a steering direction that drives directly to the goal, planning initially a circular trajectory γ_{GD} from the current robot location. This trajectory is not time-optimal, so it will only serve as an initial reference to move towards the goal. Two kinds of extremal controls are chosen to be applied every sampling time to reach *GD*, either maximum angular acceleration to optimally align first the robot to goal position, or maximum linear acceleration for optimally approaching the goal. This kind of trajectories were derived in Reister and Pin (1994) as time-optimal ones for differential-drive robots with acceleration constraints in free space. We have adapted that technique for dynamic environments, obtaining near-time-optimal trajectories. They lead to clothoid and anti-clothoid trajectories, respectively (see Figure 9(a) and (b)). Moreover, we have to take into account the kinodynamic constraints of the vehicle, which limit the acceleration and the velocity to be applied during every control sampling time.

The time-to-goal for a clothoid, $t_{G_w}(\mathbf{v}_c, \theta_m, d_G)$, and for an anti-clothoid, $t_{G_v}(\mathbf{v}_c, \theta_m, d_G)$, depends on several *decision variables*. Since these trajectories are computed from *Fresnel* functions, which analytic solution cannot be obtained for all the decision variables, we have analyzed by simulation both time functions (t_{G_w}, t_{G_v}) for different ω_0 initial conditions. In Appendix C, these functions are described. Figure 9 represents them. The conclusion of this study is that for an ample range of ω_0 and for near and far goals the clothoids result in lower times, mainly for low values of ω_0. Thus, it can be deduced that it is always better initially to increase the angular velocity up to the maximum if possible, rather than keeping the initial angular velocity, until an angular deceleration is needed for aligning. When

Table 1. Decision variables in WS and *DOVS*.

Variable	Meaning	Situation
WS		
RelPos	relative position before, in, or after the collision band	all
AngDis (θ_m)	angular distance robot–goal	all
DOVS		
FV - *DOV*	free (*FV*) and non-free (*DOV*) velocities	all
$\mathbf{V}_{high}$	minimum velocities to pass before object	PassingBefore, SlowingDown
$\mathbf{V}_{low}$	maximum velocities to give way to object	PassingBefore, SlowingDown
$\mathbf{V}_{up}$ - $\mathbf{V}_{down}$	high ($\mathbf{v} > \mathbf{V}_{high}$) - low ($\mathbf{v} < \mathbf{V}_{low}$) free velocities	all
$\mathbf{V}_{side}$	free velocities $\mathbf{v} = \{(\omega, v) \mid \frac{v}{w} = \gamma_j \notin [\gamma_l, \gamma_r]\}$	all
$\mathbf{v}_{valley}$	$(\omega, v)_{valley} = \min(\mathbf{V}_{high})$	PassingBefore
UpperFree	free angular velocities with maximum linear velocity	PassingBefore, PassingAligned
SafeVel	safe low velocities to avoid or escape from collision	SlowingDown
BoundRight–*BoundLeft*	γ_r–γ_l mapped in velocity space, boundaries for $\mathbf{V}_{side}$	AvoidingObject
v_m, ω_m, a_m, α_m	maximum linear and angular velocities and accelerations	all
VW (velocity window)	velocities that can be reached within next time interval	all
GoalDirection (*GD*)	$(\omega, v)_{GD}$ mapped direction in velocity space	all
SteeringDir (*SD*)	ω_{SD} mapped angular deviation to goal in velocity space	all

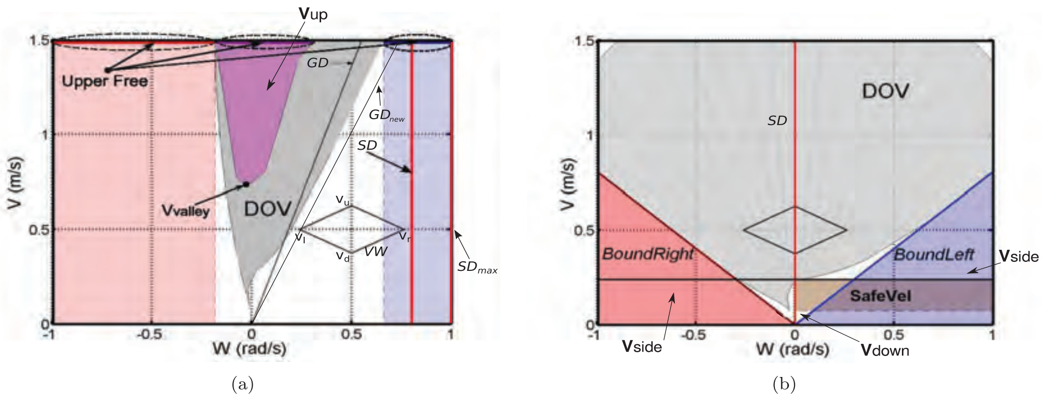


Fig. 8. Decision variables in *DOVS*. (a) Situation in which there are *UpperFree* velocities (magenta, blue, and pink); given that *GD* lies inside the *DOV*, the closest *UpperFree* to *GD* will be chosen, computing GD_{new}. (b) Situation in which *UpperFree* is occupied with velocities leading to collision (gray), only low velocities in $\mathbf{V}_{down}$ and $\mathbf{V}_{side}$ can be selected.

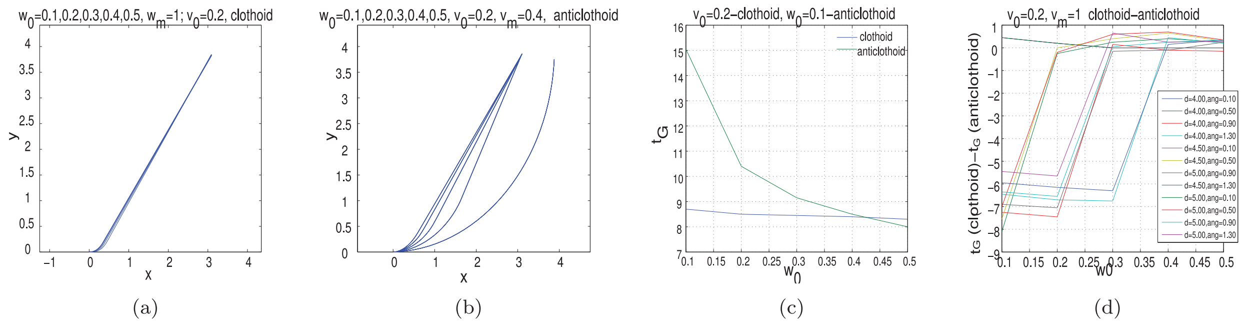


Fig. 9. (a) Clothoid and (b) anti-clothoid trajectories for different initial values of ω_0. The higher ω_0, the quicker the aligning to the goal. (c) Times to goal t_G for trajectories in (a) and (b). (d) Time to goal difference t_G(clothoid) $-t_G$(anticlothoid) for different goal distances (d) and angular deviations (ang) with high and low maximum linear velocities.

Algorithm 1 Align the robot to the goal

```
Require:
1: $\mathbf{v}_c$, the current velocity of the robot
2: $(\omega, v)_{GD}$, maximum velocity to goal direction in DOVS
3: $\theta_m$, angular deviation to goal from current robot position
4: $d_G$, distance to goal from current robot position
5: function ALIGNING($\mathbf{v}_c$, $(\omega, v)_{GD}$, $\theta_m$, $d_G$)
6:     if $|\omega_c| \neq |\omega_{GD}|$ then
7:                                        ▷ robot not aligned (clothoid)
8:         $\omega_i = \operatorname{argmin}_{\mathbf{v}\in VW}(t_{G_w}(\mathbf{v}_c, \theta_m, d_G)) =$
9:             $= \operatorname{argmin}_{\mathbf{v}\in VW}(w_m - \omega_{GD})$
10:        $v_i = v_c$
11:    else                               ▷ robot is aligning (anti-clothoid)
12:        $\omega_i = \omega_c$
13:        $v_i = \operatorname{argmin}_{\mathbf{v}\in VW}(t_{G_v}(\mathbf{v}_c, \theta_m, d_G)) =$
14:            $= \operatorname{argmin}_{\mathbf{v}\in VW}(v_c - v_{GD})$
15:    end if
16:    return $(\omega_i, v_i)$
17: end function
```

the robot is nearly aligned to the goal, a linear acceleration is applied to speed up to maximum velocity. Algorithm 1 implements this method, and is used in the strategies to avoid moving obstacles.

6.3. Situation-based navigation

The planner reasons in the *DOVS* space described in Section 5, which maps the environment dynamism of obstacles. In a scenario with obstacles, the trajectories introduced before do not lead to time-optimal trajectories because they may result in collisions. However, it is possible to find near-time-optimal and safe trajectories computed in the free velocity space FV, easily identifiable in *DOVS*. Thus, the methodology presented in Section 6.2 is applicable in this space, avoiding obstacles whilst preserving low time-to-goal motions and safety.

The strategies are associated to *situations* detected. Figure 10 shows the situation tree, where the leaves correspond to all the situations that can be identified. Each of the *situations* has an associated *action*; the sequence of actions applied to reach the goal constitutes a strategy, which implements the general decision making process described in Section 6. The robocentric planner executes cyclically, performing the strategies and so adapting to the changing environment in real-time. The planner establishes a long-term plan within the horizon defined by the field of view of the sensors and not only for the next sampling period. In this sense, this local planner is not purely reactive. It computes a safe trajectory for that space horizon. The plan is re-computed for each sampling period, so only the immediate action of the strategy required by the current situation is applied at each sampling time.

Algorithm 2 Basic FreeMotion for the robot

```
Require:
1: $\mathbf{v}_c$, the current velocity of the robot
2: $(\omega, v)_{GD}$, maximum velocity to goal direction in DOVS
3: $\theta_m$, angular deviation to goal from current robot position
4: $d_G$, distance to goal from current robot position
5: function FREEMOTION($\mathbf{v}_c$, $(\omega, v)_{GD}$, $\theta_m$, $d_G$)
6:     $\mathbf{v}_1 = \mathbf{v}_c$
7:     $\mathbf{v}_2 = Aligning(\mathbf{v}_c, (\omega, v)_{GD}, \theta_m, d_G)$
8:     $\mathbf{v}_3 = (\omega_i = 0,\ v_i = v_{i-1} + a_m\,\Delta t)$
9:                              ▷ $a_m$, maximum linear acceleration
10:    $\mathbf{v}_4 = (\omega_i = 0,\ v_i = v_m)$        ▷ $v_m$, maximum linear velocity
11:    return $(\mathbf{v}_1 - \mathbf{v}_2 - \mathbf{v}_3 - \mathbf{v}_4)$
12: end function
```

Table 2 shows the actions associated with each situation. Roughly speaking, the method consists in applying a sequence of maximum angular and linear accelerations to reach the maximum linear velocity in the free velocity space (FV), following the command computation developed in Section 6.2. Before doing it, if the goal direction mapped in the velocity space, *GD*, is inside a dynamic obstacle *DOV*, a close new goal GD_{new} is mapped in FV in each situation. This way the near-time-optimal command sequence is applied to reach that goal, avoiding the moving obstacle until *GD* is in FV again.

Algorithm 2 represents the basic motion to be applied in all situations, as shown in Table 2. The only difference between situations is the election of a new GD_{new} in the free-velocity space FV. Figures 11–15 describe situations *ObstacleFree*, *PassingBefore*, *PassingAligned*, *AvoidingObject*, and *SlowingDown*, its associated actions and the strategies derived from them. The *AvoidingCollision* situation is a consequence of the robot having entered the collision band of an obstacle and then having to execute a velocity in free space to avoid collision. The *CertainCollision* situation appears when the robot falls into an unavoidable collision situation, due either to an unexpected obstacle appearing on the scene, or because no free velocity is available (a blocking situation, e.g. objects surrounding the robot).

7. Experimental evaluation and metrics

7.1. Simulation results

In this section we evaluate the proposed technique qualitatively and quantitatively. Simulations where the robot traverses the scenario to reach different goals are shown in Extension 1.[1]

In these scenarios only moving objects appear around the robot. We have considered obstacle occupancies of 4% and 7% (Figure 16), given that the 4% value is a commonly found scenario in the literature (Bareiss and Van

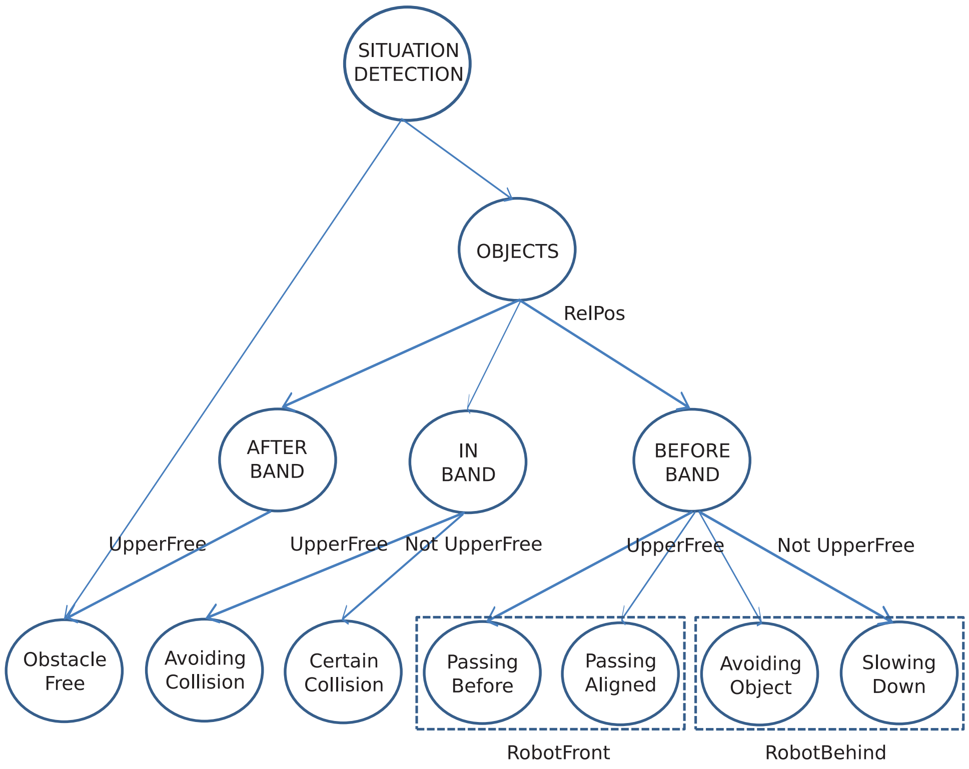


Fig. 10. Situation tree. Leaves represent all the situations that can be identified, each of which has an associated navigation strategy.

Table 2. Actions for the situations. Here $\mathbf{V}_{sol}$ represents the sequence of velocities to be applied in a situation, $\mathbf{v}_c = (\omega, v)_c$ is the current velocity of the robot, $(\omega, v)_{valley} = \mathbf{v}_{valley}$, $\mathbf{v}_{GD} = (\omega, v)_{GD}$ is the maximum velocity in *DOVS* for the goal direction (*GD*), ω_{GD} is the angular component of *GD* bounded by maximum angular velocity mapped into *DOVS*, $v_{SafeVel}(\mathbf{v}_1)$ is the nearest linear free velocity with respect to $\mathbf{v}_1$, and $VW = \mathbf{v} \in FV \cap \{\mathbf{v}_l, \mathbf{v}_u, \mathbf{v}_r, \mathbf{v}_d\}$, where $\mathbf{v}_l = (\omega_c - \alpha_m \Delta t, v_c)$, $\mathbf{v}_r = (\omega_c + \alpha_m \Delta t, v_c)$, $\mathbf{v}_u = (\omega_c, v_c + a_m \Delta t)$, and $\mathbf{v}_d = (\omega_c, v_c - a_m \Delta t)$. Here GD_{new} is the new goal direction in free velocity (*FV*) closest to the true *GD*.

Situation	Actions (velocities)
ObstacleFree (Figure 11)	$\mathbf{V}_{sol} = \mathbf{FreeMotion}(\mathbf{v}_c, GD, \theta_m, d_G)$
RobotFront Strategy	
PassingBefore (Figure 12)	$\mathbf{V}_{sol} = \mathbf{FreeMotion}(\mathbf{v}_c, GD_{new}, \theta_m, d_G) \mid \mathbf{v}_{GD_{new}} = (\omega_{valley}, v_m)$
PassingAligned (Figure 13)	$\mathbf{V}_{sol} = \mathbf{FreeMotion}(\mathbf{v}_c, GD_{new}, \theta_m, d_G) \mid \mathbf{v}_{GD_{new}} = (\omega_{GD_{new}}, v_m) \in FV,$ $\omega_{GD_{new}} = \min_{\mathbf{v} \in UpperFree}(\lvert\omega\rvert)$
RobotBehind Strategy	
AvoidingObject (Figure 14)	$\mathbf{V}_{sol} = \mathbf{FreeMotion}(\mathbf{v}_c, GD_{new}, \theta_m, d_G) \mid \mathbf{v}_{GD_{new}} = (\omega_{GD_{new}}, v_m) \in FV,$ $\omega_{GD_{new}} = \min_{\mathbf{v} \in UpperFree}(\lvert\omega - \omega_{GD}\rvert)$
SlowingDown (Figure 15)	**If** $(\omega, v)_c \in DOV$ **then** $\mathbf{v}_1 = \mathbf{v}_c$, $\mathbf{v}_2 = (\omega_2 = \omega_1, v_2 = \text{argmin}_{\omega=\omega_2}(v_c - v_{SafeVel}(\mathbf{v}_1)))$ **Else** $\mathbf{V}_{sol} = \mathbf{FreeMotion}(\mathbf{v}_c, GD_{new}, \theta_m, d_G) \mid \mathbf{v}_{GD_{new}} = (\omega, v)_{GD_{new}} \in SafeVel$ **EndIf**

den Berg, 2015; Martinez-Gomez and Fraichard, 2009). Although these occupancies are seemingly not very dense when compared to static benchmark scenarios, the large impact of moving obstacles in the effective space for safe planning (see Figure 16) makes them suitably challenging. This effective space has been computed as the mean of the forbidden velocity areas in *DOVS*, a way to quantify the limited robot manoeuvrability. See Table 3.

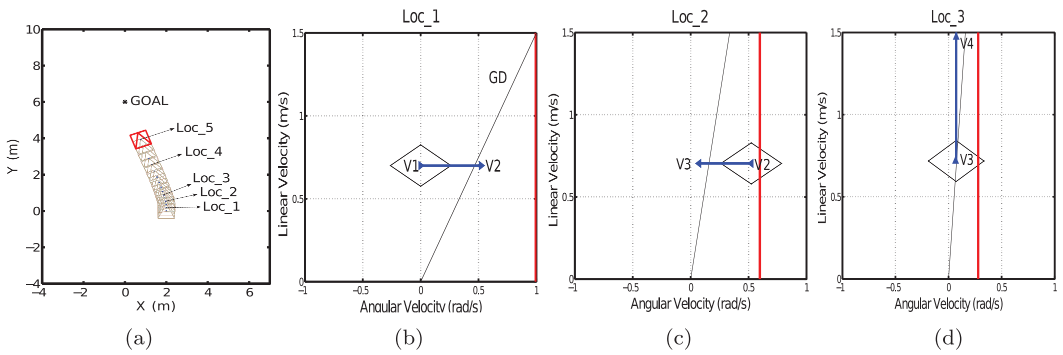


Fig. 11. Evolution of the ***ObstacleFree*** situation. Algorithm 2 implements the motion for this situation. (a) Part of the trajectory followed by the robot. (b)–(d) Velocity space of the robot and the applied actions in three representative locations of the trajectory. The vertical red line represents the steering direction *SD* and the black line the goal direction *GD*, to be reached for alignment to the goal. In Loc_1 a maximum angular acceleration ($\mathbf{v}_1$–$\mathbf{v}_2$) is applied to reach *SD* every sampling period (the angular deviation of the robot is high). In Loc_2 a maximum angular deceleration ($\mathbf{v}_2$–$\mathbf{v}_3$) is applied to reach *GD* (the angular deviation has decreased). In Loc_3 the robot applies commands at maximum linear acceleration ($\mathbf{v}_3$–$\mathbf{v}_4$), while in Loc_4 and Loc_5, a straight line at maximum linear velocity $\mathbf{v}_4$ is achieved.

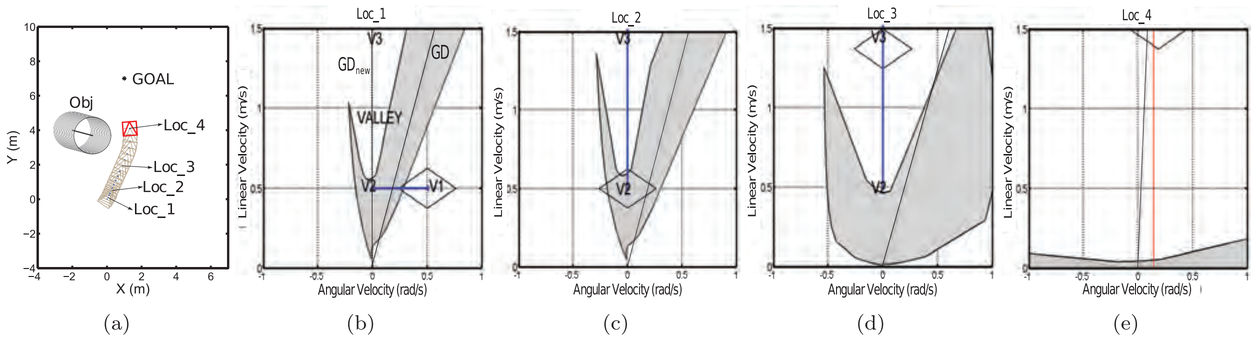


Fig. 12. Evolution of the ***PassingBefore*** situation. (a) Trajectory of the robot. (b)–(e) *DOVS* at four relevant instants during the trajectory. In this situation the robot can cross the collision band to pass before the object. There is a set of free velocities in the *UpperFree* zone which can be reached inside the bounds of the *DOV* ($\mathbf{V}_{up}$), identified as a *valley*. The depth of the valley ($\mathbf{v}_{valley}$ in Table 1) provides certain knowledge about the level of safety for the robot to perform the strategy. The deeper the valley, the greater its width, and the safer it is to traverse the *DOV* to reach the free velocities in $\mathbf{V}_{up}$. A new GD_{new} is computed close to the initial *GD* in $\mathbf{V}_{up}$, such that it contains velocity $\mathbf{v}_{valley}$. A sequence of clothoid ($\mathbf{v}_1$–$\mathbf{v}_2$) at maximum angular deceleration (b), anti-clothoid ($\mathbf{v}_2$–$\mathbf{v}_3$) at maximum linear acceleration (c), (d), and straight line at maximum linear velocity is applied (e).

We have tested navigation for three different types of scenario during simulation: *linear*, *non-linear*, both for moving obstacles, and *complete*, having static and moving obstacles. A total of 40 different scenarios are simulated in *linear* and *non-linear* sets. In each scenario the robot has to reach four different goals. In total, we have run 1440 simulations, 160 for each % of obstacle occupancy (4% and 7% for *linear*, 4% for *non-linear*), for the three sets of velocities V_1, V_2, V_3 described in Table 3, and for a maximum linear robot velocity of 1.5 m s^{-1}.

We have evaluated the technique in scenarios where the density of obstacles remains the same, so static bounds were established to define the workspace and simulate the movement of the obstacles cyclically, once the random movement has been generated. Table 4 reflects the collisions produced during simulation, as explained later. Column *Bounded (V_1)* indicates the percentage of collisions in scenarios with static bounds. A first observation is that the number of collisions is greater than in open scenarios, represented in the other three columns, due to the robot being trapped when it is close to them. Therefore, to focus on the evaluation of the robot navigating among moving obstacles, we have discarded the static bounds to evaluate the performance based on metrics. Another reason for collisions is because new obstacles appear from outside the limits of the scenario, so the robot might not have the reaction capability from kinodynamic constraints to avoid collision. In addition, we have evaluated the method when conditions about obstacle motion is uncertain, scenario *NL-Approx*. Thus, we have performed simulations in scenarios with obstacles moving with non-linear trajectories and approximating the *DOVS* model with the linear approach. The collisions greatly increase, as can be seen in Table 4. Therefore, the method benefits from the greater lookahead if there is a

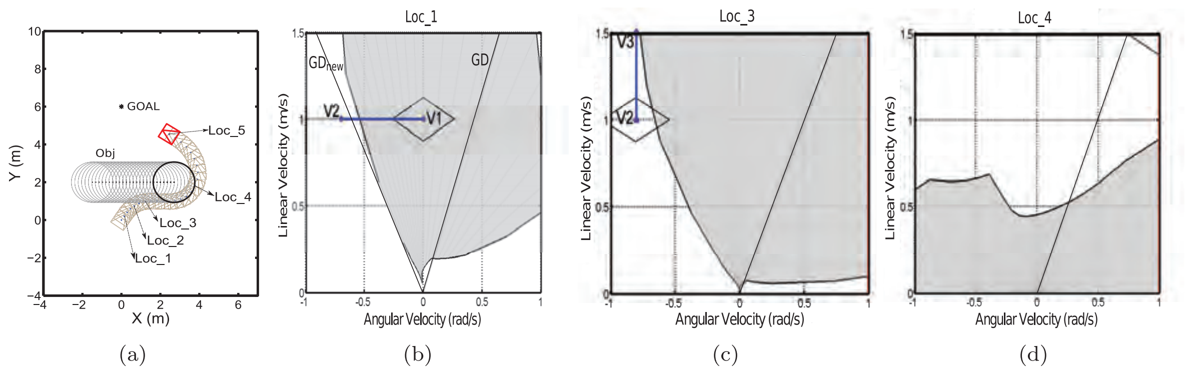


Fig. 13. Evolution in the ***PassingAligned*** situation. (a) Trajectory performed by the robot. (b)–(d) *DOVS* for different motions of the robot. The set of velocities in *UpperFree* leads the robot to move in the same direction as the object. In this case, the robot lies inside the *DOV* and *GD* is not free. Here GD_{new} is defined to escape from dangerous velocities, applying an angular acceleration motion (clothoid $\mathbf{v}_1$–$\mathbf{v}_2$). Then, maximum linear acceleration ($\mathbf{v}_2$–$\mathbf{v}_3$) is applied to move in the same direction as the obstacle (Loc_3). If the robot moves faster than the object, a valley appears inside *DOV*, which allows the robot to change the strategy to a *PassingBefore* situation. Finally, *AvoidingCollision* (Loc_4) and *ObstacleFree* (Loc_5) situations are identified when the robot lies inside the collision band and when it exits it, respectively.

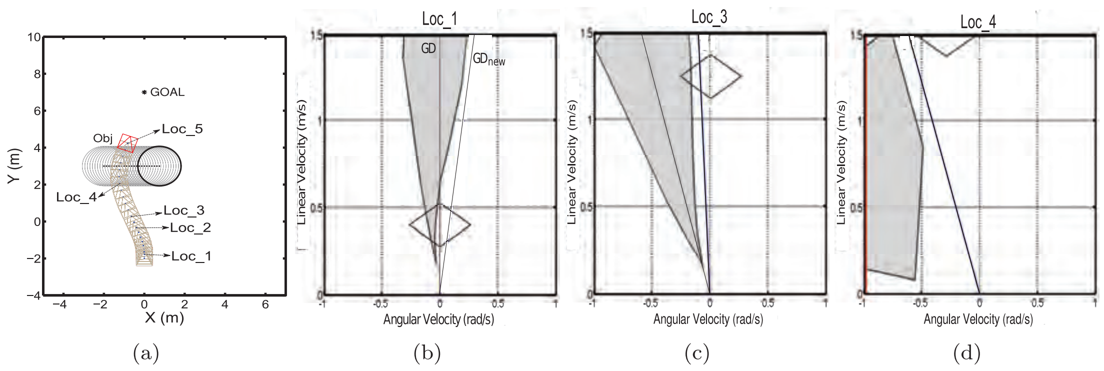


Fig. 14. Evolution of the ***AvoidingObject*** situation. (a) Partial trajectory of the robot. (b)–(d) *DOVS* representation at different instants. In this situation, the *DOV* of a moving object occupies the middle zone of *DOVS*, leaving two zones of velocities in *UpperFree* that belong to $\mathbf{V}_{side}$ (b). A new GD_{new} close to the current *GD* is selected in $\mathbf{V}_{side}$, to then apply the sequence of motions computed by Algorithm 2. The robot manoeuvres to pass behind the object.

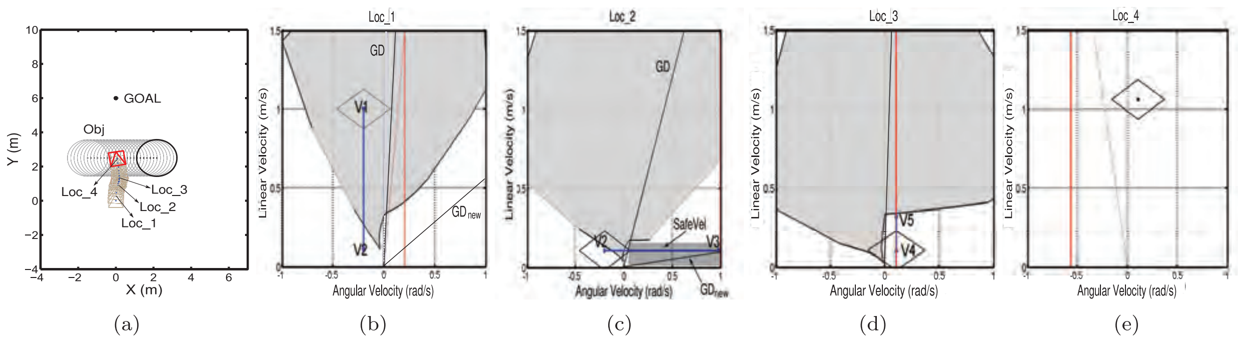
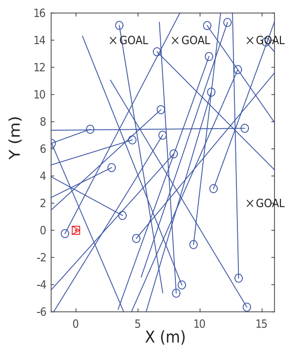


Fig. 15. Evolution in the ***SlowingDown*** situation. (a) Trajectory of the robot. (b)–(e) *DOVS* information at different locations of the robot. There are no *UpperFree* velocities in *DOVS* (see Figure 15(a) and (b)). In this case, the velocity of the robot is inside the *DOV* of the object, and the robot has to slow down to avoid a collision. A sequence of anti-clothoid trajectories (deceleration) is applied to escape from the dangerous velocities ($\mathbf{v}_1$–$\mathbf{v}_2$). Then, a sequence of clothoid trajectories ($\mathbf{v}_2$–$\mathbf{v}_3$–v_4) is computed to maintain the robot at a low safe velocity in the *SafeVel* zone until the object passes (c), following a new GD_{new} computed in *SafeVel*. The set *SafeVel* defines the velocities which will be freed first, given the motion of the object. Finally an *ObstacleFree* situation appears (e), and the corresponding strategy is applied.


(a)

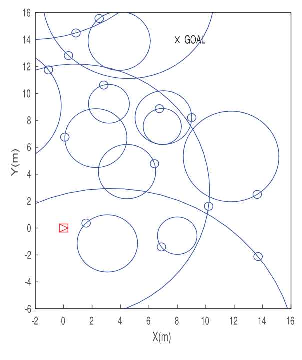

(b)

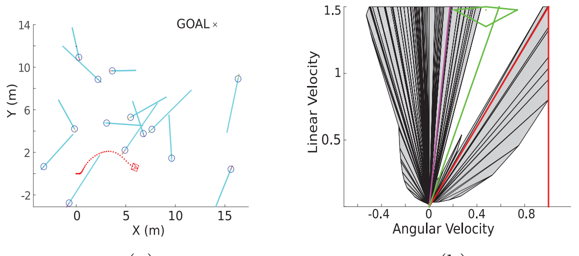
Fig. 16. Scenarios with different densities of occupancy: (a) 7% of static density corresponds to 46% of occupancy in terms of safe velocity space available to be chosen in the *linear* scenario; (b) 4% of static occupancy represents about 35% of forbidden velocities in the *non-linear* scenario with obstacles moving in circular trajectories.

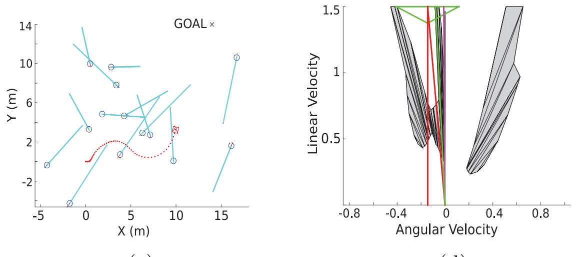
Table 3. Mean of forbidden velocity area in *DOVS* during simulations for different obstacle velocity ranges. The values of 4% and 7% for obstacle occupancy were measured as the area of the workspace occupied by the obstacles at a given instant (as if the obstacles were static). However, if we measure the occupancy in terms of forbidden velocities in *DOVS* as the area of each *DOV*, these values increase considerably.

	V_1 $v = 0.2$	V_2 $v = 0.5$	V_3 $v = 0.6..0.9$
Linear		$\omega = 0$	
4% occupancy	34.85%	40.12%	42.67%
7% occupancy	46.08%	53.98%	58.19%
Non-linear	$\omega = -0.15..0.15$	$\omega = -0.38..0.38$	$\omega = -0.45..0.45$
4% occupancy	34.97%	39.17%	40.06%

Table 4. Percentage of collisions during simulations. Note that in the highest-occupancy situations and with the scenario bounds acting as static obstacles, many trapping situations can appear so some inevitable collisions occur.

	Bounded (V_1)	V_1	V_2	V_3
Linear				
4% occupancy	5%	0%	2.5%	3.75%
7% occupancy	32.5%	5%	10%	16.875%
Non-linear				
4% occupancy	10.6%	3.75%	15%	11.875%
NL-Approx				
4% occupancy		10.62%	34.37%	46.25%


(a) (b)


(c) (d)

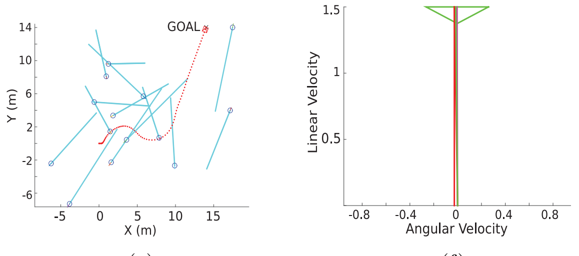

(e) (f)

Fig. 17. Simulation in a *linear* scenario for 4% obstacle occupancy and velocities in V_1. In fact, this percentage increases in a dynamic scenario, as a result of the reduction on the available workspace and velocity space. The workspace (left) and *DOVS* (right) are represented at different instants. The planner searches for free velocity sub-spaces so that the robot (red) can maintain high linear velocities (rhomboid window) during navigation.

proper estimation of the obstacle motion. Whenever the motion changes, the calculations should be recomputed and, in this case, our method behaves like a reactive one.

Figure 17 illustrates several iterations of a simulation of the *linear* scenarios. The robot manoeuvres around the obstacles while maintaining safety, selecting high velocities in the free velocity sub-space *FV*.

Figures 18 and 19 show several metrics evaluated during navigation for the overall simulations in *linear* scenarios, for each of the occupancy densities considered and for each set of velocities. As expected, in a scenario where fewer obstacles appear, the robot can move at higher velocities (Figures 18(a) and 19(a)). Safety is evaluated from the distance measured between the robot and each of the obstacles (Figures 18(b) and 19(b)). Figures 18(c) and 19(c) illustrate the time-to-goal with respect to the optimal value for each goal, computed for free space. Longer trajectories are produced when there are more obstacles in the environment.

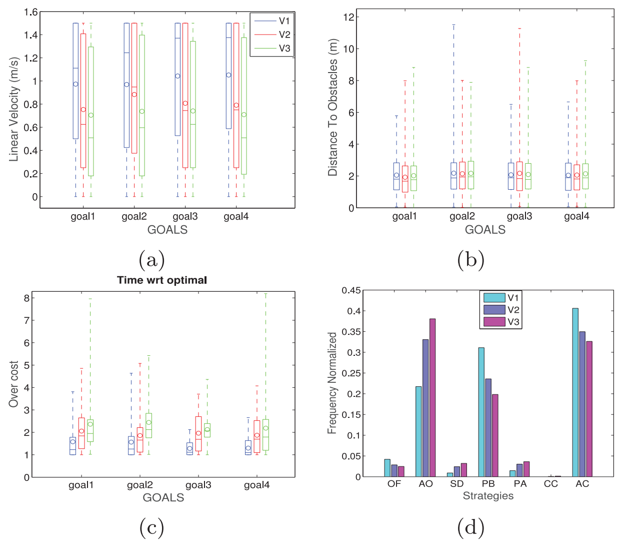


Fig. 18. Metrics for 4% occupancy scenarios with obstacles moving linearly. Percentile, mean, and median for: (a) linear velocity, (b) distance to obstacles, (c) time with respect to optimal for free space, and (d) histogram of the situations.

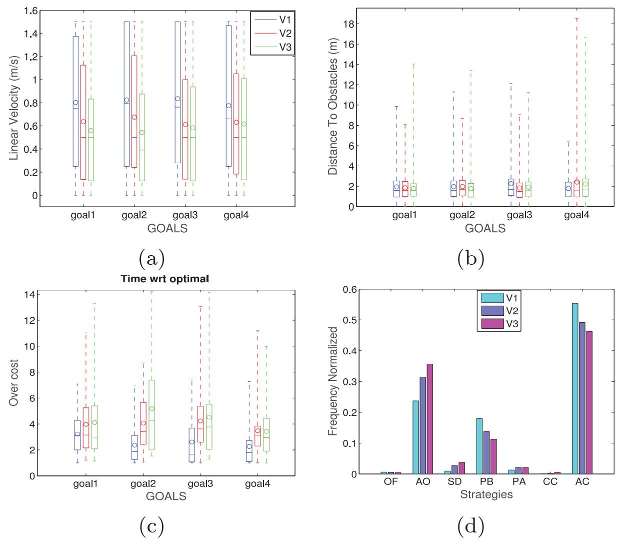


Fig. 19. Metrics for 7% occupancy scenarios, with obstacles moving linearly.

Figures 18(d) and 19(d) describe the strategies selected during navigation. *AvoidingObject* and *AvoidingCollision* are the most frequently applied strategies. *PassingBefore* strategy occurs more frequently than *SlowingDown*, which means that the robot tries to search for free spaces to pass before objects at high velocity, rather than slowing down to give way to objects. It can be seen that for 7% occupancy *ObstacleFree* and *PassingBefore* appear less frequently than for 4% occupancy, as was predictable.

Regarding simulations for *non-linear* scenarios, Figure 20 plots the same metrics as for the *linear* scenarios. Similar results are obtained, so the technique adapts well to different kinds of obstacle trajectories.

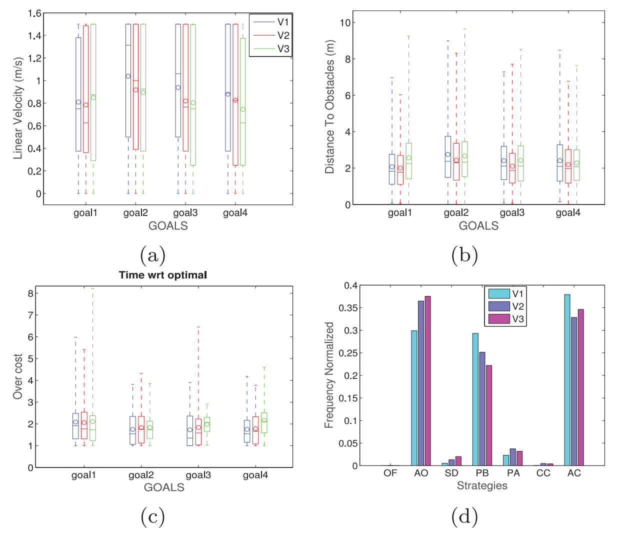


Fig. 20. Metrics for simulations for 4% occupancy of non-linear obstacles.

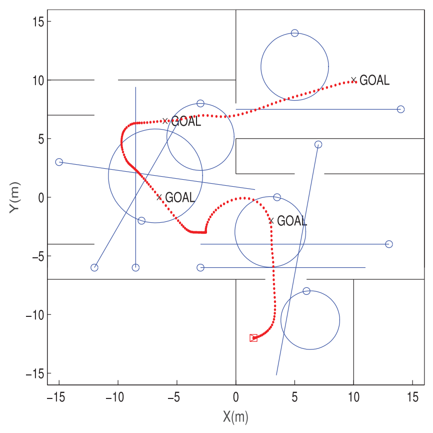


Fig. 21. *Complete* scenario with static (black), linear, and non-linear obstacles describing their trajectory (blue), and the trajectory performed by the robot (red) to achieve the different goals.

Moreover, the technique developed is conservative in the sense that any velocity in free velocity space *FV* is safe, but the velocities inside a *DOV* do not always or instantly lead to an unavoidable collision; they can be selected but only during a certain time without collision. This improvement is the subject of future work.

Finally, in the *complete* scenario shown in Figure 21 there are static and moving obstacles (following linear and non-linear trajectories) to illustrate a general situation in which a robot has to move from one room to another, navigating among several objects and four subgoals (waypoints). Figure 22 represents the metrics. In this scenario, where many static objects are present and appear in *DOVS*, the *ObstacleFree* situation is not encountered, so there are not many velocities free to manoeuver.

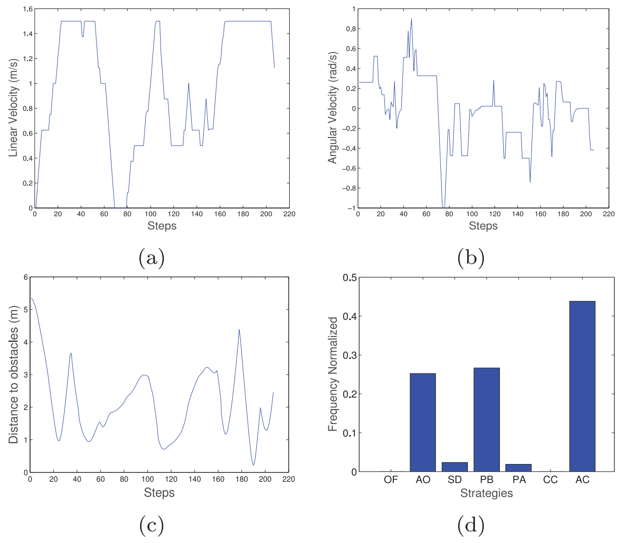


Fig. 22. Metrics for simulation of the *complete* scenario. Most of the time the robot moves at maximum velocity, slowing down when it becomes considerably disorientated from the goal (bigger changes observed for angular velocity). Strategy *PassingBefore* is applied many times, that is the robot moves many times at high velocities to pass before the obstacles, and no *CertainCollision* situation appears.

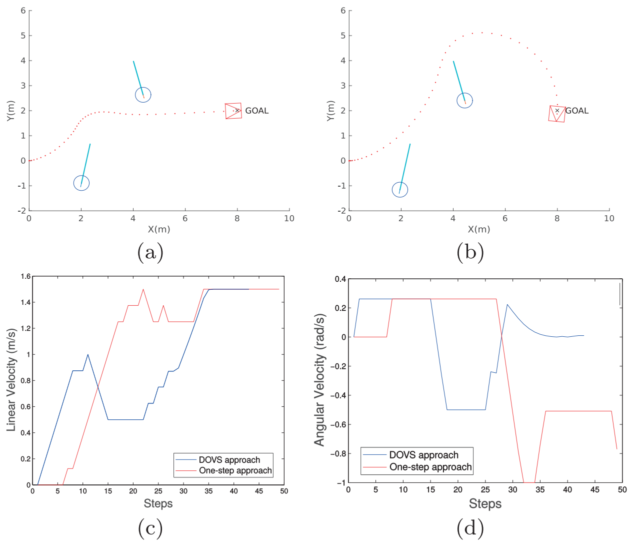


Fig. 24. Trajectories followed by the robot reasoning within *DOVS* visibility (a) and *one-step* visibility (b). Profiles of linear (c) and angular (d) velocities during navigation. Looking further ahead at a single instant (*DOVS*) provides shorter and lower time-to-goal trajectories than using the theoretical time-optimal control to the goal at every sampling time (*one-step* approach).

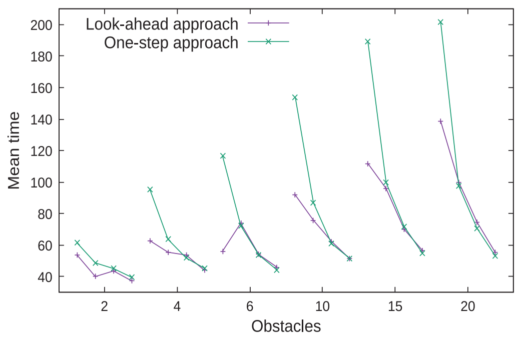


Fig. 23. Comparison of the mean of the time-to-goal for our lookahead planner and the *one-step* local method, for the velocity range considered.

7.2. Comparison with a one-step approach

We show here through simulation how our lookahead planner obtains a better behavior and lower time-to-goal than a method that considers only the next-period candidate velocities. This local planning approach is denominated *one-step* approach, and it is the decision process used in Gal et al. (2009) and Shiller et al. (2010) to develop a near-time-optimal motion planner. The basic idea is that the planner chooses collision-free velocities from the velocity window (*VW*) that generate a time-optimal trajectory to the goal, evaluated in absence of obstacles. In our case, for a non-holonomic robot, the near-time-optimal motions used are those introduced in Section 6.2. The planner proposed in this work computes collision-free commands from the velocity window based on the future evolution of the obstacles, lookahead information previously considered to plan a safe trajectory to the goal, by choosing a *Goal Direction* (*GD*) outside *DOV*.

The evaluation carried out is performed in randomly generated scenarios, where the robot has to reach four different goals, during a set of 10 simulations. The range of velocities considered for the obstacles are: 0.2, 0.5, 0.8, and 1.2. Figure 23 shows the mean values of the time-to-goal for each of the methods. It can be seen that for low obstacle velocities, our approach gets better results, whereas similar values are obtained with higher velocities. This is because the obstacles spend more time occupying the workspace: the *one-step* approach remains in the time-optimal trajectory for the current *VW*, whereas our approach favors the selection of commands towards bigger areas of free space. Therefore, both approaches obtain near-time-optimal trajectories, but ours can achieve lower times to the goal.

Figure 24(a) and (b) plot a scenario for one of the simulations designed, where the robot behavior using both methods to move around the obstacles is also shown. As can be seen, our method makes the robot to pass between both obstacles, whereas the *one-step* approach computes a longer trajectory due to the implicit reactivity of the local planning. As a consequence, the time-to-goal and the trajectory length are reduced (Figure 24(c) and (d)).

Figure 25 describes several steps of the simulation for both methods. *DOVS* lookahead plans long-term manoeuvres in the velocity free space, which permits to find safe commands when re-planning every sampling time to move between both obstacles, unlike *one-step* planner that

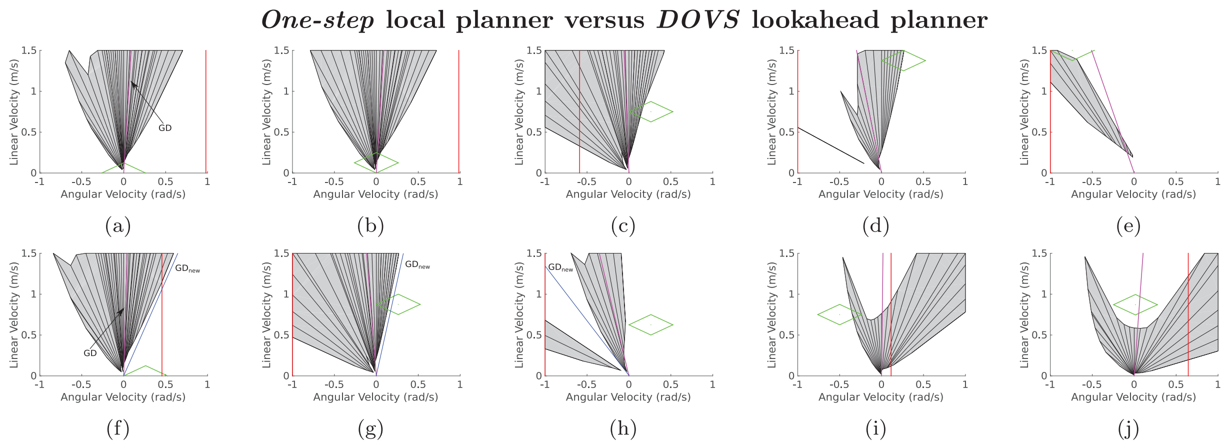


Fig. 25. (a)–(e) *One-step* local planner: the velocity which would drive the robot to the goal applying time-optimal commands from the current position–velocity in free space (mapped *GD*) is selected. In (d) the velocity obstacle prevents the robot from choosing a velocity to manoeuvre towards the goal, until the moving object passes and *GD* becomes free to be followed (e). (f)–(j) *DOVS* lookahead planner: it plans safe velocities out of the velocity obstacle for the sensor visibility horizon, driving the robot to the new mapped goal *GD$_{new}$* (f)–(h) instead of *GD*, leading the robot to slow down and manoeuvre to finally cross between both obstacles through the velocity valley (i), (j).

only re-plans locally to follow a theoretical time-optimal trajectory towards the goal in the free space.

7.3. *Real-world experiments*

The first scenario corresponds to the kind of scenarios shown in the simulations with obstacles moving randomly in all directions. The second is a more structured scenario, an overtaking manoeuvre. Extension 2 contains a video of the experiments.

7.3.1. Experimental setup. The navigation strategies have been tested on three Pioneer robots. These are differential-drive robots equipped with a 2D laser rangefinder Sick LMS-200 and on-board Intel Centrino duo at 1.6 GHz. The computation time of the navigation strategies was around 200 ms, the time for which a new laser measuring is available. The field of view of the laser rangefinder sensor was 180°, 0.5° of angular resolution and a maximum range of 8 m. The maximum translational velocity was set to 0.3 m s^{-1} and the rotational velocity to 0.4 rad s^{-1}, fitting the velocity window used. The method developed by Montesano et al. (2008) is used for mapping static and moving obstacles from rangefinder sensors, and an extended Kalman filter (EKF)-based technique is applied to track the moving objects, in which the state vector included the location and the velocities of the tracked objects.

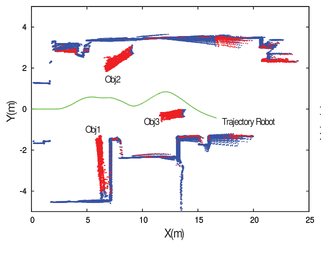
The work presented in Prassler et al. (2001) developed a navigation system on a wheelchair equipped with sensors. The authors represent the space occupied at instant *t* with a *time stamp map*, and compare the corresponding information in the previous step to determine moving and static obstacles by using a nearest-neighbor criterion. The *VO* approach is used to compute safe velocities, selecting the highest feasible velocity in the direction of the goal or a manoeuvre-avoidance velocity with a specific heading. As a result, wall-following and obstacle-avoidance behaviors are observed in simulation and real-world crowded environments.

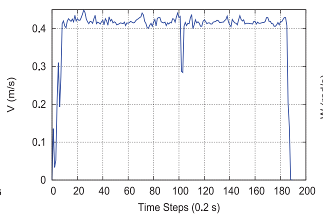
7.3.2. Experiment 1. Figure 26(a) depicts the hall-like scenario, with robot and object trajectories, not previously known. Figure 26(b)–(d) shows the velocity profiles and the strategies. *ObstacleFree*, *AvoidingObject*, and *PassingBefore* are the most frequent situations, allowing to maintain the maximum linear velocity. Figure 27 illustrates several steps of the experiment.

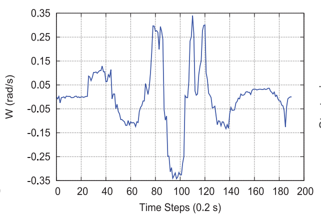
7.3.3. Experiment 2. Figures 28(a) and 29 show an overtaking manoeuvre in a narrow corridor. Before overtaking obstacle 1, the robot slows down until obstacle 2 passes. When the manoeuvre is initiated, the maximum velocity is maintained until the end (Figure 28(b)). *ObstacleFree*, *AvoidingObject*, and *SlowingDown* are the situations that occur the most. The latter makes the robot reduce the velocity before starting the overtaking manoeuvre.

8. Conclusions

A robocentric technique for planning and navigation in dynamic environments has been developed. The *DOVS* model defined for mapping the dynamics of the environment allows the planning of safe motions within a space horizon, for instance the range of visibility of the onboard sensors. Several situations depending on the *decision variables* are identified applying specific actions associated to them, which implement the *RobotFront* or the *RobotBehind* strategies. The complexity of the method does not increase


(a)


(b)


(c)

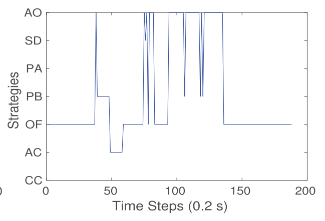
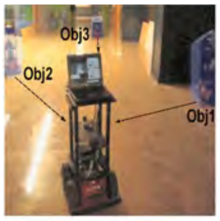
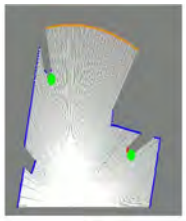
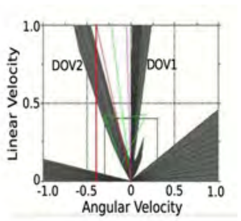

(d)

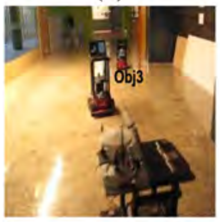
Fig. 26. Hall scenario: (a) trajectory of the robot (green) and environment perceived in experiment 1, with static and moving objects; (b) linear velocity profile; (c) angular velocity profile; (d) situations.



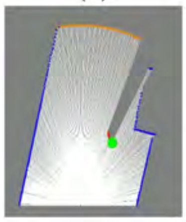
(a)


(b)


(c)



(d)


(e)

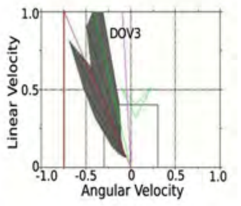
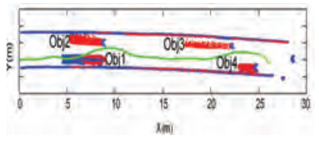
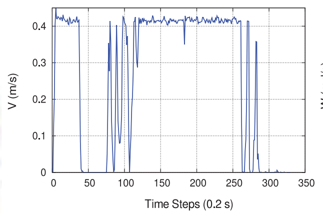
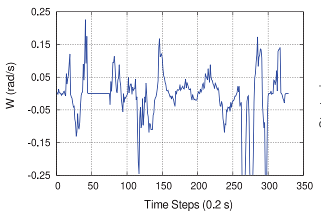

(f)

Fig. 27. Experiment 1 in the *hall* (*random*) scenario. The robot can pass between *Obj*1 and *Obj*2 (a), (b) at maximum velocity (c), and avoids *Obj*3 (d), (e) towards the side closest to the goal (f).


(a)


(b)


(c)

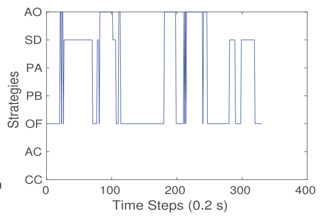

(d)

Fig. 28. Overtaking scenario: (a) trajectory of the robot (green) and environment perceived in experiment 2, (b) linear velocity profile, (c) angular velocity profile, and (d) situations.

either by trajectories that are more complex than linear obstacle trajectories, or as a result of the space–time horizon for planning, unlike other techniques described in the literature. It has been shown that the long-term strategies selected every sampling time within the space horizon visibility improve the time to goal with respect to other techniques that are purely reactive. The method has been evaluated in simulation and in real-world experiments. In a dynamic scenario, the navigation difficulty can be measured from the velocity space area occupied by the obstacles, which somehow reflects the actual capability of the robot to manoeuvre. Sometimes collisions are unavoidable, mainly in cases where the robot is trapped near static obstacles or when obstacles appear in the scenario, due to robot's kinodynamic constraints.

From the lessons learned, some improvements are being considered for future work. Using directly the *DOV* obstacles without merging them would make the processing more complex but enlarge the set of free and safe velocities to be selected to manoeuvre. Blocking situations appearing in closed scenarios or with many static obstacles could be reduced achieving a different treatment for those static

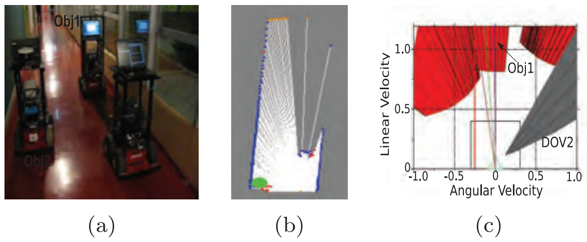


Fig. 29. Experiment 2 in the *Overtaking* scenario. The robot has to slow down before overtaking obstacle *Obj*1. Red velocity obstacles (c) are the walls on both sides of the corridor and obstacle *Obj*1, which is detected as static given that both the robot and *Obj*1 are navigating with a similar velocity.

obstacles, by prioritizing free and safe velocities that move the robot away from them. In the context of a multi-agent scenario in which several mobile robots can make decisions, a future work will be focused on the extension of the model and development of planning techniques to share the decision-making process among several robots to obtain mutually optimized plans.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was partially supported by project DPI2016-76676-R and project DGA T04-FSE.

Notes

1. An extended video for simulations in random-generated scenarios can be downloaded from http://robots.unizar.es/data/videos/simulations_random_scenarios.mp4

References

Althoff D, Althoff M, Wollherr D and Buss M (2010) Probabilistic collision state checker for crowded environments. In: *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1492–1498.

Althoff D, Kuffner JJ, Wollherr D and Buss M (2012) Safety assessment of robot trajectories for navigation in uncertain and dynamic environments. *Autonomous Robots* 32(3): 285–302.

Arkin RC (1998) *An Behavior-based Robotics*. 1st edition. Cambridge, MA: MIT Press.

Balkcom DJ and Mason MT (2002) Time optimal trajectories for bounded velocity differential drive vehicles. *The International Journal of Robotics Research* 21(3): 199–217.

Bareiss D and Van den Berg J (2013) Reciprocal collision avoidance for robots with linear dynamics using LQR-obstacles. In: *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3847–3853.

Bareiss D and Van den Berg J (2015) Generalized reciprocal collision avoidance. *The International Journal of Robotics Research* 34(12): 1501–1514.

Bautin A, Martinez-Gomez L and Fraichard T (2010) Inevitable collision states: A probabilistic perspective. In: *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4022–4027.

Bekris KE and Kavraki LE (2007) Greedy but safe replanning under kinodynamic constraints. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 704–710.

Bekris KE, Tsianos KI and Kavraki LE (2009) Safe and distributed kinodynamic replanning for vehicular networks. *Mobile Networks and Applications* 14(3): 292–308.

Bouraine S, Fraichard T, Azouaoui O and Salhi H (2014) Passively safe partial motion planning for mobile robots with limited field-of-views in unknown dynamic environments. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3576–3582.

Bouraine S, Fraichard T and Salhi H (2012) Provably safe navigation for mobile robots with limited field-of-views in dynamic environments. *Autonomous Robots* 32(3): 267–283.

Brock O and Khatib O (2000) Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths. In: *Proceedings IEEE International Conference on Robotics and Automation, 2000 (ICRA '00)*, vol. 1, pp. 550–555.

Chan N, Kuffner J and Zucker M (2008) Improved motion planning speed and safety using regions of inevitable collision. In: *17th CISM-IFToMM Symposium on Robot Design, Dynamics, and Control (RoManSy'08)*.

Fiorini P (1995) *Robot Motion Planning Among Moving Obstacles*. PhD Thesis (UMI Order No. GAX95-19816), University of California, Los Angeles, CA, USA.

Fiorini P and Shiller Z (1998) Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research* 17(7): 760–772.

Fox D, Burgard W and Thrun S (1997) The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine* 4(1): 23–33.

Fraichard T (1998) *Trajectory Planning in Dynamic Workspace: a 'State-Time Space' Approach*. Technical Report RR-3545, INRIA.

Fraichard T (2007) A short paper about motion safety. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1140–1145.

Fraichard T and Asama H (2003) Inevitable collision states. a step towards safer robots? In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003 (IROS 2003)*, vol. 1, pp. 388–393.

Frazzoli E, Dahleh MA and Feron E (2001) Real-time motion planning for agile autonomous vehicles. In: *Proceedings of the 2001 American Control Conference*, vol. 1, pp. 43–49.

Gal O, Shiller Z and Rimon E (2009) Efficient and safe on-line motion planning in dynamic environments. In: *Proceedings IEEE International Conference on Robotics and Automation*, Kobe, Japan, pp. 188–93.

Hsu D, Kindel R, Latombe JC and Rock S (2002) Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research* 21(3): 233–255.

Kalisiak M and van de Panne M (2007) Faster motion planning using learned local viability models. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 2700–2705.

Kim M and Oh JH (2016) Study on optimal velocity selection using velocity obstacle (OVVO) in dynamic and crowded environment. *Autonomous Robots* 40(8): 1459–1470.

Large F, Laugier C and Shiller Z (2005) Navigation among moving obstacles using the NLVO: Principles and applications to intelligent vehicles. *Autonomous Robots* 19(2): 159–171.

Large F, Sekhavat S, Shiller Z and Laugier C (2002) Using non-linear velocity obstacles to plan motions in dynamic environments. In: *IEEE International Conference on Control, Automation, Robotics and Vision (ICARV02)*, pp. 734–739.

LaValle SM and Kuffner JJ (2001) Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5): 378–400.

Martinez-Gomez L (2010) *Safe Navigation for Autonomous Vehicles in Dynamic Environments: an Inevitable Collision State (ICS) Perspective*. Theses, Université de Grenoble.

Martinez-Gomez L and Fraichard T (2009) Collision avoidance in dynamic environments: An ICS-based solution and its comparative evaluation. In: *IEEE International Conference on Robotics and Automation, 2009 (ICRA '09)*, pp. 100–105.

Mercy T, Loock WV and Pipeleers G (2016) Real-time motion planning in the presence of moving obstacles. In: *2016 European Control Conference (ECC)*, pp. 1586–1591.

Minguez J and Montano L (2005) Sensor-based robot motion generation in unknown, dynamic and troublesome scenarios. *Robotics and Autonomous Systems* 52(4): 290–311.

Montesano L, Minguez J and Montano L (2008) Modeling dynamic scenarios for local sensor-based motion planning. *Autonomous Robots* 25(3): 231–251.

Owen E and Montano L (2005) Motion planning in dynamic environments using the velocity space. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005 (IROS 2005)*, pp. 2833–2838.

Pallottino L, Scordio VG, Bicchi A and Frazzoli E (2007) Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics* 23(6): 1170–1183.

Park J, Choi JS, Kin J and Lee BH (2009) Moving obstacle avoidance for a mobile robot. In: *2009 IEEE International Conference on Control and Automation*, pp. 367–372.

Petti S and Fraichard T (2005) Safe motion planning in dynamic environments. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2210–2215.

Prassler E, Scholz J and Fiorini P (2001) A robotics wheelchair for crowded public environment. *IEEE Robotics Automation Magazine* 8(1): 38–45.

Reister DB and Pin FG (1994) Time-optimal trajectories for mobile robots with two independently driven wheels. *The International Journal of Robotics Research* 13(1): 38–54.

Renaud M and Fourquet JY (1997) Minimum time motion of a mobile robot with two independent, acceleration-driven wheels. In: *Proceedings 1997 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2608–2613.

Seder M and Petrovic I (2007) Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1986–1991.

Shiller Z, Gal O and Fraichard T (2010) The nonlinear velocity obstacle revisited: the optimal time horizon. In: *Guaranteeing Safe Navigation in Dynamic Environments Workshop*, Anchorage, AK.

Shiller Z, Large F and Sekhavat S (2001) Motion planning in dynamic environments: obstacles moving along arbitrary trajectories. In: *Proceedings 2001 IEEE International Conference on Robotics and Automation (ICRA2001)*, vol. 4, pp. 3716–3721.

Shiller Z, Sharma S, Stern I and Stern A (2013) Online obstacle avoidance at high speeds. *The International Journal of Robotics Research* 32(9–10): 1030–1047.

Stachniss C and Burgard W (2002) An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*, vol. 1, pp. 508–513.

Van den Berg J and Overmars M (2008) Planning time-minimal safe paths amidst unpredictably moving obstacles. *The International Journal of Robotics Research* 27(11–12): 1274–1294.

Appendix A: Index to multimedia extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of Multimedia Extension

Extension	Media type	Description
1	Video	Simulations to show the navigation performance of the robot in several scenarios and conditions.
2	Video	Real experiments in two scenarios: a hall-like scenario, and an overtaking-like manoeuvre.

Appendix B: Computation of the robot collision velocities

B.1: Obstacle linear trajectories

Figure 1(b) in Section 5 illustrates a moving object describing a linear trajectory, positioned at locations ($\mathcal{O}_i^1$ and $\mathcal{O}_i^2$). The positions are related to the instants at which the robot reaches points P_{1j} and P_{2j}, following a circular trajectory γ_j. Here P_{1j} is computed as the position at which the robot should arrive after the object has just passed it, at time t_{1j} ($\mathcal{O}_i(t_{1j}) = \mathcal{O}_i^2$) and P_{2j} is the position at which the robot should arrive just before the object reaches it, at time t_{2j} ($\mathcal{O}_i(t_{2j}) = \mathcal{O}_i^1$).

From object location $\mathcal{O}_i(t_0) = (x_o, y_o, \phi_o)$ in the robot local reference and its velocity $\mathbf{v}_o$, the points of collision $P_{1j}(x_{1j}, y_{1j})$ and $P_{2j}(x_{2j}, y_{2j})$, and the corresponding times t_{1j} and t_{2j} are calculated by solving the following equations characterized by the curvature radius r_j of path γ_j and the center $(0, y_j)$ in the robot reference frame R. For a path γ_j, t_{ij} ($i = 1, 2$) is computed from Equations (15), obtaining two solutions for each collision point P_{ij} (Equation (16)).

$$\begin{aligned} x_{ij} &= x_0 + v_0 \cos(\phi_0)\, t_{ij} \\ y_{ij} &= y_0 + v_0 \sin(\phi_0)\, t_{ij} \\ {r_j}^2 &= x_{ij}^2 + (y_{ij} - y_j)^2 \end{aligned} \tag{15}$$

$$t_{ij} = (-B \pm \sqrt{B^2 - 4AC})/2A \tag{16}$$

$$A = v_0^2$$
$$B = 2v_0(x_0\cos(\phi) + y_0\sin(\phi)) - 2v_0\sin(\phi)y_j$$
$$C = x_0^2 + y_0^2 + y_j^2 - 2y_0y_j - r_j^2$$

From xy-coordinates of points P_{1j} and P_{2j}, and times t_{ij}, the velocities $\mathbf{v}_{ij}$ are computed, as in Section 5.1. Two cases can occur. When the robot is out of the collision band, we select t_{ij} as the solution corresponding to the first intersection point (the lower value of t_{ij}) for both P_{1j} and P_{2j}. When the robot is in the collision band, the strategy is to escape from the band to avoid a collision. In this case, only the escape point P_{2j} and velocity $\mathbf{v}_{2j}$ are calculated, using the previous equations. A lower velocity would result in collision.

B.2: Obstacle circular trajectories

In a similar way, the intersection points $P_{1j}(x_{1j}, y_{1j})$ and $P_{2j}(x_{2j}, y_{2j})$ between a circular robot trajectory and a circular collision band swept by the obstacle (delimited by $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$), and the corresponding times t_{1j} and t_{2j}, are computed. These points are computed from intersections between circles C_{in} and C_{out} and the robot circular path γ_j, following Equations (17) (see Figure 30).

Here $\mathcal{O}_i^1$ and $\mathcal{O}_i^2$ are the object positions for computing the collision times t_{2j} and t_{1j}, respectively. Applying tangency conditions between robot trajectory γ_j and the circles corresponding to the object in both positions (Equation (18)), the times to collision are calculated from the angular displacement θ_{cij} and the angular velocity of the object ω_{obj} (Equations (19)). Then, the velocities $\mathbf{v}_{ij}$ are computed as in Section 5.1:

$$\begin{aligned}(x - x_c)^2 + (y - y_c)^2 &= r_{Cout}^2 = (r_c - r_{rob})^2\\ (x - x_c)^2 + (y - y_c)^2 &= r_{Cin}^2 = (r_c + r_{rob})^2 \\ (x - x_{rob})^2 + (y - y_{rob})^2 &= r_{rob}^2\end{aligned} \tag{17}$$

$$(x - x_{obj})^2 + (y - y_{obj})^2 = r_{obj}^2 \tag{18}$$

$$\begin{aligned}\theta_{cij} &= atan2(2\,x_{obj_i}\,y_{obj_i}, x_{obj_i}^2 - y_{obj_i}^2) \\ t_{ij} &= \theta_{cij}/w_{obj}, \quad i = 1,2\end{aligned} \tag{19}$$

Appendix C: Computation of time to goal from the current robot location

Trajectories are composed of combinations of straight lines, circular, clothoid, and anti-clothoid curves. The time-to-goal functions (t_G) are computed as the sum of several terms, the switching times: time for angular acceleration (t_α), time for linear acceleration at constant angular non-zero velocity (t_{aw}), time at constant angular velocity (t_w), time for angular deceleration until alignment with the goal ($t_{-\alpha}$), time for linear acceleration once the robot is aligned

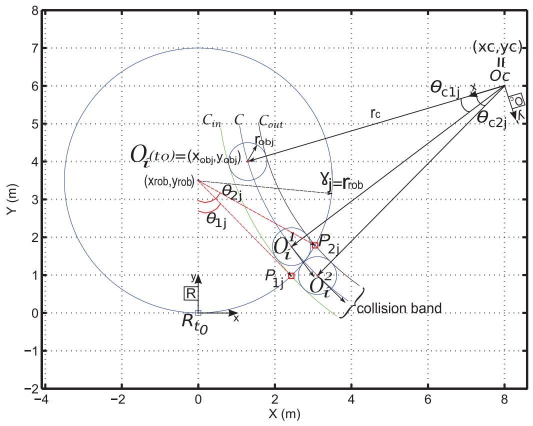


Fig. 30. Collision band, path γ_j with curvature radius r_{rob}, and collision points P_{1j} and P_{2j} with a circular object that moves along a circular trajectory in the robocentric (R) configuration space.

(t_a), and time at maximum linear velocity towards the goal (t_v). The total time to goal in both cases is

$$t_{Gw} = t_\alpha + t_w + t_{-\alpha} + t_a + t_v \qquad \text{(clothoid)}$$
$$t_{Gv} = t_{aw} + t_{-\alpha} + t_a + t_v \qquad \text{(anti-clothoid)}$$

The switching time for starting angular deceleration is computed when $\theta_a = \omega_c^2/2 * \alpha_m$ is reached, being $\omega = 0$ when the robot is aligned to the goal (x_G, y_G). The switching times are computed as follows.

(a) Time for angular acceleration (t_α)/deceleration ($t_{-\alpha}$)

$$t_\alpha = (w_\alpha - w_c)/\alpha_m$$
$$\omega_\alpha = \min(\omega_{top_w}, \omega_m); \quad \omega_{top_w} = w_c + \alpha_m * t_{top_w}$$
$$t_{-\alpha} \simeq w_c/\alpha_m; \quad \omega_{t_{-\alpha}} = w_c - \alpha_m * t_{-\alpha} \simeq 0$$

Velocity ω_α will be the angular velocity reached (ω_{top_w}) when the value $\theta_a = \omega_c^2/2 * \alpha_m$ with respect to the goal direction is measured or ω_m is reached, whilst v_c is kept. Then an angular deceleration starts until aligning the goal. The coordinates at the end this stage are ($\beta = \{\alpha, -\alpha\}$)

$$x_\beta = v_c * \int_{t_0}^{t_\beta} \cos(\theta(t))\,dt$$
$$y_\beta = v_c * \int_{t_0}^{t_\beta} \sin(\theta(t))\,dt$$
$$\theta_\beta = w_c * t_\beta + (\alpha_m * t_\beta^2)/2$$

These are non integrable equations, so they are solved numerically.

(b) Time at constant angular velocity (t_w). Time until the orientation reaches θ_a from the initial θ_c:

$$t_w = (\theta_c - \theta_a)/\omega_c$$

Both the angular and linear velocities (v_c, ω_c) are kept. The coordinates at the end of this stage are

$$\begin{aligned} x_\omega &= (v_c/\omega_c) * \sin(\theta_c - \theta_a) \\ y_\omega &= (v_c/\omega_c) * (1 - \cos(\theta_c - \theta_a)) \\ \theta_\omega &= \theta_c - \theta_a \end{aligned}$$

(c) Time at maximum linear acceleration and constant angular velocity (t_{aw}):

$$t_{aw} = (v_{aw} - v_c)/a_m; \quad v_{aw} = \min(v_{top_v}, v_m)$$
$$v_{top_v} = v_c + a_m * t_{top_v}$$

This stage finishes when the angular position with respect to the goal direction (θ_a) is reached. The coordinates reached at the end of this stage are

$$\begin{aligned} x_{aw} &= \int_{t_0}^{t_{aw}} v(t) * \cos(\theta(t))\, dt \\ y_{aw} &= \int_{t_0}^{t_{aw}} v(t) * \sin(\theta(t))\, dt \\ \theta_{aw} &= w_c * t_{aw} \end{aligned}$$

(d) Time at maximum linear acceleration (t_a):

$$t_a = (v_m - v_c)/a_m; \quad v_a = v_c + a_m * t_a$$

The stage starts when the robot is aligned ($\omega = 0, \theta_a = 0$), and accelerates up to the maximum linear velocity. The coordinates reached are

$$\begin{aligned} x_a &= (v_c + (a_m * t_a)/2) * \cos(\theta_g) * t_a \\ y_a &= (v_c + (a_m * t_a)/2) * \sin(\theta_g) * t_a \end{aligned}$$

(e) Time at maximum linear velocity (t_v). At the start of this stage, the current location of the robot is given by

$$x_c = \sum_i x_i, \quad y_c = \sum_i y_i, \quad i = \alpha, w, a_w, -\alpha, a$$

With d_G begin the distance to the goal from (x_c, y_c), the time to goal and velocities are

$$t_v = d_G/v_v, \quad v_v = v_m, \quad w_v = 0$$