HAL
archives-ouvertes.fr

An Efficient and Generic 2D Inevitable Collision State-Checker

Luis Martinez-Gomez, Thierry Fraichard

▶ **To cite this version:**

Luis Martinez-Gomez, Thierry Fraichard. An Efficient and Generic 2D Inevitable Collision State-Checker. IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, Sep 2008, Nice, France. inria-00293508v2

HAL Id: inria-00293508
https://hal.inria.fr/inria-00293508v2

Submitted on 4 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An Efficient and Generic 2D Inevitable Collision State-Checker

Luis Martinez-Gomez[†] and Thierry Fraichard[†]

***Abstract*— An Inevitable Collision State (ICS) for a robotic system is a state for which, no matter what the future trajectory of the system is, a collision eventually occurs. ICS can be used for both motion planning (to reduce the search space) and reactive navigation (for obvious safety reasons, a robotic system should never ever move to an ICS). ICS are particularly suited for navigation in dynamic environments since they take into account the future behaviour of the moving objects. Using ICS in practice is difficult given the intrinsic complexity of their characterization. The main contribution of this paper is a *generic* and *efficient* ICS-Checker, *ie* an algorithm that determines whether a given state is an ICS or not, for planar robotic systems with arbitrary dynamics moving in dynamic environments. The efficiency is obtained by applying the following principles: (a) reasoning on 2D slices of the state space of the robotic system, (b) precomputing off-line as many things as possible, and (c) exploiting graphics hardware performances. The ICS-Checker has been applied to two different robotic systems: a car-like vehicle and a spaceship. It has also been integrated in a reactive navigation scheme to safely drive the car-like vehicle.**

***Index Terms*— Motion safety; Collision avoidance; Dynamic environments; Inevitable Collision States.**

I. INTRODUCTION

The 2007 DARPA Urban Challenge[1] has demonstrated automated cars navigating autonomously in a dynamic environment. The size and dynamics of such robotic systems make them potentially harmful for themselves and their environment. Therefore, before deploying such automated cars in the real world, among human beings in many cases, it is critical to assert their operational motion safety, *ie* their ability to avoid collision with the objects of their environment.

If needed be, the analysis carried out in [1] has shown the key impact that the presence of moving objects has on the decision-making process of a robotics system when it comes to achieving safe autonomous navigation. Thus, it is of a primary importance that, in order to decide how and where to move next, the decision-making process should explicitly reason about the future behaviour of the moving objects. Failure to do so yields navigation strategies whose motion safety is not guaranteed (in the sense that situations where collisions will eventually occur can happen).

Having said that, two issues arise: the first one concerns the determination of the future behaviour of the moving objects. In certain cases, this knowledge may be available beforehand, *eg* space applications. In most cases however, it will be necessary to estimate this future behaviour (in a deterministic or probabilistic manner) using whatever information available, typically sensor data. Once a model of the future is obtained, the second issue is to reason about it in order to produce safe navigation strategies.

This paper addresses the second issue. It builds upon the *Inevitable Collision States* concept introduced in [2] (*aka* Obstacle Shadow [3] or Region of Inevitable Collision [4], [5]). An Inevitable Collision State (ICS) for a robotic system is a state for which, no matter what the future trajectory of the system is, a collision eventually occurs. ICS take into account the (possibly estimated) future behaviour of the moving objects. ICS can be used for both motion planning (to reduce the search space) and reactive navigation (for obvious safety reasons, a robotic system should never ever decide on a move that would take it to an ICS). ICS have already been used in a number of applications: (i) mobile robot subject to sensing constraints, *ie* a limited field of view, and moving in a partially known static environment [2], (ii) car-like vehicle moving in a roadway-like environment [6], [7], (iii) spaceship moving in an asteroid field [5]. In all cases, the future motion of the robotic system at hand is computed so as to keep the system away from ICS. To that end, an ICS-Checker is used. As the name suggests, it is an algorithm that determines whether a given state is an ICS or not. Similar to a Collision-Checker that plays a key role in path planning and navigation in static environments, it could be argued that an ICS-Checker is a fundamental tool for motion planning and navigation in dynamic environments. Like its static counterpart, an ICS-Checker must be computationally efficient so that it can meet the real-time constraint imposed by dynamic environments.

Characterizing the ICS set for a given robotic system is an intricate problem since it requires in theory to reason on the state space of the robotic system at hand, and above all to consider *all possible future trajectories of infinite duration* that the robotic system can follow from any given state. However, it is shown in this paper how to design a *generic* and *efficient* ICS-Checker for planar robotic systems with arbitrary dynamics moving in dynamic environments.

The ICS-Checker has been applied to two different robotic systems: (1) a car-like vehicle with complex dynamics, and (2) a spaceship with translational momentum. It has been tested in different scenarios and embedded within a simple reactive navigation scheme where it is used to determine, at each time step, a command that drives the system at hand in a state that is not an ICS thus ensuring motion safety. The paper is organised as follows: the relevant literature is briefly reviewed in §II. Then §III recalls the definition and properties fundamental to the ICS characterization and

[†]INRIA, CNRS-LIG & Grenoble University, France.

[1]`http://www.arpa.mil/grandchallenge`

outlines the ICS-Checker. Afterwards, §IV describes the principles applied to obtain an efficient and generic ICS-Checker. The instantiation of the ICS-Checker to the car-like vehicle case is presented in §V, while the spaceship case is addressed in §VI. Experimental results are given in §VII. Finally, §VIII presents the reactive navigation scheme within which the ICS-Checker has been used.

II. Related Works

The first line of work related to ICS is that of the *Velocity Obstacles* (VO), *aka Velocity Cones* [8]–[10]. VO is a representation that actually takes into account the future behaviour of the moving objects. The constraint imposed by the moving objects are mapped into the velocity space of the robotic system considered in the form of regions that represent velocities yielding a collision later in the future (if the robotic system were to keep such a velocity). VO have been used for both reactive navigation and motion planning purposes. In a sense, VO can be viewed as a restricted form of ICS (considering only future trajectories with constant velocity).

More directly related to ICS are *Viability Kernels* and *Backward Reachable Sets*. Viability Kernel (VK) comes from the Viability Theory [11]. Given a set of *viable* states for a control system, *eg* collision-free states, its VK is the set of states for which there exists a control that will maintain the system considered within the viable set. Backward Reachable Set (BRS) [12] is one of the two basic types of reachable sets for control systems. Given a set of target states, its BRS is the set of states from which trajectories start that can reach this target set. These two concepts are clearly related to ICS, they have been used in different applications, *eg* reactive navigation [13], motion planning [14] and Air Traffic Control [15]. They are both affected by the Curse of Dimensionality though and their usage have been restricted to simple control systems so far. It is not clear too how they could be extended to handle time-dependent constraints such as the one imposed by multiple moving objects.

The ICS-Checker presented in this paper is an extension of the one proposed by one of the co-authors in [16]. The new ICS-Checker is much more efficient and can handle robotics systems with arbitrary dynamics.

III. Inevitable Collision States

This section merely recalls the key ICS properties established in [2] and presents the ICS-Checker in its general form.

A. ICS Definition

Let $\mathcal{A}$ denote a robotic system operating in a workspace $\mathcal{W}$, whose dynamics is described by a differential equation such as:

$$\dot{s} = f(s, u) \tag{1}$$

where $s \in \mathcal{S}$ is the state of $\mathcal{A}$, $\dot{s}$ its time derivative and $u \in \mathcal{U}$ a control. $\mathcal{S}$ and $\mathcal{U}$ respectively denote the *state space* and the *control space* of $\mathcal{A}$. Let $\mathcal{A}(s)$ denote the closed subset of the workspace $\mathcal{W}$ occupied by $\mathcal{A}$ when it is in the state s. $\mathcal{W}$ contains a set of n_b fixed and moving objects defined as closed subsets of $\mathcal{W}$. Let $\mathcal{B}_i$ denote such an object, $i = 1 \cdots n_b$; $\mathcal{B}$ denotes the union of the workspace objects: $\mathcal{B} = \bigcup_{i=1 \cdots n_b} \mathcal{B}_i$. Since an object $\mathcal{B}_i$ maybe moving, the notation $\mathcal{B}_i(t)$ is used to denote the subset of $\mathcal{W}$ occupied by $\mathcal{B}_i$ at a particular time t in the future. Let $\tilde{u} : [0, \infty(\longrightarrow \mathcal{U}$ denote a *control trajectory*, *ie* a time-sequence of controls. The set of all possible control trajectories over $[0, \infty($ is denoted $\tilde{\mathcal{U}}$. Starting from an initial state $s(0)$ at time $t = 0$, a *state trajectory* is derived from a control trajectory $\tilde{u}$ by integrating (1). A state trajectory is a time-sequence of states, *ie* a curve in $\mathcal{S} \times \mathcal{T}$ where $\mathcal{T}$ denotes the time dimension. Abusing notation, $\tilde{u}(s(0), t)$ denotes the state reached at time t.

*Def. 1 (**Inevitable Collision State**):* s is an ICS iff $\forall \tilde{u} \in \tilde{\mathcal{U}}, \exists t, \exists \mathcal{B}_i, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset$.

Consequently, it is possible to define the set of ICS yielding a collision with a particular object $\mathcal{B}_i$:

$$\text{ICS}(\mathcal{B}_i) = \{s \in \mathcal{S} | \forall \tilde{u} \in \tilde{\mathcal{U}}, \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\} \tag{2}$$

Likewise, the ICS set yielding a collision with $\mathcal{B}_i$ for a given trajectory $\tilde{u}$ is defined as:

$$\text{ICS}(\mathcal{B}_i, \tilde{u}) = \{s \in \mathcal{S} | \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\} \tag{3}$$

B. ICS Properties

The first property shows that $\text{ICS}(\mathcal{B})$ can be derived from $\text{ICS}(\mathcal{B}_i, \tilde{u})$ for every object $\mathcal{B}_i$ and every possible future trajectory $\tilde{u}$.

*Property 1 (**ICS Characterisation**):*

$$\text{ICS}(\mathcal{B}) = \bigcap_{\tilde{u} \in \tilde{\mathcal{U}}} \bigcup_{i=1 \cdots n_b} \text{ICS}(\mathcal{B}_i, \tilde{u}) \tag{4}$$

Complex systems having an infinite number of control trajectories, the following property permits to compute a conservative approximation of $\text{ICS}(\mathcal{B})$ by using a subset only of the whole set of possible future trajectories.

*Property 2 (**ICS Approximation**):*

$$\text{ICS}(\mathcal{B}) \subseteq \bigcap_{\tilde{u} \in \mathcal{I}} \text{ICS}(\mathcal{B}, \tilde{u})$$

with $\mathcal{I} \subset \tilde{\mathcal{U}}$, a subset of the whole set of possible future trajectories.

C. ICS Checking Algorithm

Properties 1 and 2 provide the basis for a general ICS checking scheme. The steps involved in checking whether a given state s is an ICS or not are:

1) Select $\mathcal{I} \subset \tilde{\mathcal{U}}$.
2) Compute $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ for every object $\mathcal{B}_i$ and every manoeuvre $\tilde{u}_j \in \mathcal{I}$.
3) Compute $\text{ICS}(\mathcal{B}, \tilde{u}_j) = \bigcup_i \text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ for every object $\mathcal{B}_i$.
4) Compute $\text{ICS}(\mathcal{B}) = \bigcap_j \text{ICS}(\mathcal{B}, \tilde{u}_j)$ for every manoeuvre $\tilde{u}_j \in \mathcal{I}$.
5) Determine whether $s \in \text{ICS}(\mathcal{B})$. If so return True otherwise return False.

D. Imitating Manoeuvres

The first issue raised by the previous algorithm concerns the determination of $\mathcal{I}$: what control trajectories should be considered? There is an intuitive answer to that question: as per Def. 1, it appears that what characterise a state that is not an ICS is the existence of at least one control trajectory yielding a collision-free trajectory. In this respect, the control trajectories that are important should correspond to *evasive manoeuvres* (EM), *ie* trajectories seeking to avoid collisions with the objects of the workspace.

It was established in [16] that *Imitating Manoeuvres* (IM) are a good choice for $\mathcal{I}$. An IM for a given object $\mathcal{B}_i$ is the manoeuvre where the robotic system $\mathcal{A}$ tries to achieve and maintain a zero-relative velocity *wrt* $\mathcal{B}_i$. In other words, it tries to imitate the behaviour of $\mathcal{B}_i$ (doing so permits to avoid collision with $\mathcal{B}_i$). In general, $\mathcal{A}$ cannot start imitating $\mathcal{B}_i$ right away (for instance, it does not have the proper orientation) and an IM comprises two parts: (a) the "catch-up" part at the end of which $\mathcal{A}$ achieves a zero-relative velocity with $\mathcal{B}_i$, and (b) the "follow" part during which $\mathcal{A}$ duplicates $\mathcal{B}_i$'s control trajectory. The subset $\mathcal{I}$ could therefore include: (a) one IM per moving object, and (b) a fixed number of braking manoeuvres[2].

IV. Generic 2D ICS Checking Algorithm

Even with a limited subset $\mathcal{I}$ of IM, a naive implementation of the ICS Checking algorithm outlined in §III-C yields an algorithm which is computationally expensive mainly because of the cost of computing each set $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ individually (step 2 of the algorithm) and then their union and intersection (step 3 and 4). The algorithmic complexity grows exponentially with the dimensionality of $\mathcal{S}$, the state space of $\mathcal{A}$.

For a 2D workspace $\mathcal{W}$ and a planar robotic system $\mathcal{A}$ with arbitrary dynamics $\dot{s} = f(s, u)$, and whose shape can be approximated by a disk[3], it is possible however to obtain an efficient ICS Checking algorithm. This algorithm, henceforth called ICS-Check, takes as input the state to be checked, and the current model of the workspace objects (fixed and moving) and their future behaviour. ICS-Check then returns True or False depending on whether s is an ICS or not. The efficiency of ICS-Check is obtained by applying the following principles: (a) reasoning on 2D slices of the state space of $\mathcal{A}$, (b) precomputing off-line as many things as possible, and (c) exploiting graphic hardware processing performances. These principles are presented now.

A. 2D Reasoning

The state s of $\mathcal{A}$ is a n-tuple of arbitrary dimensionality. Since $\mathcal{A}$ is planar, s can be rewritten $s = (x, y, \hat{\mathbf{z}})$ with (x, y) the workspace coordinates of $\mathcal{A}$'s reference point, and $\hat{\mathbf{z}}$ the rest of the state parameters. The primary design principle behind ICS-Check is to compute the ICS set corresponding to a 2D slice of the state space $\mathcal{S}$ of $\mathcal{A}$ (instead of attempting to perform computation in the fully-dimensioned state space), and then to check if s_c belongs to this set. Assuming the state to be checked is $s_c = (x_c, y_c, \hat{\mathbf{z}}_\mathbf{c})$, the 2D slice considered is the $\hat{\mathbf{z}}_\mathbf{c}$-slice. Such a $\hat{\mathbf{z}}_\mathbf{c}$-slice being diffeomorphic to $\mathcal{W}$, the collision constraints imposed by the workspace objects can be computed in a straightforward manner: the image of a workspace object in the $\hat{\mathbf{z}}_\mathbf{c}$-slice is simply obtained by isotropically growing the object (in a C-space like fashion) of $\mathcal{A}$'s radius (recall that $\mathcal{A}$ is disk-shaped) (Fig. 1-top and middle).

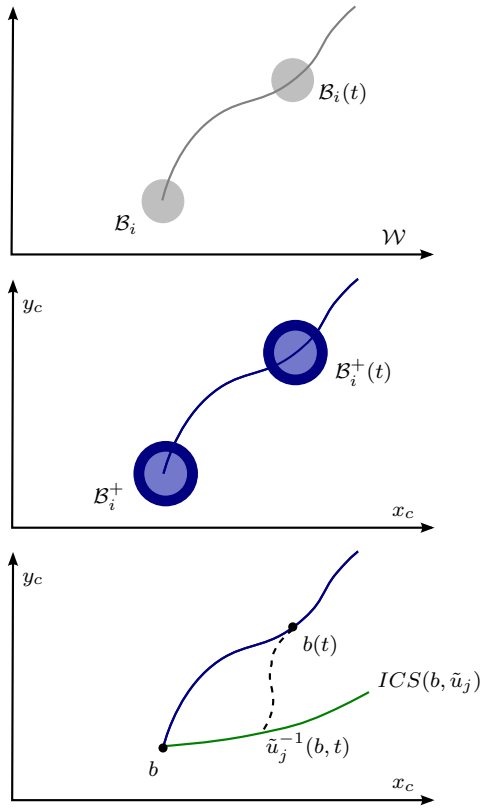


Fig. 1: Computing the image of a workspace object $\mathcal{B}_i$ in a given $\hat{\mathbf{z}}_\mathbf{c}$-slice and then computing $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ (or rather its restriction to $\hat{\mathbf{z}}_\mathbf{c}$-slice).

Now, let $\mathcal{B}_i^+$ denote such a grown obstacle and $\mathcal{B}_i^+(t)$ its position at time t. Using property 1, it can be shown that the set of states in the $\hat{\mathbf{z}}_\mathbf{c}$-slice yielding a collision with $b \in \mathcal{B}_i^+$ for a given manoeuvre $\tilde{u}_j$ is:

$$\bigcup_t \tilde{u}_j^{-1}(b, t) \tag{5}$$

where $\tilde{u}_j^{-1}$(b, t) denotes the state s such that $\tilde{u}_j(s, t) = b$. This set actually corresponds to $\text{ICS}(b, \tilde{u}_j)$ for the $\hat{\mathbf{z}}_\mathbf{c}$-slice considered (Fig. 1 bottom). Accordingly:

$$\text{ICS}(\mathcal{B}_i, \tilde{u}_j) = \bigcup_b \text{ICS}(b, \tilde{u}_j) = \bigcup_b \bigcup_t \tilde{u}_j^{-1}(b, t) \tag{6}$$

As illustrated in Fig. 1 bottom, the general shape of $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ is a semi-infinite stripe originating at $\mathcal{B}_i(0)$.

[2]Braking manoeuvres are a special case of IM where $\mathcal{A}$ imitates the behaviour of fixed objects by standing still.

[3]This assumption can be lifted off, we are currently extending ICS-Check so that it can handle robotics systems with arbitrary shapes.

It is by using (6) and keeping all computations in 2D (notwithstanding the actual dimensionality of $\mathcal{S}$) that it is possible to efficiently compute the ICS set corresponding to a given $\hat{\mathbf{z}}_\mathbf{c}$-slice.

B. Precomputing As Much As Possible

In addition to the above principle, it is possible to further improve the algorithm efficiency by precomputing off-line as many things as possible. Two items are candidates for precomputation, namely the subset $\mathcal{I}$ of imitating manoeuvres (step 1 of the algorithm) and the set $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$, *ie* the set of ICS for a given object $\mathcal{B}_i$ and a given manoeuvre $\tilde{u}_j$ (step 2 of the algorithm). Note that it is not always possible to precompute these two items since they depend on the problem at hand and the particulars of the future behaviour of the moving objects. As explained in the case study presented in §V, the problem considered and the assumptions made concerning the future behaviour of the moving objects permit to precompute beforehand both these items. If precomputing is not possible (or not desirable) the algorithm can accomodate online computation with an estimate of the future behaviour of arbitrary moving objects as shown in the case study presented in §VI.

C. Exploiting Graphics Hardware Performances

Focusing now on the steps 3 and 4 of the ICS checking algorithm, it can be seen that these steps perform the union and the intersection of regions. For arbitrary regions, such operations are costly to implement however, because the regions considered are planar, it turns out that it is possible to do such unions and intersections with OpenGL[4] primitives and thus to exploit the processing power of standard Graphics Processing Unit.

As illustrated in Figs. 1 and 5, computing $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ boils down to drawing the 2D shape of the isotropically grown workspace objects in the $\hat{\mathbf{z}}_\mathbf{c}$-slice considered. This is achieved in a straightforward manner by drawing the necessary 2D shapes in the OpenGL buffer. The OpenGL display list mechanism (precompiled OpenGL commands) is intensively used to speed up this process. It permits to keep in the GPU memory most of the geometric primitives used by the algorithm thus avoiding unnecessary data transfer from CPU to GPU.

Steps 3 and 4 of ICS-CHECK that involves computing unions and intersections of arbitrary 2D shapes are performed very efficiently taking advantage of the RGB colour coding scheme. The key idea is to assign a RGB colour to each EM $\tilde{u}_j \in \mathcal{I}$ and to draw the different $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ by performing a bitwise logical AND ($\bigwedge$) at the pixel level between the EM colour and the current OpenGL buffer colour (initialized to white, *ie* #FFFFFF). Let col_j denote the colour assigned to $\tilde{u}_j$, the colour assignment is done so as to satisfy the following property:

$$\bigwedge_{\tilde{u}_j \in \mathcal{I}} col_j = 0 \text{ and } \bigwedge_{\tilde{u}_j \in \mathcal{I}_s \subset \mathcal{I}} col_j \neq 0 \qquad (7)$$

[4] http://www/opengl.org.

where 0 is the black colour and $\mathcal{I}_s$ is an arbitrary subset of $\mathcal{I}$. With such a colour assignment, colours corresponding to the same EM yield the same colour (thus performing a union), while colours corresponding to different EM yield a different colour (thus performing an intersection). If a particular pixel of the OpenGL buffer is black, it means that all the EM of $\mathcal{I}$ have contributed to its colour, In other words, it is an ICS.

V. CAR-LIKE CASE STUDY

This section describes the instantiation of ICS-CHECK to the case of a disk-shaped car-like vehicle.

A. Robotic System Model

Let us consider a disk-shaped robot $\mathcal{A}$ that moves like a car-like vehicle. A *state* of $\mathcal{A}$ is defined as a 5-tuple $s = (x, y, \theta, v, \xi)$ where (x, y) are the coordinates of the rear wheel, θ is the main orientation of $\mathcal{A}$, v is the linear velocity of the front wheel, and ξ is the orientation of the front wheels (steering angle). A control of $\mathcal{A}$ is defined by the couple (α, γ) where α is the rear wheel linear acceleration and γ the steering velocity. The motion of $\mathcal{A}$ is governed by the following differential equation:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\xi} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ v\tan\xi/L \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \alpha + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \gamma \qquad (8)$$

where L is the wheelbase of $\mathcal{A}$ and:

$$v \in [0, v_{\max}], |\xi| \leq \xi_{\max}, |\alpha| \leq \alpha_{\max} \text{ and } |\gamma| \leq \gamma_{\max}$$

B. Workspace Model

$\mathcal{A}$ moves in a 2D workspace $\mathcal{W}$ cluttered up with fixed and moving objects. Knowledge about the future behaviour of the moving objects is provided by an independent motion prediction module. Although ICS-CHECK could handle objects with arbitrary shapes and arbitrary future behaviours, the implementation in this case study assumes, without loss of generality, that the objects are disk-shaped or polygonal and that they move with a constant linear velocity (other shapes and other future behaviours could easily be accounted for). A moving object is therefore characterized by its velocity parameters (v_{obs}, θ_{obs}) with v_{obs} the velocity magnitude and θ_{obs} its direction.

C. Precomputation

Step 1 of ICS-CHECK involves the determination of a set $\mathcal{I}$ of evasive manoeuvres featuring a fixed set of Braking Manoeuvres (BM) and one Imitating Manoeuvre (IM) per moving object. BM are independent of the moving objects and can be selected a priori. Selecting extreme braking manoeuvres, *ie* with extremal controls $\alpha_{\max}$ and $\gamma_{\max}$, is sufficient. IM on the other hand are moving object-dependent: each IM is computed given the actual velocity parameters (v_{obs}, θ_{obs}) of the moving object considered.

The whole point of precomputing a given EM $\tilde{u}_j$ is to precompute $\tilde{u}_j(s, t), \forall t$ which in turn permits to determine

$\tilde{u}_j^{-1}(b,t)$ so that Eq. (6) can be solved efficiently. To that end the state space parameters $\hat{\mathbf{z}}_\mathbf{c} = (\theta, v, \xi)$, the velocity parameters (v_{obs}, θ_{obs}) and the time dimension are discretized. The procedure that determines the IM involves finding the control $u^*(t) \longmapsto \mathcal{U}$ in $t_0 \leq t \leq t_f$ which minimizes the cost function J:

$$\min_{u(t)\in\mathcal{U}} J = t_f \tag{9}$$

where t_f is free, subject to robot dynamics in (8) and boundary conditions:

$$s_0 = (x_0, y_0, \theta_0, v_0, \xi_0) \tag{10}$$

$$s_f = (free, free, \theta_{obs}, v_{obs}, 0) \tag{11}$$

The problem is solved using a bang-bang control principle. A sequence of controls $(\pm\alpha_{max}, \pm\gamma_{max})$ are used during specific time intervals. Given t_f (*ie* the time to complete the manoeuvre) the interval duration is found solving the segment linear equations for the velocity profile:

$$v(t) = \pm\alpha_{max}(t) + v_0 \tag{12}$$

$$v(t) = \mp\alpha_{max}(t - t_f) + v_{obs} \tag{13}$$

and similarly for the steering angle profile:

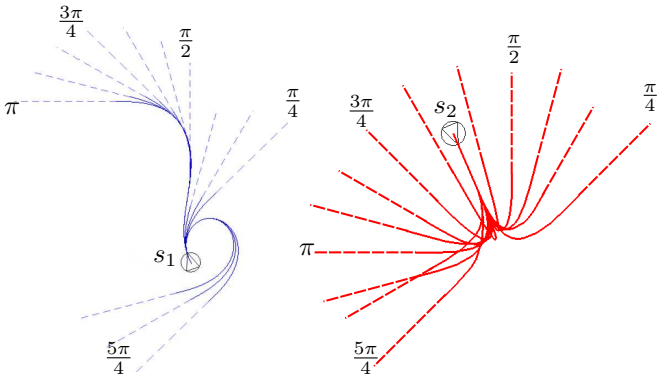
$$\xi(t) = \pm\gamma_{max}(t) + \xi_0 \tag{14}$$

$$\xi(t) = \mp\gamma_{max}(t - t_f) \tag{15}$$

The control sequence guarantees that the vehicle reaches the final conditions v_{obs} and $\xi = 0$ but not necessarily the final condition for the main orientation and thus an error function is introduced:

$$d\theta = \theta_{obs} - \theta(t_f) \tag{16}$$

The minimum value of t_f that makes $d\theta = 0$ solves also Eq. (9) so an incremental search of the value is performed changing t_f at constant intervals while evaluating Eq. (16). Fig. 2a depicts a set of IM for one value of $\hat{\mathbf{z}}_\mathbf{c} = (\theta, v, \xi)$ and different values of (v_{obs}, θ_{obs}).



(a) Car-like model: $s_1 = (0, 0, \frac{2\pi}{3}, 1.4, -0.45)$.

(b) Spaceship model: $s_2 = (0, 0, -1.2, 0.5, -1, 0)$.

Fig. 2: IM for obstacles with $v_{obs} = 1$ and $\theta_{obs} \in [\frac{\pi}{4}, \frac{5\pi}{4}]$ with a constant $\frac{\pi}{12}$ step.

VI. Spaceship Case Study

This section describes the instantiation of ICS-Check to the case of a disk-shaped spaceship similar to the one presented in [5]. It moves in an environment resembling that of the *Asteroids* game.

A. Robotic System Model

The spaceship $\mathcal{A}$ is able to thrust forward or rotate in either direction. The rotation has no effect on its translation which is subject to momentum. A *state* of $\mathcal{A}$ is defined as a 6-tuple $s = (x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$ where (x, y) are the spaceship-center coordinates, θ is the main orientation of $\mathcal{A}$, $\dot{x}$ is the speed of the spaceship along the x-axis, $\dot{y}$ is the speed of the spaceship along the y-axis, and $\dot{\theta}$ is the angular speed. A control of $\mathcal{A}$ is defined by the couple (α, τ) where α is the linear acceleration along the ship main orientation and τ is the angular acceleration. The motion of $\mathcal{A}$ is governed by the following differential equation:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} \alpha + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tau \tag{17}$$

where

$$\alpha \in [0, \alpha_{\max}] \text{ and } |\tau| \leq \tau_{\max}$$

B. Workspace Model

$\mathcal{A}$ moves in a 2D workspace $\mathcal{W}$ cluttered up with fixed and moving objects (*cf* §V-B). The implementation in this case study assumes that the objects can move with a constant linear and angular velocity. In other words, they can follow straight or circular paths. A moving object is therefore characterized by its velocity parameters: (v_{obs}, θ_{obs}) in the linear case, and (v_{obs}, ω_{obs}) in the circular case. v_{obs} is the linear velocity magnitude and θ_{obs} its direction. ω_{obs} is the angular velocity.

C. Precomputation

While precomputing improves the efficiency of the algorithm, it is also possible to accommodate an online computation of the EM using simple strategies as shown in this case study. The IM for an object moving in a linear path is achieved in a two steps process:

1) The spaceship is rotated to match its orientation to the direction of the vector from the difference of the obstacle and spaceship velocities vectors:

$$\theta = \tan^{-1}\left(\frac{v_{obsy} - \dot{y}}{v_{obsx} - \dot{x}}\right) \tag{18}$$

2) Maximum thrust is applied (α_{max}).

The process is repeated iteratively until the vector velocities are equal. Fig. 2b depicts a set of IM for a given value of $\hat{\mathbf{z}}_\mathbf{c} = (\theta, \dot{x}, \dot{y}, \dot{\theta})$ and different values of (v_{obs}, θ_{obs}). As for circular paths, the current implementation of ICS-Check does not compute the corresponding IM (it will be done

later). It is not a problem though since it has been shown that, as per property 2, the approximation of the ICS set obtained is conservative.

VII. Simulation Results

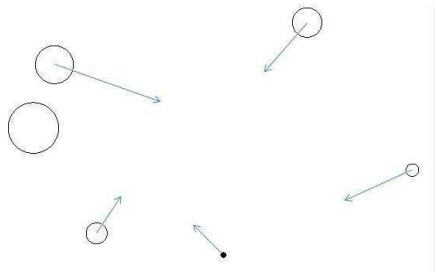


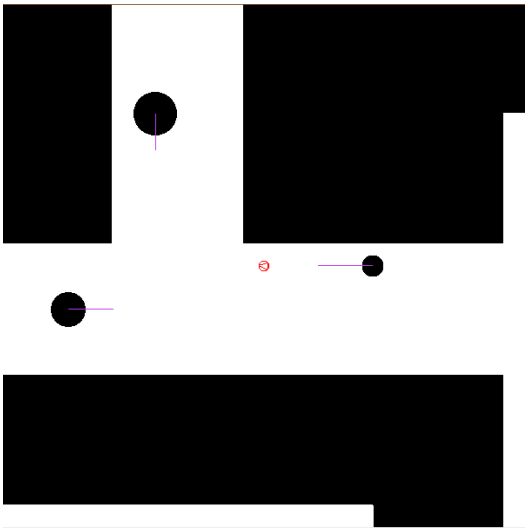
Fig. 3: Scenario #1 (car-like vehicle).

The algorithm ICS-Check outlined in §IV and detailed in §V and §VI has been implemented and tested on various scenarios. Three such scenarios are presented in §VII-A, §VII-B and §VII-C. The performances of ICS-Check are then evaluated in §VII-D.

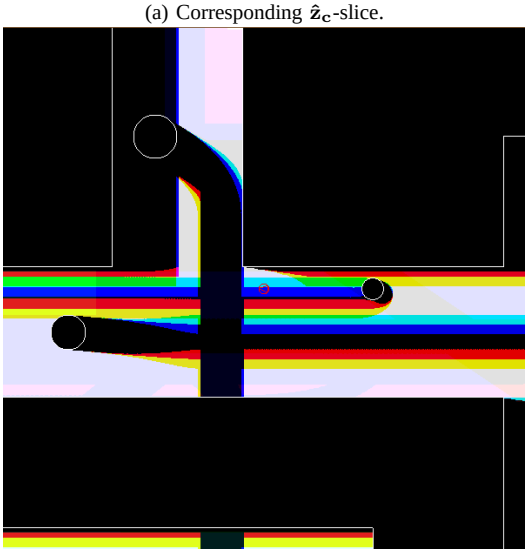
A. Scenario #1—Car-Like Vehicle

In the scenario considered here for the car-like model, the workspace features 5 disk-shaped objects: a fixed one and 4 moving ones. Let $s_c = (0, 0, \frac{2\pi}{3}, 1.4, 0)$ denote the state to be checked for ICS-ness. ICS-Check first computes the ICS set for the corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice, $\hat{\mathbf{z}}_\mathbf{c} = (\frac{2\pi}{3}, 1.4, 0)$, and then checks whether s_c is an ICS by finding out the colour of the $(0, 0)$ pixel in the $\hat{\mathbf{z}}_\mathbf{c}$-slice. Fig. 3 depicts the corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice. It features the 5 isotropically grown objects plus the central point that corresponds to s_c. The arrow at that point is the orientation $\theta = \frac{2\pi}{3}$ of s_c. The arrow attached to each object represents its velocity (direction and magnitude).

Fig. 5 illustrates how ICS-Check computes $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$, *ie* the set of ICS corresponding to a given EM $\tilde{u}_j$ and a given object $\mathcal{B}_i$. Fig. 5a depicts $\tilde{u}_j$. It is the imitating manoeuvre corresponding to the lower moving object which is moving in a $\frac{\pi}{3}$ direction. Fig. 5b is the equivalent of Fig. 1-bottom: it shows how $\tilde{u}_j^{-1}(b_i, t)$ is computed where b_i is the center of the disk object $\mathcal{B}_i$ located on the upper-left corner and moving in the $-\frac{\pi}{8}$ direction. Fig. 5c depicts $ICS(b_i, \tilde{u}_j)$ while Fig. 5d depicts the set $ICS(\mathcal{B}_i, \tilde{u}_j)$ (step 2 of ICS-Check). This procedure is repeated for every object yielding $ICS(\mathcal{B}, \tilde{u}_j)$ (Fig. 6a) (step 3 of ICS-Check). Then, it is repeated for every EM. Fig. 6b depicts the set $ICS(\mathcal{B}_i, \tilde{u}_k)$ corresponding to another EM $\tilde{u}_k$. At the end of the day, the drawing of the different $ICS(\mathcal{B}_i, \tilde{u}_j)$ on the same buffer yield the final ICS set $ICS(\mathcal{B})$ (step 4 of ICS-Check). The final step of ICS-Check simply consists in checking the colour of s_c. If it is black, s_c is an ICS otherwise it is not (step 5 of ICS-Check). In the present case, s_c is not an ICS.



(a) Corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice.



(b) $ICS(\mathcal{B})$.

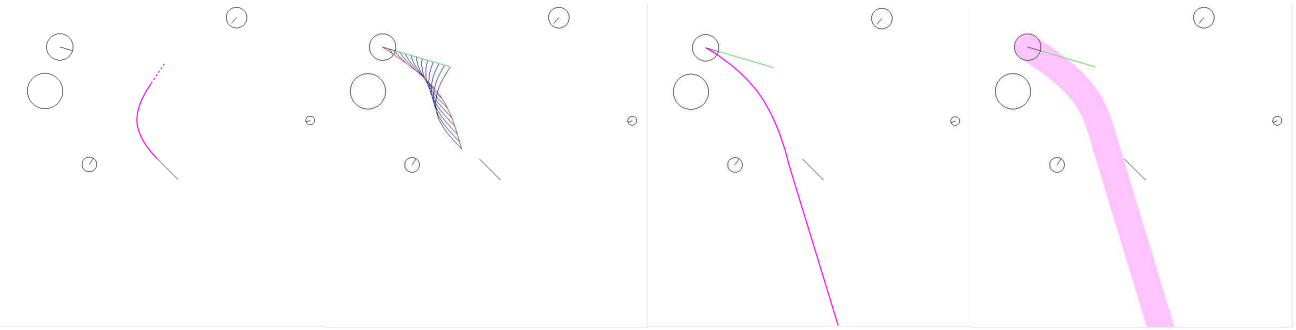
Fig. 4: Scenario #2 (car-like vehicle in a maze).

B. Scenario #2—Car-Like Vehicle

In the scenario considered here for the car-like model, the workspace features 3 disk-shaped objects moving in a maze-like environment (with polygonal fixed objects). The state to be checked for ICS-ness is $s_c = (-23, 5, 3, 1.9, 0)$. Fig. 4a depicts the corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice. s_c is the center point of the red circle and its orientation is indicated thanks to the inscribed triangle. Fig. 4b depicts the final ICS set $ICS(\mathcal{B})$. In this case too, s_c is not an ICS.

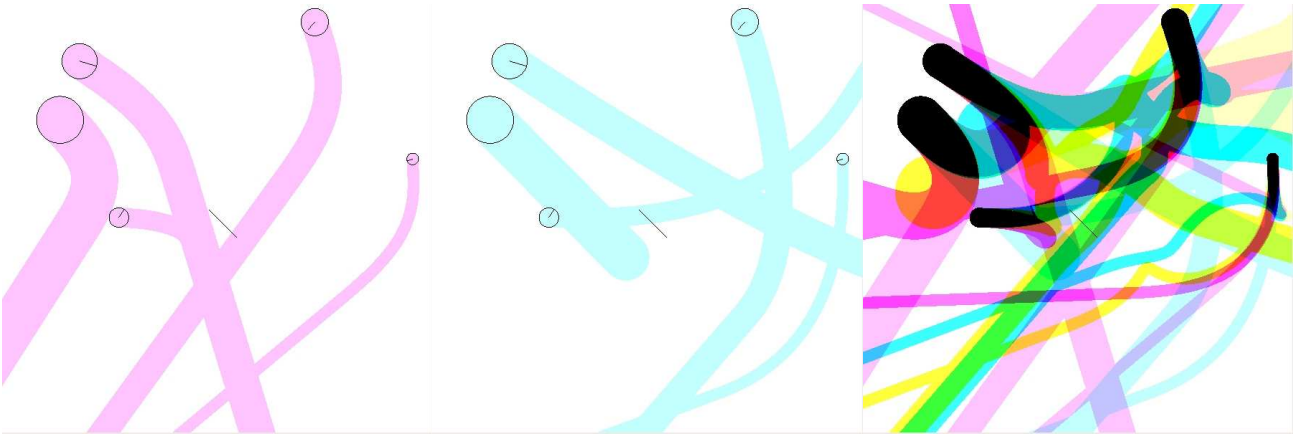
C. Scenario #3—Spaceship

In the scenario considered here for the spaceship model, the workspace features 6 disk-shaped objects: a fixed one and 5 moving ones (linear or circular paths). The state to be checked for ICS-ness is $s_c = (0, 0, 1, 1, 1, 0)$ and Fig. 7a depicts the corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice. Fig. 7b depicts the final ICS set $ICS(\mathcal{B})$. In this case too, s_c is not an ICS.



(a) $\tilde{u}_j$. (b) $\tilde{u}_j^{-1}(b_i, t)$. (c) $ICS(b_i, \tilde{u}_j)$. (d) $ICS(\mathcal{B}_i, \tilde{u}_j)$.

Fig. 5: Computing ICS$(\mathcal{B}_i, \tilde{u}_j)$ (see text).



(a) $ICS(\mathcal{B}, \tilde{u}_j)$. (b) $ICS(\mathcal{B}, \tilde{u}_k)$. (c) Final ICS set $ICS(\mathcal{B})$ (in black).

Fig. 6: Computing ICS$(\mathcal{B})$ (see text).

TABLE I: ICS-CHECK average running times.

# Objects	Time (ms)
1	1.6
2	3.4
3	7.0
4	10.7
5	13.2

D. ICS-CHECK *Performances*

The running time complexity of ICS-CHECK grows linearly with the number of objects. The running times of the current implementation of ICS-CHECK[5] has been measured using scenario #1. Objects were introduced one by one starting from the fixed object. Then, in each case, 1000 states were randomly sampled and checked for ICS-ness. Table I reports the results obtained. They are satisfactory especially since there is still room for improvement.

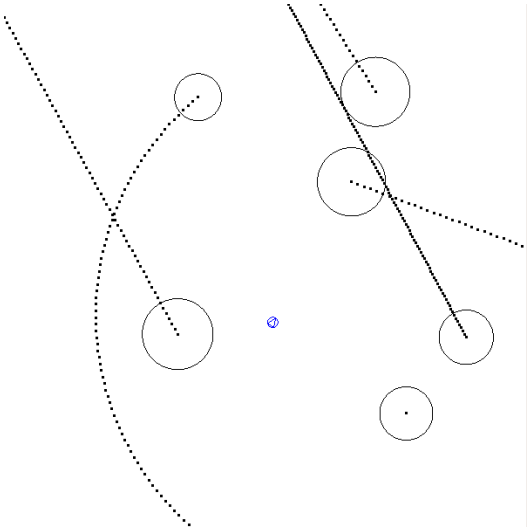
VIII. ICS-CHECK AND REACTIVE NAVIGATION

In our opinion, ICS-CHECK should play a role similar to that of a Collision Checker. It can be used in a motion planning context (to reduce the search space) but also, and above all, to achieve safe navigation (by ensuring that the navigation scheme considered does not drive the robotic system at hand in an ICS).

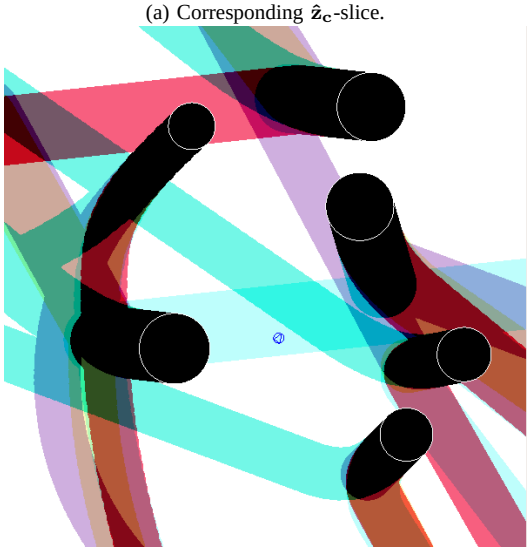
To evaluate the usefulness of ICS-CHECK in a navigation context, it has been integrated within a simple reactive navigation that operates in the following way: at each time step, the control space of the robotic system at hand is sampled according to a given sampling strategy. The state that would be reached should a given control be applied is computed and then checked for ICS-ness. If the state is not an ICS then the corresponding control is selected. Otherwise another control is checked. The process is repeated until the decision time (which is equal to one time step) is over. Given that the decision time is upper bounded, the controls corresponding to the different EM should be tested first in order to ensure motion safety.

This basic navigation scheme is only concerned with safety, it is not aimed at reaching a particular goal or achieveing a particular task. Its primary purpose is to demonstrate that, although the decision-making process has a limited look-ahead and an upper-bounded decision time, it is possible to guarantee motion safety in a dynamic environment.

[5] C++ implementation on an average laptop computer: AMD Turion 64x2 1.81 Ghz CPU, 1.93 GB RAM, and Nvidia GeForce Go 6150 GPU.



(a) Corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice.



(b) $ICS(\mathcal{B})$.

Fig. 7: Scenario #3 (spaceship among asteroids).

A video attached to this paper illustrates how the navigation scheme performs. At each time step, the video depicts the $\hat{\mathbf{z}}$-slice corresponding to the desired state that has been selected. The view is centered on the car-like vehicle. Objects moving at constant linear velocity randomly enter the field of view of the vehicle. The key thing to notice is how the vehicle stays away from the black areas.

IX. Conclusion and Discussion

This paper has presented an Inevitable Collision States (ICS) checking algorithm, *ie* an algorithm that determines whether a given state for a robotic system is an ICS or not. The algorithm presented is *generic* and *efficient*. It can be used for any planar robotic systems with arbitrary dynamics moving in dynamic environments. The ICS-Checker has been integrated in a reactive navigation scheme where it is used to safely drive a car-like vehicle with complex dynamics moving in a dynamic environment.

Now, it is important to note that the motion safety is guaranteed with respect to the model of the future that is provided to the ICS-Checker. So far, a deterministic model of the future has been considered yielding a binary definition of what constitutes an ICS.

The next key step is to move to a probabilistic model of the future so as to account for the uncertainties affecting the prediction of the future evolution of the moving objects. This in turn should yield a probabilistic definition of an ICS. The challenge then will be to design a probabilistic ICS-Checker that could be used to safely navigate partially know dynamic environments.

Acknowledgements

This work has been supported by the European Commission contracts “Cybercars-2 FP6-IST-2004-028062” and “Have-It FP7-IST-2007-212154”.

References

[1] T. Fraichard, “A short paper about motion safety,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Roma (IT), Apr. 2007.

[2] T. Fraichard and H. Asama, “Inevitable collision states. a step towards safer robots?” *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.

[3] J. Reif and M. sharir, “Motion planning in the presence of moving obstacles,” in *Proc. of the IEEE Int. Symp. on Foundations of Computer Science*, Portland, OR (US), Oct. 1985.

[4] S. LaValle and J. Kuffner, “Randomized kinodynamic planning,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Detroit, MI (US), May 1999.

[5] N. Chan, M. Zucker, and J. Kuffner, “Towards safe motion planning for dynamic systems using regions of inevitable collision,” in *Proc. of the workshop on Collision-free Motion Planning for Dynamic Systems*, Rome (IT), Apr. 2007, workshop held in association with the IEEE Int. Conf. on Robotics and Automation.

[6] S. Petti and T. Fraichard, “Safe motion planning in dynamic environments,” in *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Edmonton, AB (CA), Aug. 2005.

[7] R. Benenson, S. Petti, T. Fraichard, and M. Parent, “Toward urban driverless vehicles,” *Int. Journal of Vehicle Autonomous Systems*, vol. 6, no. 1/2, 2008.

[8] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using velocity obstacles,” *Int. Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, July 1998.

[9] A. Chakravarthy and D. Ghose, “Obstacle avoidance in a dynamic environment: A collision cone approach,” *IEEE Transactions on Systems, Man, and Cybernetics, Part A – Systems and Humans*, vol. 28, no. 5, pp. 562–574, Sept. 1998.

[10] E. Owen and L. Montano, “Motion planning in dynamic environments using the velocity space,” in *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Edmonton, AB (CA), Aug. 2005.

[11] J.-P. Aubin, *Viability Theory*. Birkhuser, 1991.

[12] I. Mitchell and C. Tomlin, “Overapproximating reachable sets by hamilton-jacobi projections,” *journal of Scientific Computing*, vol. 19, no. 1-3, pp. 323–346, Dec. 2003.

[13] M. Kalisiak and M. van de Panne, “Approximate safety enforcement using computed viability envelopes,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, New Orleans, LA (US), Apr. 2004.

[14] ——, “Faster motion planning using learned local viability models,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Roma (IT), Apr. 2007.

[15] R. Teo and C. Tomlin, “Computing danger zones for provably safe closely spaced parallel approaches,” *Journal of Guidance, Dynamics and Control*, vol. 26, no. 3, pp. 434–443, May-June 2003.

[16] R. Parthasarathi and T. Fraichard, “An inevitable collision state-checker for a car-like vehicle,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Roma (IT), Apr. 2007.