Safe Motion using Viability Kernels

Mohamed A. Bouguerra, Thierry Fraichard and Mohamed Fezari

Abstract— **A prerequisite to safe robot motion is to avoid Inevitable Collision States (ICS). However, the characterization of the ICS set is a challenge. Several approximation methods have been proposed, most of which either are overly conservative or fail to provide proper motion safety guarantees. In order to to improve safety guarantees, we build upon Viability Theory and adapt an algorithm designed to approximate the Viability Kernel, a concept similar to ICS. Our algorithm is applied first to a challenging static environment scenario. It is then extended to handle dynamic environments. Although it is not possible in general to ensure safety forever, we manage nonetheless to achieve infinite motion safety in two special cases.**

I. Introduction

With the growth of Service Robotics, the number of autonomous mobile robots is increasing. They come in a variety of types and carry out many different tasks. However, regardless of where and why they move, the safety of their motions remains a prime concern. In this respect, a robot should be able to recognize and avoid not only collision states, but also Inevitable Collision States (ICS) [1]. As the name suggests, an ICS is a state for which no matter what the future trajectory of the robot is, a collision will eventually occur. Needless to say that characterizing and avoiding ICS is crucial to motion safety.

Characterizing the set of ICS is in general intractable since it requires in theory to check for collision the infinite number of possible trajectories of infinite duration that the robot could execute from a given state. Considering only a finite subset of so-called evasive trajectories is the stance commonly taken by ICS-based approaches. It yields an approximation of the ICS set which is conservative but whose quality heavily depends on the choice of the evasive trajectories (with ill-chosen evasive trajectories, most of the states may end up being labeled as ICS).

In static workspaces, braking trajectories [2], *i.e.* trajectories that bring the robot to a stop, are good candidates if the robot can stop of course. If the robot is a plane and cannot stop mid-air, circling trajectories [3] are an obvious choice that may not work so well if the workspace features narrow corridors. In dynamic workspaces, things get worse. Even assuming that knowledge about the future behavior of the moving obstacles is available, selecting the appropriate evasive trajectories remains an open problem.

Interestingly, a concept similar to ICS have appeared in an other domain, namely the Viability Kernel (VK) from the Viability Theory [4]. Given a dynamical system, and a subset of states satisfying a given set of constraints, the VK is the subset of states from which the system can be maintained within the constrained subset forever. Note that, when the constraints are to avoid collision, the VK is actually the complement of the ICS set. Computing VK and ICS are therefore related and since there exist algorithms that compute approximations of VK, it is interesting to explore if they can be used in a collision avoidance context.

In this paper, we exploit the VK algorithm from [5] to provide a good approximation of the ICS set (good in the sense that it guarantees motion safety in a less conservative manner). First, the algorithm is adapted in order to be conservative in static environments. The algorithm is evaluated in the case of an unstoppable robot navigating narrow passages. Then, the algorithm is extended to handle dynamic environments. Although it is in general impossible to guarantee motion safety forever, we establish and demonstrate that it can be achieved for two classes of dynamic environments: the freezing class where the environment becomes static at some point, and the periodic class where the moving obstacles have a periodic behaviour.

II. Related Works

Although the details may vary, most of the proposed ICS approximation methods rely on the same principle: a subset of evasive trajectories is selected and states are labeled ICS if none of the evasive trajectories are collision-free. In static workspaces, braking trajectories (which drive the robot to a stop), are an obvious choice [6], [7]. When the robot cannot stop, *e.g.* it is a plane, circling trajectories have been considered [3]. In dynamic workspaces, imitating trajectories (which maintain zero relative velocity with the obstacle), have been proposed in [2]. Whatever the subset of evasive trajectories selected, it is difficult to ensure the quality of the approximation for all situations and not end up with most states conservatively labeled as ICS. This difficulty plus the fact that absolute motion safety requires to reason over an infinite time horizon [8] have led some authors to settle for weaker motion safety guarantees. For instance, [9] introduces passive safety; it guarantees that if a collision occurs, the robot will be at rest.

Other methods settle to even less. They aim to improve the chance of surviving collisions yet with no strict guarantees. They use other types of trajectories. For instance, trajectories that are guaranteed to be collision-free only up to a finite time [10], [11], or trajectories that are collision-free with respect to one obstacle at a time, instead of considering them all at once [12], [13].

M. Bouguerra and M. Fezari are with the Electrical Engineering Dept., Annaba Univ., Algeria mabouguerra@gmail.com, mouradfezari@yahoo.fr. Thierry Fraichard is with INRIA, CNRS-LIG and Grenoble Univ., France thierry.fraichard@inria.fr

On the viability front, diverse VK approximation methods have been proposed. The method considered in this work [5] discretizes the system in time and space, and then it unveils the discrete VK in an iterative manner. The method in [14] builds upon the first one. It defines an SVM model that can be used to find viable controls in a faster way, but does not provably converge to the actual VK. The algorithm [15] determines whether a given state is locally viable *i.e.* viable for a limited time horizon, by minimizing the cost to constraints using simulated annealing. In [16] the authors made a relation between VK and maximal reachable sets, and used the already exiting Lagrangian techniques for computing the reachable sets to under-approximate the VK.

Closer to our work, machine learning is used in [17] and [18] to approximate the VK for mobile robots. The purpose was in [18] to filter out nonviable states from the search space, to speed up motion planners, while in [17] was to help augmenting systems safety by preventing them from entering failure regions. A learning approach is prone to misclassification, which may not be a problem in the first case, but would void safety guarantees in the latter one.

III. Viability in a Nutshell

We recall here the basic concepts of the viability theory, the reader is referred to [4] for more details. Viability theory studies the evolution of dynamical systems and their capacity to satisfy *viability constraints*, *e.g.* avoiding collisions for a mobile robot, remaining dynamically balanced for a humanoid robot. Viability constraints generally define a subset of the state space of the system, the *admissible* space. A *viable* state is guaranteed to have at least one sequence of controls which, when applied from said state, will keep the system from failure, *i.e.* keep it in the admissible space. Conversely, *nonviable* states are those where failure is no longer avoidable. Note that when collision avoidance is the viability constraint then nonviable states are inevitable collision states. The *viability kernel* of an admissible space is the set of all its viable states.

These concepts can be formalized as follows. Consider the continuous-time dynamical system:

$$\begin{cases} x'(t) = F(x(t), u(t)) \\ u(t) \in U(x(t)) \end{cases} \tag{1}$$

with $x(t) \in X$ the system state at time t and $u(t)$ the control applied. $U(x(t))$ is the set of allowed controls at time t that can be state-dependent. Viability constraints are characterized by the compact subset A of the state space within which the system must be kept.

Let $x(\cdot) : t \to x(t)$ denote the *evolution* of the system when the sequence of controls $u(\cdot) : t \to u(t)$ is applied to it. An evolution $x(\cdot)$ is said to be viable in the admissible space A on an interval $[0, T[$ (where $T \leq +\infty$) if for every time $t \in [0, T[$, $x(t)$ belongs to A. Accordingly, we call *viable* states those from which starts at least one evolution viable in A at all times *i.e.* on the interval $[0, +\infty)$.

The basic problem in the viability theory is to find the *viability kernel* of the admissible space, the set of all its viable states.

Def. 1 (Viability Kernel):

$$\text{Viab}_F(A) = \{x_0 \in A \mid \exists\, x(\cdot) : x(0) = x_0 \text{ and } \forall t \geq 0,\, x(t) \in A\} \tag{2}$$

The admissible space A is said to be *viable* under F whenever it is equal to its viability kernel $\text{Viab}_F(A)$.

The next question is then to provide the *regulation map*, the set-valued map $x \in \text{Viab}_F(A) \rightsquigarrow R(x) \subset U(x)$ that indicates at each state the *viable* controls that, when applied, will keep the system inside the VK.

Def. 2 (Regulation Map): The set-valued map $x \rightsquigarrow R(x)$ is a regulation map governing viable evolutions if the VK of A is *viable* under the control system:

$$\begin{cases} x'(t) = F(x(t), u(t)) \\ u(t) \in R(x(t)) \end{cases} \tag{3}$$

In the following section, we present the *viability algorithm*, devised by Saint-Pierre in [5], that aims to approximate the VK of an admissible space under a control system, and also to provide the regulation map, that governs evolutions viable in that admissible space.

IV. Viability Algorithm

The viability algorithm operates in two stages. It first approximates the original continuous problem by discretizing it in time and space. Then, it computes the exact VK for the discretized problem in an iterative way.

A. Discretization

We first discretize the problem in time. There exist more or less sophisticated ways to transform continuous-time models into discrete counterparts. We use for instance the simplest, the Euler explicit discrete scheme. Let ρ be the time discretization step, and G the discrete dynamical system:

$$\begin{cases} x_{n+1} = G(x_n, u_n) = x_n + \rho F(x_n, u_n) \\ u_n \in U(x_n) \end{cases} \tag{4}$$

Then, we reduce the state space to a finite subset of X, for instance a grid of step h, denoted X_h. But since we cannot define the above discrete system on the finite grid X_h, because nothing guarantees that for all $x_n \in X_h$, the image $G(x_n, u_n)$ is in X_h, we introduce G^r the extension of G with a ball of radius r:

$$G^r = G + r\mathcal{B} \tag{5}$$

where $\mathcal{B}$ is the unit ball. And we choose r such that:

$$\forall x \in X_h\,,\, G^r(x, u) \cap X_h \neq \emptyset \tag{6}$$

we can take for instance, and without loss of generality $r = h$. Thus, we obtain the discrete and finite dynamical system:

$$\begin{cases} x_{n+1} \in x_n + \rho F(x_n, u_n) + h\mathcal{B} \\ u_n \in U(x_n) \end{cases} \tag{7}$$

Finally, the control space is also reduced to a finite subset, and so the set of controls that are allowed at each state x_n becomes finite and we denote it $U_d(x_n)$.

B. Construction of the Discrete and Finite Viability Kernel

In the next step, we construct the VK of $A_h = A \cap X_h$ under the discrete and finite system (7) iteratively as follows:

We initialize $A^0 = A_h$, and we define the sequence of subsets $A^1, A^2, A^3, ..., A^n, ...$ recursively, such that:

$$A^{n+1} = \{x \in A^n \mid \exists u \in U_d(x) \, : \, G^r(x,u) \cap A^n \neq \emptyset\} \quad (8)$$

This will basically refine the grid A_h iteratively, by discarding at each iteration the states from which the system will inevitably leave the grid in the next step.

Now if we denote $A^\infty = \bigcap_{n=0}^{\infty} A^n$, it is provable to say that A^∞ is the largest subset of A_h such that:

$$\{\forall x \in A^\infty \, , \, \exists u \in U_d(x) \, : \, G^r(x,u) \in A^\infty\} \quad (9)$$

or equivalently:

$$A^\infty = \text{Viab}_{G^r}(A_h) \quad (10)$$

and since A_h is finite, there exists a finite integer p such that:

$$\forall n \geq p \, : \, A^n = A^p \quad (11)$$

thus we obtain the exact VK of A_h under the discrete and finite system (7), in a finite number of steps.

Then, we can recover the discrete regulation map $R_{h,\rho}$, defined for every state in $\text{Viab}_{G^r}(A_h)$ as:

$$R_{h,\rho}(x) = \{u \in U_d(x) \mid G^r(x,u) \in \text{Viab}_{G^r}(A_h)\} \quad (12)$$

This regulation map provides all the viable actions that are available at each state. Choosing actions according to $R_{h,\rho}$, the system is ensured to stay in A_h at all times.

It is important to note that although the finite and discrete kernel $\text{Viab}_{G^r}(A_h)$ may consist of a good approximation to the VK for the continuous problem $\text{Viab}_F(A)$, Saint-Pierre proved that, and gave the conditions for which, the approximated kernel $\text{Viab}_{G^r}(A_h)$ converges to the actual kernel $\text{Viab}_F(A)$ as h and ρ go to zero.

$$\lim_{h,\rho \to 0} \text{Viab}_{G^r}(A_h) = \text{Viab}_F(A) \quad (13)$$

We refer the reader to [5] for more details on the convergence issue, as well as the proof for the result stated in (10).

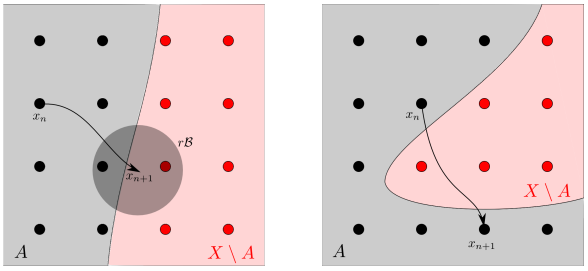
V. Limitations of the Viability Algorithm

The viability algorithm as described in the previous section does not readily fit our purpose with regard to guaranteed motion safety. It shows the following two issues:

A. Non Conservativeness

The algorithm does not provide a conservative approximation. That is the approximate kernel $\text{Viab}_{G^r}(A_h)$ is not necessarily included in the exact one $\text{Viab}_F(A)$. Put differently, evolutions that are viable in A_h under G^r may not be viable in A. This is due to two reasons:

The first one is a result of working with G^r instead of G. The algorithm could wrongly not discard a nonviable state x_n, whose all successors $G(x_n, u_n)$ lie outside A, just because $G^r(x_n, u_n) \cap A_h \neq \emptyset$. This is distinctly possible to occur near the boundary of A (Fig. 1a).



(a) $G(x_n, u_n)$ lies outside A, eventhough $G^r(x_n, u_n) \cap A_h \neq \emptyset$

(b) The two endpoints are in A, but the trajectory in between is not

Fig. 1: Algorithm Issues

Second, evolutions viable in A_h under G^r are only expected to satisfy the constraints at the discrete time steps. We can have an evolution $x(\cdot)$ where x_n and x_{n+1} belong to A_h, but the trajectory in between leaves A (Fig. 1b).

B. Dynamic Environments

Most workspaces where robots are set to operate are dynamic, they feature moving obstacles. This causes the admissible space A to become changing *i.e.* time dependent.

The viability algorithm was meant, and only applied, to approximate the VK when the admissible space A is static. To cope with dynamic environments, an obvious suggestion would be to move to the state-time space framework. That is to add the absolute notion of time as an extra dimension, denoted here τ, and to augment the dynamics to become:

$$\begin{cases} (x_{n+1}, \tau_{n+1}) & = H((x_n, \tau_n), u_n) \\ & = (x_n + \rho F(x_n, u_n), \tau_n + \rho) \\ u_n \in U_d \end{cases} \quad (14)$$

Then, A can be defined in the state-time space as the set of all collision-free tuples (x, τ).

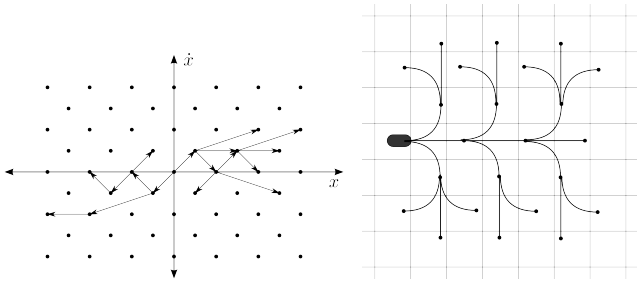
Moving to the state-time space is indeed the key to deal with dynamic environments, however, we cannot apply the viability algorithm right away to compute the VK of the so defined A under H, because of the following reason. As the admissible space should be compact, the time dimension must be bounded at some T, and so all evolutions starting from A will have to leave it anyway, because the time variable will keep increasing and never stop or go backward. As a result, the algorithm would return an empty set.

The way we have dealt with dynamic environments, as well as the method to ensure a conservative approximation are presented in the next section.

VI. Extending the Limits

A. Providing a Conservative Approximation

To turn over the issues presented in V-A, and set back motion safety that we strive for, we take the following measures.



(a) A state lattice for a one dimensional double integrator system.

(b) An approximate state lattice for the Dubins car.

Fig. 2: Construction of state lattices.

First, we opt for another discretization of the state space X_h. Instead of defining a random grid, we construct a state lattice based on the model of the system. The state lattice is constructed as follows:
After choosing a time step ρ, and a finite subset of controls U', we start from some *origin* state, and we grow a reachability graph by applying all the possible controls in U' for a fixed duration ρ, with the assumption that the control is constant during the time step.
For fully-actuated systems, we obtain a regular lattice which has a fixed spacing in each dimension (Fig. 2a) [19]. But we can also construct approximate lattices for under-actuated systems, by growing the reachability graph upon a regular grid, and ensure that no more than one graph vertex lies on each grid cell (Fig. 2b) [20].
By building X_h as such, we do not need to work with the extension of G with $r\mathcal{B}$, since by construction $\forall x \in X_h : G(x,u) \in X_h$. Thence, we preserve the system safety.

To cope with the second issue, we make a preprocessing step to filter the controls available at each state in A_h. We only allow controls for which the system will stay in A until it reaches the next state in A_h. So instead of working with U_d, we define the map $U_e : x_n \in A_h \rightsquigarrow U_d(x_n)$ so that:

$$\forall u_n \in U_e(x_n) : G(x_n, u_n) \in A_h \text{ and } x(t) \in A,\ \forall t \in [n, n+\rho] \tag{15}$$

A collision checker such as PQP [21] can be used to fulfill the second condition *i.e.* the trajectory between the two states is collision free, or equivalently, it remains in A.

By doing these modifications, we can now ensure a conservative approximation of the kernel, which means that $\text{Viab}_G(A_h) \subset \text{Viab}_F(A)$. So starting from any state in $\text{Viab}_G(A_h)$, the system is guaranteed to remain in A.

B. Dealing with Dynamic Environments

In dynamic environments, as the time dimension has to be bounded, it is not possible to determine states that are viable forever. What we can do instead, is to find the states that are viable up to the bounded time horizon T. To this end, we formulate the problem as finding the viability kernel with target.

If a subset $C \subset A$ is regarded as a target, the viability kernel of A with target C is the subset of states in A from which starts at least one evolution viable in A forever or until it reaches C in finite time. This relatively new concept has been introduced in [4], and it can be computed using the same viability algorithm with slight adaptations:
First, we set $A^0 = A_h \setminus C$, and instead of (8) we define the sequence of subsets $A^1, A^2, A^3, ..., A^n, ...$ as follows:

$$A^{n+1} = \{x \in A^n \mid \exists u \in U_d(x) : G^r(x,u) \cap (A^n \cup C) \neq \emptyset\} \tag{16}$$

In our case, if we set $C = \{(x,\tau) \in A \mid \tau = T\}$, the viability kernel with target $\text{Viab}_H(A,C)$ represents the set of states from which the system is guaranteed to stay collision-free until time T. This is in fact all we can determine in the general case.

Nonetheless, there are some special cases of dynamic environments whose nature allows us to rewrite the dynamics of the system in such a way we circumvent the problem stated in V-B. We will then be able to compute their VK using the viability algorithm normally, and so we guarantee system safety forever. We distinguish two special cases that we dub respectively: *freezing* world and *periodic* world.

1) Freezing case: In the first case, there exists a finite time T in the future at which the admissible space A *freezes*, it stops changing. Be it the obstacles stay still, or leave the workspace never to enter it again.

A characteristic of this class of dynamic environments is that:

$$\forall \tau > T : (x,\tau) = (x,T) \tag{17}$$

This allows us to rewrite the dynamics of the system as:

$$\begin{cases} (x_{n+1},\tau_{n+1}) &= H((x_n,\tau_n),u_n) \\ &= (x_n + \rho F(x_n,u_n), \tau_n + \rho) \text{ if } \tau_n < T \\ (x_{n+1},\tau_{n+1}) &= H((x_n,\tau_n),u_n) \\ &= (x_n + \rho F(x_n,u_n), \tau_n) \text{ if } \tau_n \geq T \\ u_n \in U_d & \end{cases} \tag{18}$$

By doing so, the system need not to leave the admissible space A that we bound its time dimension at T, and so we can compute the VK of A under the system (18) using the viability algorithm normally. Not only that, by determining the VK of the bounded A, we can also tell whether the states whose $\tau > T$ are viable or not, thanks to (17)

2) Periodic case: In the second case, the change of A is periodic. There is a time T for which the obstacles return to their initial states and repeat the same motion all over again.

This type of workspaces (*aka* repetitive workspaces) is often observed in practice. The workspace may consist of moving obstacles having a continuous periodic motion *e.g.* revolving doors, sliding doors, and elevators; or having discrete modes, changing from one another in a periodic manner.

As with the previous case, this class of environments has the following characteristic:

$$\forall \tau > T : (x,\tau) = (x, \tau \bmod T) \tag{19}$$

which could lead us to the following dynamics:

$$\begin{cases} (x_{n+1}, \tau_{n+1}) & = H((x_n, \tau_n), u_n) \\ & = (x_n + \rho F(x_n, u_n), (\tau_n + \rho) \bmod T) \\ u_n \in U_d \end{cases} \tag{20}$$

In like manner, we bound the admissible space's time dimension at T, and we compute its VK under (20) using the viability algorithm as normal. Here also, the computed VK allows us to determine the viability of states even beyond the time horizon T, because of (19).

VII. Simulation Results

To assess its efficiency, we have implemented the viability algorithm for a simple robotic system in different workspace scenarios. For each scenario, we have computed the approximate VK and the corresponding regulation map. Then, we have used this regulation map to safely navigate the workspace. The navigation scheme is rather simple. Starting from a state of the VK, the robot chooses at each time step, a control to apply among the viable controls that are available at the present state. This set of viable controls is determined according to the regulation map. The choice of the control depends on the objective the robot is about to achieve, and it would maximize some defined utility function.

We present, for each of the test cases, figures illustrating 2D slices of the VK, at fixed values of velocity v, and time t. The VK is shown in green, and ICS in red. We have also simulated the navigation tasks using ROS and Gazebo. We attached to the paper a video showing the simulation runs.

A. Robot Model

We consider a point mass, that we denote $\mathcal{A}$, whose acceleration a is directly controlled. It operates in a 2D workspace $\mathcal{W}$. Its state x is represented by a tuple (p, v), where p is its position, and v is its Cartesian velocity. The motion is governed by:

$$\begin{cases} \dot{p} = v \\ \dot{v} = u \end{cases} \tag{21}$$

where $|v| \leq v_{max}$ and $|u| \leq a_{max}$.

B. The viability Algorithm at Work

1) Unstoppable robot amidst narrow corridors: We consider a static workspace as depicted in (Fig. 3a). We lower bound the velocity of the robot to be $v > v_{min}$, and we choose v_{min} so that we prevent the robot from moving in circles, given the constraint: $|u| \leq a_{max}$. The viability algorithm comes in most handy, whenever it is unclear how to choose safe evasive trajectories, thanks to its brute force nature.

The approximated kernel is depicted in (Fig. 3a) as 2D slice at fixed value of v. Using the regulation map corresponding to this approximated kernel, the robot was able to safely navigate the workspace, as shown in the simulation. As expected, the viability algorithm was able to find if, from any state, a safe trajectory exists, however arbitrary this trajectory might be.

2) The compactor scenario: In this scenario, the workspace features two obstacles, one of which is moving towards the other at constant velocity until they meet, resembling to a trash compactor (Fig. 3b). As simple as this scenario might seem, none of the previously proposed methods could give, in their respective systematic way, a reasonably conservative approximation of its ICS set (or equivalently the VK). The VK approximation provided by the viability algorithm (Fig. 3b) was conservative, yet good enough to allow the robot to safely traverse the compactor, as shown in the simulation. Note that this example falls within the freezing class that we discussed in VI-B.1.

3) Periodic workspace: This scenario is the other special class of dynamic environments. We have shown in the previous section that absolute motion safety can be guaranteed in this class of workspaces using the viability algorithm. We consider here a simple example. It consists of two cells where the only way to go from one another is to pass through a revolving door in the middle (Fig. 3c).

Using the VK approximation computed with the viability algorithm (Fig. 3c), the robot was able to safely go back and forth between the two cells many times, and it is provable it can continue doing so forever without entering into collision. The simulation run as well as the 2D slice of the kernel approximation that changes in real time, are illustrated in the attached video.

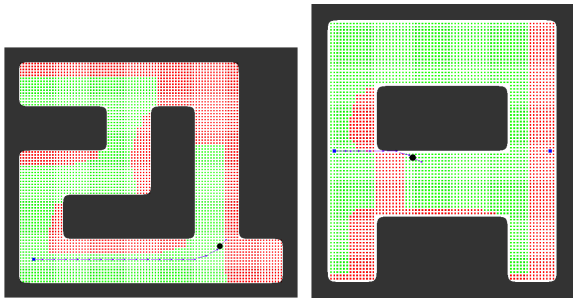
C. Computational Efficiency

The computational complexity of the viability algorithm is exponential in the state and the action spaces. The algorithm was implemented on an average laptop using Python. It took on an average of three runs, 54, 1614, and 1088 seconds, to compute the viability kernel for the first, second, and the third scenarios respectively. The running time gets significantly bigger as the dimension increases *i.e.* as with the additional time dimension in the case of dynamic workspaces. The running time does also depend on the size of the admissible space A, since it is the set that is manipulated and iterated over. That said, the algorithm would perform better in more cluttered workspaces. This in part justifies the difference between the running times of the second and the third cases. Thankfully, the VK has to be computed only once and offline, and then it can be used online to safely navigate the workspace, as in the scenarios considered.

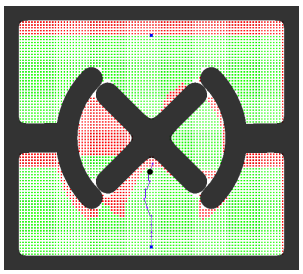
VIII. Discussion and Conclusion

We have used the viability algorithm as a mean to approximate the VK, and thus the ICS set, to ensure motion safety of mobile robotic systems. It needed to make some modifications to the algorithm to obtain a conservative approximation of the VK, and also to cope with dynamic environments.

The method has shown many strengths as well as shortcomings. One advantage is that it can be used to compute the VK for systems with any dynamics and of any dimension, at least theoretically. Secondly, it does not require to predefine



(a) $v_x = 6, v_y = 6, t = 16$ (b) $v_x = 8, v_y = -4, t = 8$



(c) $v_x = 2, v_y = 6, t = 10$

Fig. 3: 2D slices of the VK of the test scenarios, corresponding to fixed values of velocity v_x, v_y, and time t

a set of evasive trajectories; instead, it will brute-force search them all, but in a way that mitigates the combinatorial explosion. This is equivalent to say it is resolution-complete. At any given state, it will find an evasive trajectory, if one exists under that discretization. This would be particularly useful when no clear choice of evasive trajectories is available. Also, the algorithm not only determines whether or not a given state is safe, but also it provides the viable controls that will maintain the system inside the VK. Moreover, it offers under some assumptions a convergence proof. The VK of the discretized problem converges to the continuous one, as the discretization steps decrease.

In dynamic environments, we have shown it not to be possible in general to ensure absolute motion safety. This is due to the fact that we must bound the time dimension, and reason over a finite time horizon. Nonetheless, we were able to obtain motion safety guarantees, for some special classes of dynamic environments. Another interesting feature of the algorithm is the possibility to define states to avoid of any nature other than collision states *e.g.* tipping over.

On the other hand, the main drawback of the algorithm is its computational complexity, it is exponential in the state and the control spaces. The algorithm may not be fast enough for real time settings, but it is sill interestingly useful in applications where the VK is computed off-line and then is used for safe navigation afterwards. This has been shown to be doable with the three workspace scenarios that we considered in our experiments.

This work can be extended in many ways. We could evaluate the algorithm on more complex robotic systems and in other scenarios. We could also make an experimental comparison with some ICS approximation techniques. Then, It would be of interest to investigate other potential connections between the viability theory and robots motion safety.

REFERENCES

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

[2] L. Martinez-Gomez and T. Fraichard, "An efficient and generic 2d inevitable collision state-checker," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, Nice (FR), 2008.

[3] T. Schouwenaars, J. How, and E. Feron, "Receding horizon path planning with implicit safety guarantees," in *American Control Conference, 2004. Proceedings of the 2004*, Bonston (US), Jun 2004.

[4] J.-P. Aubin, A. Bayen, and P. Saint-Pierre, *Viability Theory: New Directions*. Springer, 2011.

[5] P. Saint-Pierre, "Approximation of the viability kernel," *Applied Mathematics and Optimization*, vol. 29, no. 2, pp. 187–209, 1994.

[6] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *Robotics and Automation, 2007 IEEE International Conference on*, Roma (IT), 2007.

[7] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," in *Robotics and Automation, 2007 IEEE International Conference on*, 2007.

[8] T. Fraichard and T. Howard, "Iterative motion planning and safety issue," in *Handbook of Intelligent Vehicles*, A. Eskandarian, Ed. Springer, 2012, pp. 1433–1458.

[9] S. Bouraine, T. Fraichard, and H. Salhi, "Provably safe navigation for mobile robots with limited field-of-views in dynamic environments," *Autonomous Robots*, vol. 32, no. 3, pp. 267–283, 2012.

[10] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[11] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[12] N. Chan, J. Kuffner, and M. Zucker, "Improved motion planning speed and safety using regions of inevitable collision," in *17th CISM-IFToMM symposium on robot design, dynamics, and control*, 2008.

[13] Z. Shiller, O. Gal, and A. Raz, "Adaptive time horizon for online avoidance in dynamic environments," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, Sept 2011.

[14] G. Deffuant, L. Chapel, and S. Martin, "Approximating viability kernels with support vector machines," *Automatic Control, IEEE Transactions on*, vol. 52, no. 5, pp. 933–937, 2007.

[15] N. Bonneuil, "Computing the viability kernel in large state dimension," *Journal of Mathematical Analysis and Applications*, vol. 323, no. 2, pp. 1444–1454, 2006.

[16] S. Kaynama, J. Maidens, M. Oishi, I. M. Mitchell, and G. A. Dumont, "Computing the viability kernel using maximal reachable sets," in *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, 2012.

[17] M. Kalisiak and M. van de Panne, "Approximate safety enforcement using computed viability envelopes," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, Apr. 2004.

[18] ——, "Faster motion planning using learned local viability models," in *Robotics and Automation, 2007 IEEE International Conference on*, Roma (IT), Apr. 2007.

[19] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[20] J. Barraquand and J. C. Latombe, "Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles," *Algorithmica*, vol. 10, no. 2-4, pp. 121–155, 1993.

[21] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: a hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996.