Ministère de l'Enseignement Supérieur et de la Recherche scientifique
وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA

جامعة باجي مختار- عنابة

Faculté des Sciences de l'Ingéniorat **Année : 2019/2020**
Département d'Electronique

THESE

Présentée en vue de l'obtention du diplôme de ***DOCTORAT 3ème cycle***

Navigation Sûre de Robots Mobiles par Théorie de Viabilité

(Safe Navigation of Mobile Robots using Viability Theory)

Option

Automatique et Signaux

Par

Bouguerra Mohamed Amine

Rapporteur : Pr. Fezari Mohamed MCA Université Badji Mokhtar, Annaba

Devant le Jury

Président :	Doghmane Noureddine	Pr.	Univ. Annaba
Examinateur :	Lakel Rabeh	Pr.	Univ. Annaba
Examinateur :	Moussaoui Abdelkrim	Pr.	Univ. Guelma
Examinateur :	Lafifi Mohamed Mourad	MCA	Univ. Annaba

Année Universitaire : 2019/2020

الملخص

يُعتبر ضمان القيادة الذاتية الآمنة مشكلة جوهريّة في العديد من تطبيقات الروبوتات المتنقّلة. تُدرس هذه المسألة عادةً في إطار حالة الاصطدام الحتمية (ICS) الذي يقتصر بطبيعته على تجنّب الاصطدام. تنحرّى هذه الأطروحة استعمال نظريّة أكثر عموما تعرف بنظريّة البقاء (Viability Theory) كبديل عندما تتعلّق القيادة الآمنة بقيود متعدّدة وليس مجرّد تفادي التصادم. في قلب نظريّة البقاء، هناك ما يعرف بنواة البقاء (Viability Kernel)، وهو مجموع حالات النظام الروبوتي التي يمكن أن ينطلق منها مسار واحد على الأقل يخضع لجميع قيود الحركة إلى الأبد. تقدّم هذه الأطروحة خوارزمية تقوم بشكل مسبق بحساب تقريبي لنواة البقاء يتميز بكونه متحفظّا وكذلك قادرا على التعامل مع الأوساط المتغيّرة كتلك التي تحوي عوائق متحركة. لتبيين مدى فائدة نواة البقاء المحسوبة مسبقا، في ما يتعلّق بضمان القيادة الآمنة، تمّ استعمالها ضمن خطة ملاحة تفاعلية أثبتت قدرتها على قيادة النظام الروبوتي في سيناريوهات مختلفة تتعلق بعدة قيود للحركة (تفادي التصادم، الرؤية، السرعة) بدون خرق القيود الخاصة بالحركة على الإطلاق.

الكلمات المفتاحية: القيادة الذاتية، السلامة المضمونة، تجنب الاصطدام، نظرية البقاء

ABSTRACT

Guaranteed motion safety is a substantial issue in many mobile robotics applications. This issue is usually tackled in the Inevitable Collision State (ICS) framework which is inherently limited to collision avoidance. This thesis explores the use of the more general Viability theory as an alternative when safe motion involves multiple motion constraints and not just collision avoidance. Central to Viability is the so-called *viability kernel*, *i.e.* the set of states of the robotic system for which there is at least one trajectory that satisfies the motion constraints forever. This thesis presents an algorithm that computes off-line an approximation of the viability kernel that is both conservative and able to handle time-varying constraints such as moving obstacles. To demonstrate the usefulness of the computed viability kernel with regard to motion safety, it has been used inside a basic on-line reactive navigation scheme that proved able to control the robot in several different scenarios involving multiple motion constraints (collision avoidance, visibility, velocity) without ever violating the motion safety constraints at hand.

Keywords: Autonomous Navigation, Provable Safety, Collision Avoidance, Viability Theory

RÉSUMÉ

La garantie de sûreté de mouvement est un problème important dans de nombreuses applications de robotique mobile. Ce problème est généralement abordé dans le cadre des États de Collision Inévitable ce qui est intrinsèquement limité à l'évitement des collisions. Cette thèse explore l'utilisation de la théorie plus générale de la Viabilité comme alternative lorsque la sûreté de mouvement implique des contraintes autres que l'évitement de collision. Le noyau de viabilité, *i.e.* l'ensemble des états du système robotique pour lesquels il existe au moins une trajectoire qui satisfait les contraintes de mouvement indéfiniment, est un élément central de la théorie de la viabilité. Cette thèse présente un algorithme qui calcule hors ligne une approximation du noyau de viabilité qui est à la fois conservative et capable de gérer des contraintes dynamiques telles que des obstacles mobiles. Pour démontrer l'utilité du noyau de viabilité quand à la garantie de sûreté de mouvement, il a été utilisé dans un schéma de navigation réactive en ligne qui s'est avéré capable de piloter le système robotique dans de différents scénarios impliquant plusieurs contraintes de mouvement (évitement de collision, visibilité, vitesse) sans jamais violer les contraintes de mouvement en vigueur.

Mots-clés: Navigation Autonome, Sûreté Garantie, Évitement de Collision, Théorie de Viabilité

ACKNOWLEDGMENTS

First and above all, thanks Allah for giving me the will and strength to finish this work. Then, I would like to thank my parents and my spouse for being an incredible support during this journey. I would like to thank my supervisors, Prof. Fezari and Prof. Fraichard, for their immense help and for putting up with my numerous shortcomings. I would like to thank the jury members, Prof. Doghmane, Prof. Moussaoui, Prof. Lakel, and Prof. Lafifi, for the time and effort they have put in reviewing my thesis and for their valuable suggestions to improve it. I would like to thank all my colleagues and friends for their never ending encouragement. I would like to thank all who were of any help little or big to me towards continuing until the end.

LIST OF FIGURES

ACRONYMS

CVA: Conservative Viability Algorithm

FOV: Field of View

ICS: Inevitable Collision States

RM: Regulation Map

ROS: Robot Operating System

VA: Viability Algorithm

VK: Viability Kernel

LIST OF PUBLICATIONS

1. M. Bouguerra, T. Fraichard, and M. Fezari. Safe motion using viability kernel. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, Seattle (US), May 2015

2. Mohamed Amine Bouguerra, Thierry Fraichard, and Mohamed Fezari. Viability-based guaranteed safe robot navigation. *Journal of Intelligent & Robotic Systems*, Nov 2018

TABLE OF CONTENTS

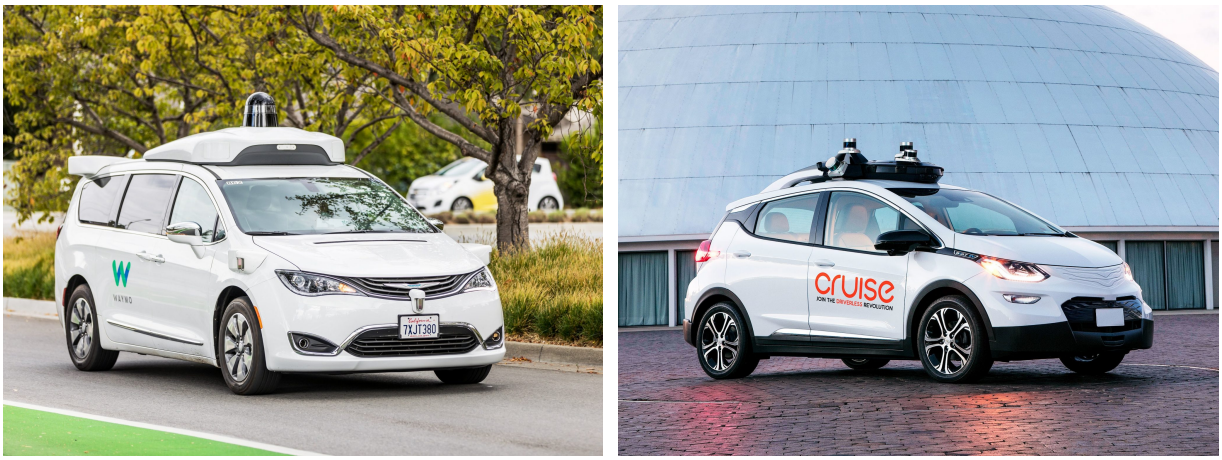
Chapter 1

INTRODUCTION

1.1 Context and Motivation

Mobile robotic systems are increasingly leaving the research laboratories to take part in our everyday lives. They come in a variety of types and can carry out so many different tasks. But before we can have them around, it is essential to assert that these robotic systems can *safely* navigate among the persons and objects that may populate their operational environment. Safe navigation refers to the ability of the mobile robotic system to go from one location to another while avoiding dangerous situations, such as collisions. Safe navigation grows more and more critical in applications where the size and dynamics of the mobile robotic system make it potentially harmful for itself and its environment.

One of the trending applications where motion safety plays a major role is the self-driving vehicles *i.e.* cars or trucks in which human drivers are not required to take control in order to safely operate the vehicle (*e.g.* Fig. 1.1). The amount of hype around self-driving vehicles is rightfully justified when we look at the numerous potential benefits that these machines can bring. First and foremost, they would substantially reduce the traffic accidents. According to the World Health Organization [38], every 24 seconds someone dies on the road, and by some estimates [50], more than 90% of all accidents are attributed to the driver's error such as inattention and poor



(a) Waymo (b) Cruise AV

Figure 1.1: Self-driving cars.

decision making. In addition to that, self-driving vehicles would improve access to mobility for the elderly and disabled, and also increase the efficiency of how people move about by reducing traffic congestion and allowing more free time during commuting. For several years, many manufacturers have been developing and testing their self-driving vehicles in public roads. These self-driving vehicles were reported to have driven an impressive number of kilometers[1]. On the not-so-bright side though, these tests have known several incidents of self-driving vehicle crashes over the past few years. The most notable incident is the Uber crash that led to a pedestrian fatality (Fig. 1.2). These incidents put in evidence a couple of important facts. First, despite the progress made in self-driving vehicles development, motion safety remains an open problem. Second, motion safety is a fundamental problem that, in many applications, can be life critical.

[1]In October 2018, Google self-driving cars (now Waymo) were reported to have logged over 16 million kilometers on public roads in the United States.

Figure 1.2: (Left) Location of the Uber crash, showing the paths of the pedestrian in orange and of the Uber test vehicle in green. (Right) Post-crash view of the Uber test vehicle, showing damage to the right front side.

1.2 Guaranteed Motion Safety

It is now clear that before we can deploy mobile robotic systems in different real-world applications, it is essential to assess their motion safety in a rigorous manner. The levels of motion safety that a robotic system can achieve must be explicitly stated as well as the operational conditions under which they can be guaranteed [20].

When studying motion safety, it is important to first distinguish between the various reasons that could potentially lead to an accident. The source of a crash can be mechanical, such as an incorrect assembly of a brake system, or electrical such as faulty internal wiring. It could be a failure of computing hardware chips, or due to errors or bugs in the autonomy software. Accidents might as well be caused by bad or noisy sensor data or inaccurate perception. And they can also happen due to incorrect planning or decision-making, inadvertently selecting hazardous actions.

In this thesis, we restrict ourselves to the latter case and look at the motion safety problem solely from a decision-making point of view.

Even if we do assume the robotic system is working perfectly from a hardware and software point of view, and has an accurate perception of its surrounding at all times, the problem of guaranteed motion safety remains quite challenging. This challenge stems from the fact that in most applications, mobile robotic systems are designed to operate in dynamic environments *i.e.* among moving objects such as human beings, vehicles or other robotic systems. In dynamic environments, the robotic system is required to account for the future behavior of the moving objects when deciding its course of action. Failure to do so yields navigation strategies whose motion safety is not guaranteed.

Accounting for the future behavior of the moving objects entails two issues. The first one concerns the determination of a model of their future behavior. In certain cases, this model may be available beforehand. In most cases however, it will be necessary to estimate this future behavior based on the information provided by its sensors. This thesis will suppose that this first issue is already solved, *i.e.* it is assumed that a model of the future has been determined. Thus the work will concentrate only in the second issue necessary to account for the moving objects. This second issue has to do with how to arrive to a decision that guarantees the motion safety of the system with respect to a given model of the future. This key question to guaranteed motion safety is usually tackled using the Inevitable Collision States (ICS) Framework [16].

1.2.1 Motion Safety from an ICS Perspective

An Inevitable Collision State for a given robotic system is a state for which, no matter what the future trajectory followed by the system is, a collision with an object eventually occurs. Imagine

the state of a robotic system with limited deceleration capabilities heading towards a nearby wall at a very high speed. Although not in collision at the present state, it has nothing to do in this case but to crash into the wall. For obvious safety reasons, the robotic system should never ever find itself in an ICS. The concept of ICS is well suited to address the issue of safe motion in dynamic environments since it takes into account both the dynamics of the robotic system and the future behavior of the moving objects. However, employing ICS presents its own challenges. First, the intrinsic complexity of their characterization must be worked out to determine if a state is an ICS or not. Once the safety verification of a given state can be performed, the next move is to employ that information in a collision avoidance scheme to keep the robotic system at hand safe from falling in an ICS.

1.2.2 Viability Theory as an Alternative

Although the perspective given by ICS provides an answer to the issue of guaranteed motion safety when it comes to avoiding collisions, there is often more to motion safety than mere collision avoidance. In many cases, the motion of the robotic system must satisfy various types of constraints in order to be considered safe. For instance, a legged robot should maintain its balance, the speed of an airplane should never go beyond its stalling speed, or a spy robot should stay out of sight of a patrol. No matter what the set of constraints that the robotic system ought to satisfy, they yield a set of forbidden states that the robot should avoid. The tenet of this thesis is that motion safety in general should receive an ICS-like treatment. In other words, the robotic system should of course avoid the states that violate the constraints, but more importantly, it should avoid the states inevitably leading to them.

To address this key question, this thesis considers the *Viability* framework which is more gen-

eral than the ICS framework. Viability theory [2] addresses the following question: how to control dynamical systems subject to *viability constraints*? Viability constraints define a subset of the state space of the system within which the system should remain. A *viable* state is guaranteed to have at least one sequence of controls which will keep the system within the viability constraint set indefinitely. Conversely, *nonviable* states are those where failure is no longer avoidable. Note that when collision avoidance is the only viability constraint, the nonviable states are Inevitable Collision States. The *viability kernel* of the viability constraints is the set of all its viable states. In this framework, the ability to design a control system for a robot that is able to compute its viability kernel and remain inside it at all times is also the key to guaranteed motion safety.

1.3 Contribution and Thesis Organization

The primary contribution of this thesis is the **Conservative Viability Algorithm (CVA)**. It consists of an adaptation of the viability kernel approximation algorithm presented in [44]. Our adaptation is designed to make the algorithm: (i) *conservative*, and (ii) able to handle *time-varying viability constraints* such as moving obstacles. These two features are essential to address the issue of guaranteed safe motion in dynamic environments. To showcase its versatility and its ability to address different kinds of viability problems, the algorithm proposed has been implemented and used to compute the viability kernel for the case of a double integrator robot in seven scenarios with mixed combinations of workspace (static, moving obstacles) and viability constraint types (collision avoidance, visibility, velocity). To demonstrate the usefulness of the viability kernel computed off-line by the Conservative Viability Algorithm, it is used on-line inside a basic and purely reactive navigation scheme that proved able to control the robot in the different scenarios without *ever* violating the viability constraints at hand. The viability kernel could just as well be used inside a motion planner in order to compute safe motions.

The rest of the thesis is organized as follows: Chapter 2 reviews the relevant literature concerning ICS characterization methods, with a discussion of their advantages and disadvantages as opposed to the proposed approach. Chapter 3 recalls the key concepts of the Viability theory, reviews the various viability kernel approximation methods, and finally presents in details the algorithm of [44]. Then, in Chapter 4 we give a discussion about the limitations of the original algorithm with regard to guaranteed motion safety, and describe how to transform it into a conservative one able to handle time-varying viability constraints. In Chapter 5, we present how to formulate robotics-related case studies with various motion safety constraints as a viability problem. The simulation results of the robotic scenarios demonstrating guaranteed safe navigation in different situations are presented and discussed in Chapter 6. Finally, Chapter 7 gives a summary of the thesis contribution and states the different research directions that can be pursued based on the results obtained in this thesis.

Chapter 2

STATE OF THE ART

2.1 Introduction

When it comes to avoiding collision, it is now understood that designing a control system for a mobile robot that is able to compute its inevitable collision states and stay away from them at all time is the key to guaranteed motion safety. The difficulty of this approach though lies in the very computation of the set of inevitable collision states. Basically, to determine whether or not a given state is an ICS, it is required to check for collision every possible future trajectory of infinite duration the robot might follow from said state. Problematically enough, the number of those trajectories is virtually infinite, which renders the task of computing the exact set of ICS intractable. This has led many research efforts to settle for computing approximations of the ICS set.

Although the details may vary, most of the proposed ICS approximation methods rely in essence on the same principle: among all the possible future trajectories of the robotic system, a subset of so called *evasive trajectories* is selected, and then a state is deemed ICS if none the evasive trajectories starting from it is collision-free. This results in an over-approximation of the ICS set which is conservative, but whose quality depends essentially on the choice of evasive trajectories. In some cases, those evasive trajectories are easily identifiable, and provide a reasonably good approximation of the ICS set. But when it is not the case, the approximation may get excessively conservative to the point where most to all states are labeled as ICS.

Below we give an overview of the different ICS approximation methods reported in the literature both in static and dynamic environments. Based on the choice of evasive trajectories, we present the levels of motion safety these methods can provide and the conditions under which it can be guaranteed. We also briefly mention, in the last section of this chapter, some related works that treat the problem of guaranteed motion safety using theoretical frameworks other than ICS.

2.2 Static Environments

In the case of static environments *i.e.* with no moving objects, the choice of the appropriate evasive trajectories is relatively easy. *Braking trajectories*, which drive the robotic system to stop moving as shortly as possible have been the go-to choice in several works [5, 32, 47]. In these works, a state from which the robotic system is able to completely stop before it collides with its surrounding is considered a *safe* state, otherwise it is labeled an ICS and thus should be avoided. This strategy has proved able to provide a reasonable conservative approximation of the ICS set (reasonable in the sense that it guarantees the motion safety of the robotic system but at the same time it does not prevent it from effectively navigating its environment).

Nevertheless, an issue naturally arises with this strategy whenever the robotic system at hand is not able to stop at all *e.g.* it is a plane. This issue has been addressed in [46, 6] using *loiter circles* instead of braking trajectories. This indeed represents an answer to this issue but so long as the robotic system is not operating in a workspace that features narrow corridors. In fact, the limited turning radius of the robotic system may prevent it from performing loiter circles if the corridor is too narrow. In that case, all the states inside the corridor would be labeled as ICS and thus make it impossible for the robotic system to navigate through them.

2.3 Dynamic Environments

With the presence of moving objects, guaranteed motion safety becomes even more complicated. First, it is required that the future behavior of the moving objects is fully known, which is hardly the case. Then even if we assume that the future model of the moving objects is available, it is not clear in general how to select the appropriate set of evasive trajectories so as to obtain a reasonable approximation of the ICS set. Based on the choice of evasive trajectories, we can distinguish three different approaches in the literature that address this issue. The first one strives to guarantee absolute motion safety *i.e.* no collision will ever take place, by putting some assumptions on the future behavior of the moving objects. The second one settles to guarantee a weaker form of motion safety *i.e.* some collisions might be allowed. While the third one aims to maximize the chance of surviving collisions but with no guarantees whatsoever.

2.3.1 Guaranteed Absolute Safety

The most notable work in this category is [32]. It proposes to use evasive trajectories that maintain zero relative velocity with the moving object *aka* imitating trajectories. This idea was implemented in a very efficient ICS checker that would tell whether a state is an ICS based on collision testing of the different imitating trajectories corresponding to the different moving objects. However, since the environment would contain multiple moving objects, one imitating trajectory for one object is not guaranteed to remain collision-free with respect to the other objects forever. So in order to guarantee absolute motion safety, it was necessary therein to assume that the workspace is bounded and each moving obstacle eventually leave the workspace. These two assumptions allow to limit the lookahead with which the collision checking is done, since at some point in the future the workspace becomes and remains either empty or constant.

2.3.2 Guaranteed Weaker Safety

Some methods acknowledged the difficulty to find evasive trajectories of infinite duration in dynamic environments, and settled to weaker guarantees. In [10], passive safety is introduced; it guarantees that if a collision would ever occur, the robot will be at rest. In this work, a state is considered *passively* safe if there is a braking trajectory that starts from it and is collision-free until the robotic system has stopped. An even stronger form of motion safety is the *friendly passive safety* [30]. It ensures that if a collision ever happens the robot will be at rest, and the obstacles could have avoided the collision if they wanted to. These forms of weaker motion safety are interesting and could be a reasonable goal to strive for whenever the moving objects are considered friendly and non oblivious such as moving people, animals or other robotic systems. In fact, passive safety has become a de facto safety level and has been incorporated in numerous works involving various robotics applications ranging from cars [41, 30] to helicopter [14], and from service robots [28] to bipeds [7] .

2.3.3 Non Guaranteed Safety

Other methods settle to even less, they aim to improve the chance of surviving collisions with no strict guarantees however. For instance, the authors in [21, 22] resolve to evasive trajectories that are guaranteed to be collision-free only up to a finite time, whereas in [12, 49] the authors suggest to use evasive trajectories which are collision-free with respect to one obstacle at a time, instead of considering them all at once. Doing so is supposed to enhance the overall safety of the robotic system, but it does not provide any guarantees.

2.4 Methods based on other frameworks

It is worth to mention that the problem of guaranteed motion safety of robotic systems has also been addressed using formal verification instead of the ICS framework. For instance, in [1] an approach for formally verifying the safety of automated vehicles is proposed. The verification is performed online by predicting the set of all possible occupancies of the automated vehicle and other traffic participants on the road using reachability analysis. Then, the safety is guaranteed with respect to the modeled uncertainties and behaviors if the occupancy of the automated vehicle does not intersect that of other traffic participants for all times. In [36], the authors use formal verification to address the problem of obstacle avoidance for ground robots. For instance, they analyse and formally verify safety with respect to static obstacles, and passive and friendly passive safety with respect to moving obstacles. In addition to that, they provide liveness properties which means that provable safety is flexible enough to let the robot navigate its environment. These results have been achieved by developing the corresponding hybrid system models and then using differential dynamic logic theorem-proving techniques to formally verify their correctness.

Chapter 3

VIABILITY THEORY

3.1 Introduction

Viability theory [2] is a mathematical framework featuring a set of techniques designed to address the problem of controlling dynamical systems whose state must evolve while satisfying a number of constraints at all times. Such systems can be found in many domains from biology to economics, from environmental sciences to financial markets, from control theory and robotics to cognitive sciences. It is, for instance, the case in economics when the system has to adapt to scarcity constraints, balances between supply and demand, and many other constraints. It is also the case in biology as with the concepts of "constance du milieu intérieur" and "homeostasis". And it is equally the case in control theory and, in particular, in robotics, when the state of the system must evolve while avoiding dangerous situations such as collision with obstacles.

The inputs of a typical viability problem consist of a dynamical system which is controlled by means of state-dependent controls, and the *viability constraints set* which is defined by a subset of the state space of the system within which the system should remain. Accordingly, 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 within the viability constraint set indefinitely (Fig. 3.1). 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 [18]. The *viability kernel* of the constraint set is the set of all its viable states. The aim of viability theory is to

first, compute the viability kernel of the dynamical system and then, to define the *regulation map* (feedbacks, closed-loop controls, etc) which provides to the controlled system, at each state, the subset of controls that will keep the system inside this viability kernel indefinitely. These concepts will be formalized in the following section.

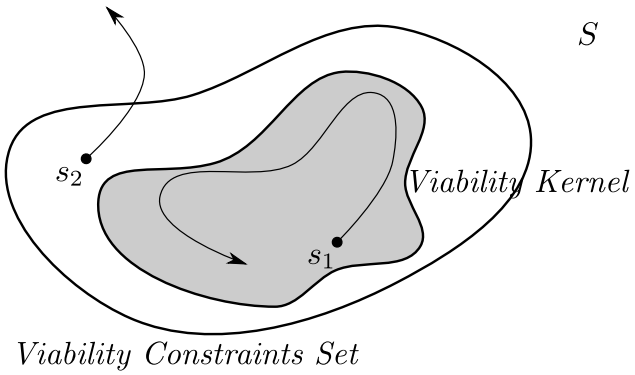


Figure 3.1: Main viability concepts: s_1 is viable, s_2 is nonviable.

3.2 Definitions and Notations

Let $\mathcal{A}$ denote a continuous-time dynamical system whose dynamics is described by differential equations of the form:

$$\dot{s}(t) = f(s(t), u(t)) \tag{3.1}$$

where $s(t) \in S$ is the state of $\mathcal{A}$ at time t. The state of $\mathcal{A}$ is influenced by a control $u(t) \in \mathcal{U}$ that can be state-dependent. S and $\mathcal{U}$ respectively denote the state space and the control space of $\mathcal{A}$. Viability constraints are characterized by the compact subset $\mathcal{K} \in S$ within which the system must be kept.

Let $\tilde{u} : [0, t_f] \longrightarrow \mathcal{U}$ denote a *control trajectory*, *i.e* a time-sequence of controls, t_f is the duration of $\tilde{u}$. The set of all possible control trajectories is denoted $\tilde{\mathcal{U}}$. Starting from an initial state

$s(t_0)$ at time t_0, a *state trajectory* $\tilde{s}(s(t_0), \tilde{u})$ is derived from a control trajectory $\tilde{u}$ by integrating (3.1). $\tilde{s}(s(t_0), \tilde{u}, t)$ denotes the state reached at time t. A state trajectory $\tilde{s}(s(t_0), \tilde{u})$ is said to be viable in $\mathcal{K}$ on an interval $[0, t_f], t_f \leq +\infty$, if $\forall t \in [0, t_f]$, $\tilde{s}(s(t_0), \tilde{u}, t) \in \mathcal{K}$. *Viable* states are those for which there exists at least one control trajectory $\tilde{u}$ yielding a state trajectory viable in $\mathcal{K}$ at all times *i.e.* on the interval $[0, +\infty)$.

The basic problem in the viability theory is to find the *viability kernel* of $\mathcal{K}$, *i.e.* the set of all its viable states:

Definition 1 (Viability Kernel)

$$\mathit{Viab}_f(\mathcal{K}) = \{s(t_0) \in \mathcal{K} \mid \exists\, \tilde{u} \in \tilde{\mathcal{U}} : \forall t \geq 0,\ \tilde{s}(s(t_0), \tilde{u}, t) \in \mathcal{K}\} \tag{3.2}$$

The viability kernel may be equal to the viability constraint set $\mathcal{K}$, in which case the viability constraint set $\mathcal{K}$ is called *viable* under the dynamical system f. The viability kernel may also be equal to the empty set, in which case the viability constraint set $\mathcal{K}$ is called a *repellor*, because all state trajectories starting from it eventually violate the constraints.

Once the viability kernel is determined, the next step is to compute the *regulation map* associated with it. The regulation map of the system is the set-valued map $s \in \text{Viab}_f(\mathcal{K}) \rightsquigarrow R(s) \subset \mathcal{U}$ that indicates at each state the set of *viable* controls that, when applied, will maintain the system inside the viability kernel.

Definition 2 (Regulation Map)

$$R(s_0) = \{u_0 \in \mathcal{U} \mid \exists\, \tilde{u} \in \tilde{\mathcal{U}} : \tilde{u}(0) = u_0 \textit{ and } \forall t \geq 0,\ \tilde{s}(s(t_0), \tilde{u}, t) \in \mathcal{K}\} \tag{3.3}$$

The regulation map represents a look-up table that serves as a guidance when controlling the system so as it never violates the viability constraints. If more than one viable control is available at

a certain state $s(t)$, the size of the subset $R(s(t))$ measures the *redundancy* which represents a guarantee of robustness for the system. Depending on the application at hand, the control selection would be made according to a certain optimization criterion.

The computation of the exact viability kernel of a dynamical system under viability constraints is in fact as challenging as computing the exact set of inevitable collision states, which led many research efforts to resolve to approximations as well. In the following section will we give an overview of the different viability kernel approximation methods reported in the literature, and we also provide examples of research works that have utilized the viability theory in the field of robotics.

3.3 Literature Review

Several methods have been proposed for the approximation of the viability kernel. For low dimensional systems with non linear dynamics, we can distinguish three types of methods. First, we have methods based on the discretization of the system such as the viability algorithm [44]. Second, there are the methods based on the viscosity solutions for Hamilton-Jacobi partial differential equations such as [34, 29]. Third and most recently, we can find methods based on interval analysis such as [37]. Now for systems whose dynamics can be described as polynomials, more efficient methods based on invariance sets have been proposed as in [48, 26]. For linear systems with higher dimensions however, Lagrangian techniques can be exploited as it has been done in [31].

Viability Theory has seen applications in several fields, including mobile robotics. In [7], the problem of a biped robot that has to maintain its balance while also ensuring passive safety has been addressed using Model Predictive Control. Closer to the work proposed in this thesis, a discrete method based on [44] was developed in [27] for the purpose of safe autonomous racing, *i.e.* to drive

as fast as possible around a predefined track. In [23] and [24], machine learning has been deployed to approximate the viability kernel for mobile robots. The purpose in [24] was to filter out unsafe states from the search space, to speed up motion planners, while in [23], it 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.

In the next section, we present in details the original viability algorithm from [44] upon which we build our proposed method.

3.4 Viability Algorithm

The viability algorithm from [44] operates in two stages. First, it approximates the original continuous problem by discretizating it in time and space. Then, it computes the exact viability kernel for the discretized problem in a recursive way.

3.4.1 Discretization

The first step is to discretize the system in time. There exist different methods to transform a continuous-time model into its discrete counterpart. For the sake of clarity, we consider the Euler explicit discrete scheme which is the most straightforward. Under this scheme, the discrete-time version of the dynamical system (3.1) is:

$$\begin{cases} s_{n+1} = g(s_n, u_n) = s_n + \rho f(s_n, u_n) \\ u_n \in \mathcal{U} \end{cases} \tag{3.4}$$

where ρ is the discrete time step. Afterwards, the state space of the system is discretized into a grid (regular or not). For clarity purposes, let us consider a regular grid of step d, denoted S_d. Note

however that the discrete system (3.4) cannot be defined on the grid S_d because nothing guarantees that, for all $s \in S_d$, the image $g(s,u)$ belongs to S_d. To address this issue, g^r is introduced, it is the extension of g with an hyperball of radius r:

$$g^r = g + \mathcal{V}(r) \tag{3.5}$$

where $\mathcal{V}(r)$ is the hyperball of radius r. r is chosen such that:

$$\forall s \in S_d \, , \; g^r(s,u) \cap S_d \neq \emptyset \tag{3.6}$$

An obvious choice is $r = d$. Finally, the control space must also be reduced to a finite subset denoted $\mathcal{U}_d$. As such, the following discrete and finite dynamical system is obtained:

$$\begin{cases} s_{n+1} \in g^r(s_n, u_n) = s_n + \rho f(s_n, u_n) + \mathcal{V}(d) \\ u_n \in \mathcal{U}_d \end{cases} \tag{3.7}$$

3.4.2 Computing the Discrete and Finite Viability Kernel

In the second step, the viability kernel of $\mathcal{K}_d = \mathcal{K} \cap S_d$ for the discrete and finite system (3.7) is computed as follows: $\mathcal{K}^0$ is initialized to $\mathcal{K}_d$, and the sequence of subsets $\mathcal{K}^1, \mathcal{K}^2, \mathcal{K}^3, ..., \mathcal{K}^n, ...$ is recursively defined such that:

$$\mathcal{K}^{n+1} = \{ s \in \mathcal{K}^n \mid \exists u \in \mathcal{U}_d \, : \, g^r(s,u) \cap \mathcal{K}^n \neq \emptyset \} \tag{3.8}$$

This will incrementally refine the grid $\mathcal{K}_d$ by discarding at each iteration the states from which the system will inevitably leave the grid in the next step. Let $\mathcal{K}^\infty = \bigcap_{n=0}^{\infty} \mathcal{K}^n$, it has been established in [44] that $\mathcal{K}^\infty$ is the largest subset of $\mathcal{K}_d$ such that:

$$\{ \forall s \in \mathcal{K}^\infty \, , \, \exists u \in \mathcal{U}_d \, : \, g^r(s,u) \in \mathcal{K}^\infty \} \tag{3.9}$$

or equivalently:

$$\mathcal{K}^{\infty} = \text{Viab}_{g^r}(\mathcal{K}_d) \tag{3.10}$$

and, since $\mathcal{K}_d$ is finite, there exists a finite integer p such that:

$$\forall n \geq p \, : \, \mathcal{K}^n = \mathcal{K}^p \tag{3.11}$$

which guarantees the convergence of the recursion. Thus the viability kernel of $\mathcal{K}_d$ for the discrete and finite system (3.7) can easily be computed in a finite number of steps using (3.8) (see Algorithm 1).

Algorithm 1: Viability Algorithm [44]

Input: Discrete-time dynamical system g^r; Discrete state space S_d; Discrete control space $\mathcal{U}_d$; Viability constraint set $\mathcal{K}$

Output: Viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$

$\mathcal{K}_d \leftarrow S_d \cap \mathcal{K}$;

$n \leftarrow 0$;

$\mathcal{K}^0 \leftarrow \mathcal{K}_d$;

repeat

  $\mathcal{K}^{n+1} \leftarrow \{s \in \mathcal{K}^n \mid \exists u \in \mathcal{U}_d(s) : g^r(s,u) \in \mathcal{K}^n\}$

  $n \leftarrow n + 1$

until $\mathcal{K}^n = \mathcal{K}^{n+1}$;

return $\mathcal{K}^n$

Once the viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ has been computed, it is straightforward to retrieve the discrete regulation map R_d which is defined for every state in $\text{Viab}_{g^r}(\mathcal{K}_d)$ as:

$$R_d(s) = \{u \in \mathcal{U}_d \mid g^r(s,u) \in \text{Viab}_{g^r}(\mathcal{K}_d)\} \tag{3.12}$$

This regulation map provides all the viable controls that are available at each state. Choosing controls belonging to R_d ensures that the system will remain in $\mathcal{K}_d$ at all times. It is important to note that, although the viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ is only an approximation of the viability kernel for the continuous problem $\text{Viab}_f(\mathcal{K})$, the authors in [44] proved and gave the conditions for which, the approximated kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ converges to the actual kernel $\text{Viab}_f(\mathcal{K})$ as d and ρ go to zero:

$$\lim_{d,\rho \to 0} \text{Viab}_{g^r}(\mathcal{K}_d) = \text{Viab}_f(\mathcal{K}) \tag{3.13}$$

The reader is referred to [44] for more details on the convergence issue, as well as the proof of the result stated in (3.10).

Over the following chapter, we discuss the limitations of the viability algorithm as presented in this chapter when it comes to guaranteed motion safety in dynamic environments. Based on this discussion, we present the necessary modifications that we propose in order to make the algorithm conservative and able to handle dynamic environments.

Chapter 4

PROPOSED METHOD

4.1 Conservative Viability Algorithm

Despite the convergence results mentioned in section 3.4, the fact remains that the viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ is only an approximation of the exact viability kernel $\text{Viab}_f(\mathcal{K})$. The main issue is that this approximation is not conservative, *i.e.* certain states will be labeled by Algorithm 1 as belonging to $\text{Viab}_{g^r}(\mathcal{K}_d)$ when, in truth, they do not belong to $\text{Viab}_f(\mathcal{K})$. The non conservative nature of $\text{Viab}_{g^r}(\mathcal{K}_d)$ is due to the various discretization assumptions, both in time and space, that have been made in order to obtain the finite and discrete dynamical system (3.7). From a viability point of view, it is critical to address these issues in order to obtain a conservative viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$. Below, we propose three design principles that address the various problematic aspects of the time and space discretization and thus aim to guarantee a conservative approximation of the viability kernel.

4.1.1 Exact Time Discretization

To begin with, the time discretization of the continuous dynamical system (3.1) into (3.4) using an approximate method such as the Euler explicit scheme yields discrepancies between $\tilde{s}(s_0, u, \rho)$ and $g(s_0, u)$ for a starting state s_0. Such discrepancies directly affect the resulting viability kernel in many ways. For instance, it may happen that the successor state $\tilde{s}(s_0, u, \rho)$ computed using the

approximate system (3.4) is inside the viability kernel while the actual successor $g(s_0, u)$ is not. In this case, the starting state s_0 will not be discarded by Algorithm 1 although it is clearly non viable. To avoid this issue, one must always resort to an exact time discretization of the system whenever possible.

4.1.2 In-between States Checking

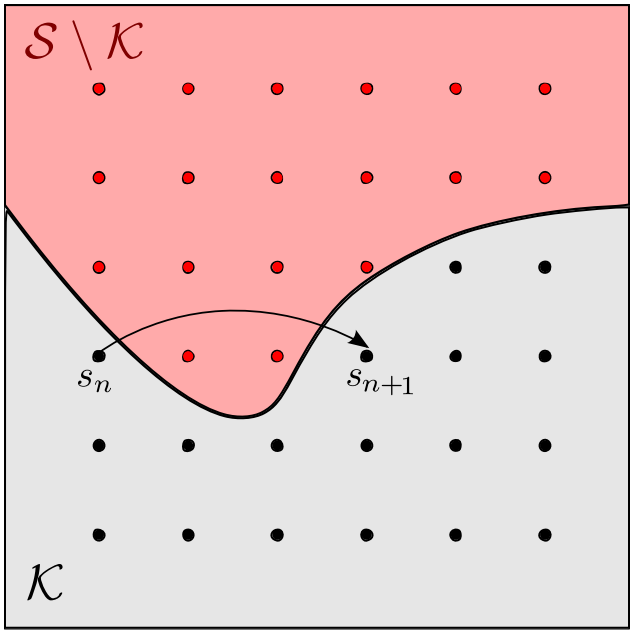


Figure 4.1: s_n and s_{n+1} are in $\mathcal{K}$ but the state trajectory in between is not.

The second issue also has to do with the time discretization of the system. It may happen that both s_n and its successor s_{n+1} belong to $\mathcal{K}_d$, but the state trajectory between them leaves $\mathcal{K}$ (Fig. 4.1). This would usually happen around small or narrow forbidden areas and when the time discretization step ρ is not small enough. Addressing this issue is relatively easy, it suffices to also check whether the state trajectory between s_n and s_{n+1} satisfies the viability constraints. This

would of course entails an additional computational burden but justifiably so since it will enforce the guaranteed motion safety that we strive for. Moreover, as we will discuss later, this should not be an issue since this computation is usually done off-line and only once.

4.1.3 Building a State Lattice

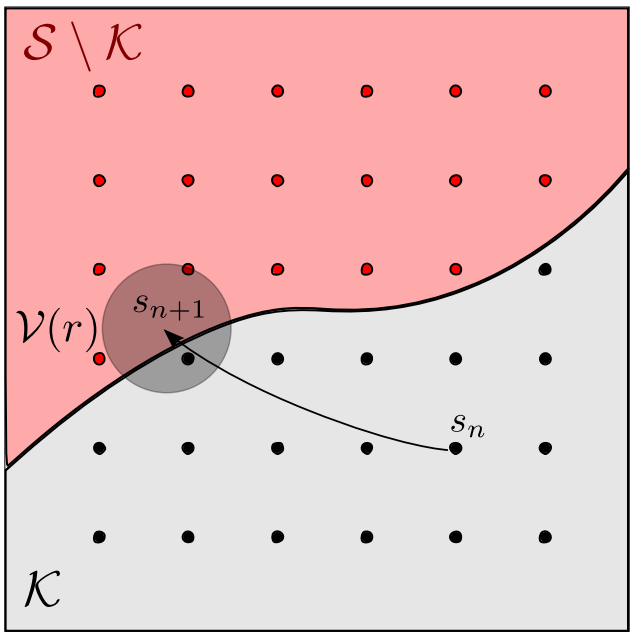


Figure 4.2: $s_{n+1} = g(s_n, u)$ is outside $\mathcal{K}$ but $g^r(s_n, u) \cap \mathcal{K}_d \neq \emptyset$.

Finally, recall from §3.4.1, the introduction of the hyperball $\mathcal{V}(r)$ to adapt (3.4) to the finite grid S_d. This yields two problems: the first one appears when a state s_n has a successor state $s_{n+1} = g(s_n, u)$ that does not belong to $\mathcal{K}$ but is such that $g^r(s_n, u) \cap \mathcal{K}_d \neq \emptyset$ (Fig. 4.2). In this case, although s_n is non viable, it will not be discarded by Algorithm 1. This problem is more prone to happen in proximity to the borders of the forbidden areas. The second problem stems from the fact that a state s_n does not have to reach another viable state s_{n+1} in order to classify

as viable, it just has to get *close* to it. This results in discontinuous state trajectories that might be viable by the algorithm standards but that the actual system may not be able to follow in practice (Fig. 4.3).

We address this issue through an appropriate state space discretization in such a way the need of the hyperball $\mathcal{V}(r)$ is relaxed. This is achieved by building a *state space lattice* based on the dynamical model of the system. A state space lattice is a prevalent structure in the field of robot motion planning [15, 40, 42]. It consists of a graph whose vertices represent a regular sampling of the state space and whose edges correspond to a carefully crafted set of controls (the case study presented in §5 details how the state space lattice is built for a double integrator system).

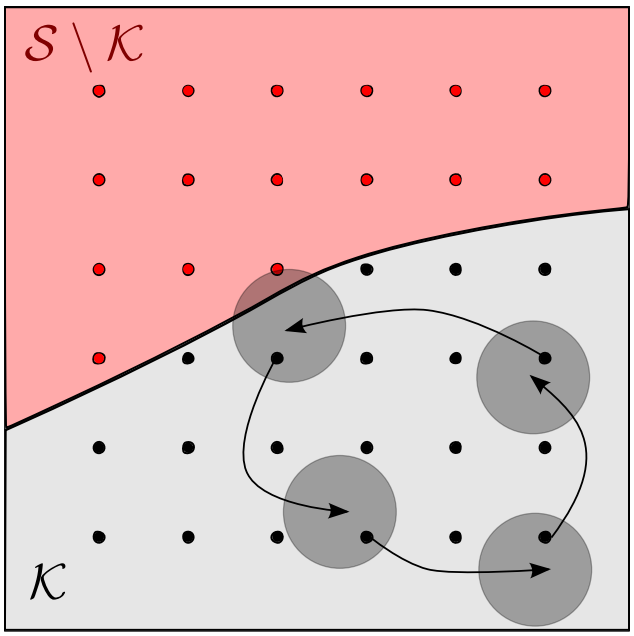


Figure 4.3: The state trajectory is viable by the algorithm standards but discontinuous in practice.

Now, when S_d is a state space lattice, it is not necessary to extend g with $\mathcal{V}(r)$ since, by construction, $\forall s \in S_d, g(s,u) \in S_d$, and a conservative viability kernel $\text{Viab}_g(\mathcal{K}_d)$ can actually be

computed using Algorithm 2, a slightly modified version of Algorithm 1.

Algorithm 2: Conservative Viability Algorithm

Input: Exact discrete-time dynamical system g; State space lattice S_d; Discrete control space $\mathcal{U}_d$; Viability constraint set $\mathcal{K}$

Output: Conservative viability kernel $\text{Viab}_g(\mathcal{K}_d)$

$\mathcal{K}_d \leftarrow S_d \cap \mathcal{K}$;

$n \leftarrow 0$;

$\mathcal{K}^0 \leftarrow \mathcal{K}_d$;

repeat

  $\mathcal{K}^{n+1} \leftarrow \{s \in \mathcal{K}^n \mid \exists u \in \mathcal{U}_d(s) : g(s,u) \in \mathcal{K}^n$ and trajectory from s to $g(s,u) \subset \mathcal{K}\}$

  $n \leftarrow n + 1$

until $\mathcal{K}^n = \mathcal{K}^{n+1}$;

return $\mathcal{K}^n$

4.2 Time-Varying Viability Constraints

Recall from 3.2 that the viability constraint set $\mathcal{K}$ has been defined as the compact subset of the state space S within which the dynamical system must remain. What happens now when the viability constraints are time-dependent *i.e.* they vary with time? It is the case for instance when viability is related to the collision avoidance of obstacles that are moving. The viability algorithm as described in 3.4 was designed and only applied to approximate the viability kernel when the the viability constraint set $\mathcal{K}$ is static. To cope with dynamic environments however, we suggest to cast the problem into the state-time space framework [16]. The state-time framework is in fact the key to deal with moving obstacles in robotics.

4.3 State-time Framework

In the state-time framework, we add the absolute notion of time as an extra dimension to the state space, and so the dynamical system (3.4) can be rewritten:

$$\begin{cases} (s_{n+1}, \tau_{n+1}) = h((s_n, \tau_n), u_n) = (g(s_n, u_n), \tau_n + \rho) \\ u_n \in \mathcal{U}_d \end{cases} \tag{4.1}$$

where τ denotes time. In this framework, it becomes possible to consider time-dependent viability constraints by defining $\mathcal{K}$ as the set of all the tuples (s, τ) that satisfy the viability constraints. Adapting Algorithm 2 so that it operates in state-time is straightforward. It suffices to define viability constraints by defining $\mathcal{K}$ as such, and to compute the viability kernel under the dynamical system (4.1) instead of (3.4).

One problem remains though, $\mathcal{K}$ has to be compact (recall that Algorithm 2 relies upon this assumption to converge). We can of course enforce this condition by upper-bounding the time dimension with some time horizon T_h which allows to obtain a compact viability constraint set $\mathcal{K}$. However, keeping in mind that a state is viable if it exists at least one sequence of controls that keeps the dynamical system in the viability constraint set *indefinitely*, it is obvious that, whatever the sequence of controls which is applied to the system from a given starting state-time, at some point in time, as soon as τ becomes greater than T_h, the state-time (s, τ) will leave $\mathcal{K}$ and the starting state-time will be considered as nonviable. In this situation, Algorithm 2 would always return an empty set.

The problem that we stated above would only confirm, yet from the different viability point of view, that the problem of guaranteeing absolute motion safety in dynamic environments is not solvable in the general case. Nevertheless, we have identified two interesting classes of situations where the nature of the time-dependent viability constraints are such that it becomes possible to

circumvent the problem stated above and to actually compute the viability kernel using Algorithm 2 as normal. These two classes of situations are respectively called *freezing* and *periodic*, they are presented in the next two sections.

4.3.1 Freezing Case

In this class of situation, it is assumed that the viability constraints stop varying at a given time T_f in the future. It is for instance the case where the moving obstacles will either stand still or leave the environment altogether after T_f. One feature of this class of situation is that after the time horizon T_f, all the (state, time) whose first component correspond to the same robotic system state are practically the same. In other words, the following expression holds:

$$\forall \tau > T_f \ : \ (s, \tau) = (s, T_f) \tag{4.2}$$

We can exploit this feature in order to circumvent the problem stated in 4.2 in two steps. The first step is to define the viability constraint set $\mathcal{K}$ as the set of all the tuples (s, τ) for which s satisfies the viability constraints and $\tau \leq T_f$, note that $\mathcal{K}$ is compact. The next step is to rewrite the dynamical system (4.1) as follows:

$$\begin{cases} (s_{n+1}, \tau_{n+1}) & = h((s_n, \tau_n), u_n) = (g(s_n, u_n), \tau_n + \rho) \text{ if } \tau_n < T_f \\ (s_{n+1}, \tau_{n+1}) & = h((s_n, \tau_n), u_n) = (g(s_n, u_n), \tau_n) \text{ if } \tau_n \geq T_f \\ u_n \in \mathcal{U}_d \end{cases} \tag{4.3}$$

Under (4.3), it can be noted that, whatever the sequence of controls which is applied to the system from a given starting state-time, the time component of the state-time of the system will never be greater than T_f. It therefore becomes possible to compute the viability kernel of $\mathcal{K}$ using Algorithm 2 normally.

4.3.2 Periodic Case

In this class of situation, it is assumed that the time-dependence of the viability constraints is periodic with a period T_p. This is the case for instance where the moving obstacles return to their initial state and repeat the same motion over and over again. This is often observed in practice where the workspace could consists of moving obstacles having a continuous periodic motion *e.g.* revolving doors, sliding doors, and elevators; or having discrete modes, changing from mode to mode in a periodic manner. We can notice that this class of situations also has an interesting feature regarding the (state, time) tuples whose first components correspond to the same robotic system state and the second components are separated by a time duration which is a multiple of the period T_p. These (state, time) tuples are practically the same. In other words, the following expression holds:

$$\forall \tau > T_p \, : \, (s, \tau) = (s, \tau \bmod T_p) \tag{4.4}$$

In this case, we can also exploit this feature to circumvent the problem stated in 4.2. To do so, the first step is once again to define the viability constraint set $\mathcal{K}$ as the set of all the tuples (s, τ) for which s satisfies the viability constraints and $\tau \leq T_p$. The next step is to rewrite the dynamical system (4.1) as follows:

$$\begin{cases} (s_{n+1}, \tau_{n+1}) = h((s_n, \tau_n), u_n) = (g(s_n, u_n), (\tau_n + \rho) \bmod T_p) \\ u_n \in \mathcal{U}_d \end{cases} \tag{4.5}$$

Under the system (4.5), as in the freezing case, whatever the sequence of controls which is applied to the system from a given starting state-time, the time component of the state-time of the system will never be greater than T_p, and it is possible to compute the viability kernel of $\mathcal{K}$ using Algorithm 2 as normal.

In the next chapter, we will bring our focus on the problem of robotics motion safety. For instance, we first present the robotic model that we will work with. Then, we give examples of several constraints that could be related to robotics motion safety, and show how we can express them as viability constraints.

Chapter 5

ROBOTICS CASE STUDIES

5.1 Introduction

Viability theory is a general theoretical framework that can be applied in different domains to tackle different viability problems. From this point on, we bring back our focus on the domain we are interested in to investigate how viability and the conservative viability algorithm can be used to solve the problem of guaranteed safe navigation of mobile robotic systems. In this chapter in particular, we begin by presenting the robotic system model that we will be working with in §5.2. This system model represents the first element that defines the viability problem. In the same context, we present the exact time discretization along with the state space lattice construction (which is essentially a state space discretization) of the system at hand. Recall from §4.1 that the exact time discretization and the state space lattice are required by the conservative viability algorithm in order to produce a conservative viability kernel that guarantees the safety of the robotic system. In §5.3, we present in detail various safety-related robotic motion constraints and how they can be expressed as viability constraints. These viability constraints represent the second element that defines a viability problem.

5.2 Robotic System Model

5.2.1 Robotic System

Let $\mathcal{A}$ denote a disk-shaped robotic system operating in a two-dimensional workspace $\mathcal{W}$. The dynamics of $\mathcal{A}$ correspond to a double integrator system whose acceleration a is directly controlled. A state s of $\mathcal{A}$ is represented by a tuple (p, v), where p is a 2D position, and v a Cartesian velocity. The motion of $\mathcal{A}$ is governed by the following differential equations:

$$\begin{cases} \dot{p} = v \\ \dot{v} = a \end{cases} \tag{5.1}$$

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

We consider this system model an interesting choice because of two reasons. First, it is a simple model for which it is possible to obtain an exact time-discretization and there exist a clean state space lattice solution. This simplicity serves the purpose of illustrating the application of the conservative viability algorithm fairly well. At the same time, this system model is realistic enough to validate the results obtained with the conservative viability algorithm. In fact, it is a second-order acceleration-controlled system which is more realistic than the first-order velocity-controlled systems that are sometimes used, *e.g.* the notorious "Dubins airplane" [13, 39]. However, keep in mind that the conservative viability algorithm remains applicable to any dynamical system provided that an exact time discretization is obtained and a solution to the state space lattice is found.

5.2.2 Exact Time Discretization

The first step of the conservative viability algorithm is to discretize the system in time and space. For a time step ρ, the following discrete state-transition equations are easily derived:

$$\begin{cases} p_{n+1} = p_n + v\rho + \frac{1}{2}a\rho^2 \\ v_{n+1} = v_n + a\rho \end{cases} \tag{5.2}$$

Eq. (5.2) is the exact discrete-time version of the dynamical system (5.1), the equivalent of (3.4), that the conservative version of the viability algorithm requires.

5.2.3 State Space Lattice

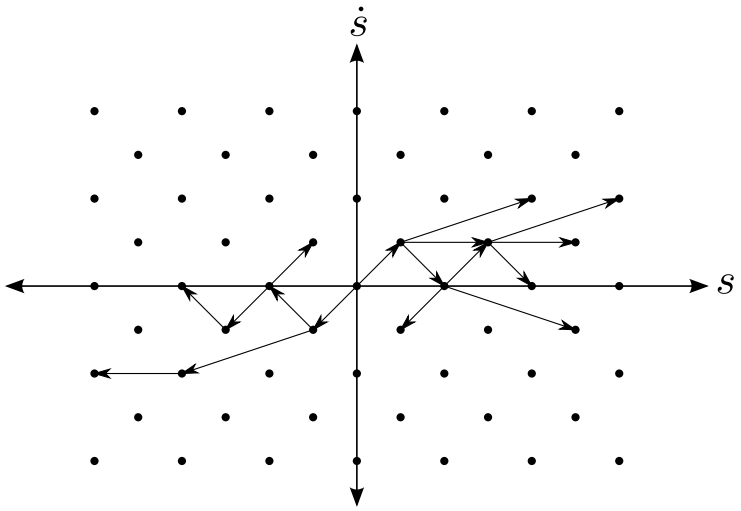


Figure 5.1: State space lattice for a 1D double integrator.

Recall from §4.1.3 that in order to obtain a conservative viability kernel that guarantees the motion safety of the robotic system, we suggested that we discretize the state space of the system

in such a way the need of the hyperball $\mathcal{V}(r)$ is relaxed. This is in effect achieved by constructing a *state space lattice* based on the dynamical model of the system.

For a fully-actuated system such as (5.2), a state space lattice can be built according to the method described in [15] which is outlined as follows. Let the set of possible controls $\mathcal{U}_d$ be restricted to $\{-a_{max}, 0, a_{max}\}$, and the time step ρ be chosen such that v_{max} is a multiple of $a_{max}\rho$. The term (u, ρ)-bang refers to applying a control $u \in \mathcal{U}_d$ for a duration ρ. Let $s_0 = (p_0, v_0)$ denote the *origin* state. The state space lattice S_d is the set of all states $s_i = (p_i, v_i)$ reachable from s_0 by a sequence of (u, ρ)-bangs. It is straightforward to establish that:

$$\begin{cases} p_i = p_0 + \frac{1}{2} m_i a_{max} \rho^2 \\ v_i = v_0 + n_i a_{max} \rho \end{cases} \tag{5.3}$$

where $m_i, n_i \in \mathbb{N}$. Thus, the state space lattice S_d is a regular grid which has a spacing of $a_{max}\rho^2$ in position and $a_{max}\rho$ in velocity. Note that the grid positions p_i for odd multiples of $a_{max}\rho$ are offset by $\frac{1}{2}a_{max}\rho^2$ from the grid positions for even multiples of $a_{max}\rho$ (see Fig. 5.1 for a one-dimensional system example). Note also that, in a space$\times$time perspective, the lattice S_d has a constant spacing ρ in the time dimension.

The method outlined above can be applied to construct a state space lattice for arbitrary fully-actuated robotic systems. The case of under-actuated robotic systems such as car-like vehicles is trickier to handle, however a number of solutions that could be used have been proposed, *e.g.* [33, 40, 42, 43, 51]. As soon as the state-lattice had been defined, Algorithm 2 can be applied.

5.3 Robotic Viability Constraints

The controlled dynamical system constitutes one of the inputs of a viability problems. The other input that defines a viability problem is the viability constraints set. One of the advantages of Via-

bility is its versatility and ability to handle different types of viability constraints. To demonstrate this versatility, various kinds of safety-related robotic motion constraints are considered. The first kind is standard collision avoidance (§5.3.1). The second kind has to do with visibility, it becomes relevant as soon as the robotic system at hand is engaged in pursuit-evasion missions (§5.3.2). The third and last kind arises when the robotic system is subject to certain restrictions on its velocity, *e.g.* an airplane robot whose speed is lower-bounded (§5.3.3). All of these robotic motion constraints are formally defined and expressed as viability constraints.

5.3.1 Collision Avoidance

Let us assume that the workspace $\mathcal{W}$ contains a set of b fixed and moving objects. Let $\mathcal{B}_i$ denote such an object, $\mathcal{B}_i(t)$ denotes the closed subset of $\mathcal{W}$ occupied by $\mathcal{B}_i$ at time t. Likewise, $\mathcal{B}_i([t_1, t_2])$ denotes the space$\times$time region occupied by $\mathcal{B}_i$ during the time interval $[t1, t2]$. Note that $\mathcal{B}_i = \mathcal{B}_i([0, \infty))$. Let $\mathcal{B}$ denote the union of the workspace objects (both in space and time):

$$\mathcal{B} = \bigcup_{i=1}^{b} \mathcal{B}_i = \bigcup_{i=1}^{b} \mathcal{B}_i([0, \infty)) = \bigcup_{i=1}^{b} \bigcup_{t \in [0,\infty)} \mathcal{B}_i(t) \tag{5.4}$$

In viability terms, the viability constraint set $\mathcal{K}_c$ within which the system $\mathcal{A}$ must be kept at all times is the set of states where the robotic system $\mathcal{A}$ is not in collision with any of the workspace obstacles:

$$\mathcal{K}_c = \{(s(t) \in S \mid \mathcal{A}(s(t)) \cap \mathcal{B}(t) = \emptyset\} \tag{5.5}$$

with $\mathcal{A}(s(t))$ denotes the closed subset of the workspace $\mathcal{W}$ occupied by $\mathcal{A}$ when it is in the state $s(t)$.

5.3.2 Visibility

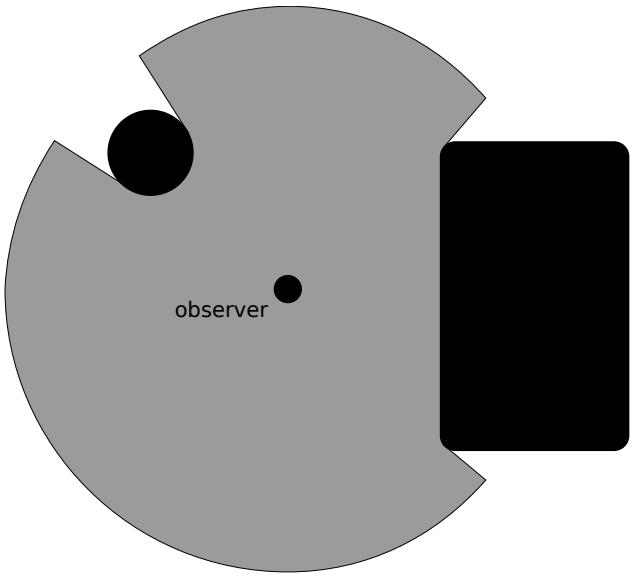


Figure 5.2: Field of view (gray area) for an observer among obstacles (black areas).

Constraints related to Visibility do arise whenever the robotic system at hand is engaged in pursuit-evasion missions. Such missions generally feature at least an observer and a target. The observer is equipped with sensors allowing it to acquire information about a limited region of the workspace $\mathcal{W}$. The region of $\mathcal{W}$ that is known by the observer at a given state $s(t)$ is called its field of view and is denoted $\text{FoV}(s(t))$. The shape this field of view $\text{FoV}(s(t))$ depends on the type of equipped sensors. The unknown regions of the workspace $\mathcal{W}$ are either out of the sensors' range or occluded by other workspace objects (Fig. 5.2).

The robotic system $\mathcal{A}$ can either endorse the role of the observer or the target. When the robotic system $\mathcal{A}$ is the observer, it means that it should always maintain one or more workspace targets $\mathcal{B}_i^t$ within its field of view at all times. The viability constraint set $\mathcal{K}_v$ corresponding to this scenario

is defined as follows:

$$\mathcal{K}_v = \{s(t) \in S \mid \bigwedge_i \text{FoV}(s(t)) \cap \mathcal{B}_i^t(t) \neq \emptyset\} \tag{5.6}$$

Now, when the robotic system $\mathcal{A}$ is the target, that means it should always stay out of the field of view of one or more workspace observers $\mathcal{B}_i^o$ at all times. The corresponding viability constraint set $\mathcal{K}_v$ is defined in this case as:

$$\mathcal{K}_v = \{s(t) \in S \mid \bigwedge_i \mathcal{A}(s(t)) \cap \text{FoV}(\mathcal{B}_i^o(t)) = \emptyset\} \tag{5.7}$$

Other variants of visibility-related viability problems could similarly be defined, *e.g.* the case where the robot is a target and should always stay inside the field of view of the workspace observers at all times.

5.3.3 Velocity

The aforementioned collision avoidance and visibility constraints have to do with the configuration of the robotic system $\mathcal{A}$ *i.e.* position and orientation. It is possible to have other robotic motion constraints that involve other components of the robotic system's state, such as the velocity. Examples of velocity-related constraints are numerous. One example is if the velocity of the robotic system $\mathcal{A}$ is lower bounded *i.e.* unstoppable, such as an airplane, or if the workspace $\mathcal{W}$ comprises regions that require upper-bounded speed, *e.g.* the roadway. Another interesting example of velocity-related constraints that also involve visibility is when a workspace observer can only sense moving targets. So if the robotic system is the target, standing still within the observer's field of view would be considered OK. All such constraints can readily be expressed via the definition of the corresponding viability constraint set.

5.3.4 Conclusion

The viability framework is most useful whenever the system is subject to various viability constraints since they can all be treated in a unified manner as a single viability problem. No matter how different in nature the various robotic motion constraints the robotic $\mathcal{A}$ is subject to, they can all be expressed under the form of different viability constraint sets $\mathcal{K}_i$. These viability constraints sets can then all be merged to form the final viability constraint set $\mathcal{K}$ that will feed Algorithm 2:

$$\mathcal{K} = \bigcap_i \mathcal{K}_i \tag{5.8}$$

Chapter 6

RESULTS AND ANALYSIS

6.1 Introduction

In this chapter, we evaluate the performance of our proposed method with regard to guaranteed motion safety in practice. We consider the robotic system model $\mathcal{A}$ that we presented in §5.2. This robotic system will have to undergo several safe navigation tests involving different workspace types (static, freezing, periodic), and different viability constraints (collision avoidance, visibility ,velocity). It led to the definition of seven scenarios with mixed combinations of workspace and viability constraint types. The seven scenarios are described in the subsequent sections where details about the workspace and the viability constraints are given. For each scenario, the Conservative Viability Algorithm 2 has been implemented to compute off-line the viability kernel (VK) and the regulation map (RM) for the robotic system. In a second stage, the computed viability kernel and the regulation map have been used within an on-line reactive navigation scheme that can drive the robotic system without ever violating the motion constraints at hand.

6.1.1 Computing VK and RM

To demonstrate its versatility and its ability to address different kinds of viability problems, the conservative viability algorithm 2 has been implemented for the case of the double integrator system and tailored to handle different scenarios. For each scenario, Algorithm 2 has been used to

compute the conservative and discrete viability kernel $\text{Viab}_g(\mathcal{K}_d)$ and the corresponding regulation map R_d. The computed regulation map R_d will then serve as a look-up table that indicates at each state the available controls that, when applied, will ensure that the system remains inside the viability kernel.

A straightforward analysis of Algorithm 2 shows that its time complexity depends on the size of the discrete set of states and the discrete set of controls (line 5). In other words, it grows exponentially with the dimensions of the state and the control spaces. In the current implementation (in Python on an average laptop), the running times for the different scenarios range from 6 to 20 minutes. It is not really a problem though since it should be kept in mind that the computation of $\text{Viab}_g(\mathcal{K}_d)$ and R_d is done *off-line* and *only once* for each scenario.

6.1.2 Safe Reactive Navigation

To demonstrate the usefulness of the computed $\text{Viab}_g(\mathcal{K}_d)$ and R_d, they have been used within an efficient on-line navigation scheme that is able to drive the system $\mathcal{A}$ around its workspace $\mathcal{W}$ while *always* respecting the different viability constraints at hand. The navigation scheme is rather simple and purely reactive: starting from an arbitrary state belonging to the viability kernel $\text{Viab}_g(\mathcal{K}_d)$, the navigation scheme selects, at each time step, the control to apply to $\mathcal{A}$ among the viable controls that are available at the current state. The set of viable controls is determined according to the regulation map R_d. The choice of the control depends on the task at hand and could be chosen randomly or so as to minimize a given cost function $\mathcal{C}$, *e.g.* distance to a goal. The control selection algorithm is outlined in Algorithm 3.

Simulations showing the navigation scheme at work have been carried out using ROS[1] and GAZEBO[2]. For illustration purposes, three snapshots at different times of a typical simulation run[3] are given. Each snapshot depicts the workspace (black regions are obstacles), the system's current position and the trail of its trajectory. The two-dimensional slice of the viability kernel corresponding to the current velocity of $\mathcal{A}$ is overlaid on the workspace: the viability kernel is shown in green and its complement in red. In the visibility-related scenarios, the gray areas correspond to the states where the visibility constraints do not hold because of the field of views. In all cases, the scenarios illustrate the ability of a purely reactive viability-based navigation scheme to control $\mathcal{A}$ forever without ever violating any of the viability constraints at hand.

Algorithm 3: Safe Reactive Navigation

Input: Current state s_0; Discrete regulation map R_d ; Cost function $\mathcal{C}$

Output: Next control u^*

1 $u^* \leftarrow \text{argmin}_{u \in R_d(s_0)} \mathcal{C}(g(s_0, u))$;

2 **return** u^*

6.2 Static Workspace

For this scenario, the workspace $\mathcal{W}$ contains static obstacles only (Fig. 6.1). To emulate an airplane, the velocity of $\mathcal{A}$ is lower bounded: $v > v_{min}$. Besides, the upper bound on $\mathcal{A}$'s acceleration $|a| \leq a_{max}$ and the width of $\mathcal{W}$'s corridors are such that it prevents $\mathcal{A}$ from flying in circles at any given position in $\mathcal{W}$. Thus, the viability constraints in this scenario are collision avoidance and velocity.

[1] `http://www.ros.org`

[2] `http://gazebosim.org`

[3] Full videos available at `http://thierry.fraichard.free.fr/research`

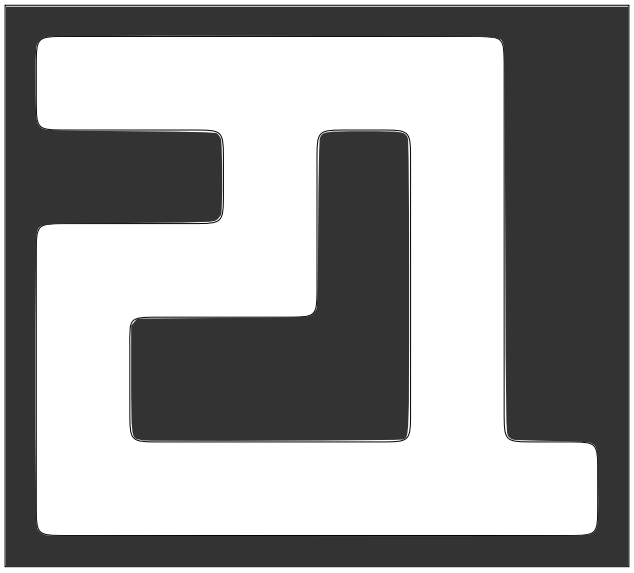
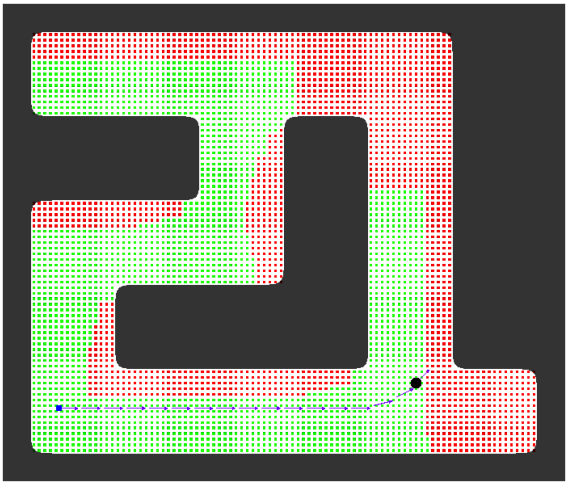


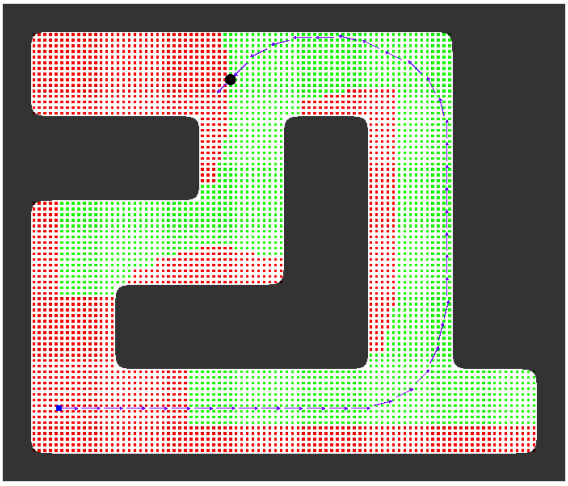
Figure 6.1: Test workspace for the plane scenario

This scenario represents an interesting case where neither the typical braking trajectories [5, 32, 47] nor the circling trajectories [46] are available.

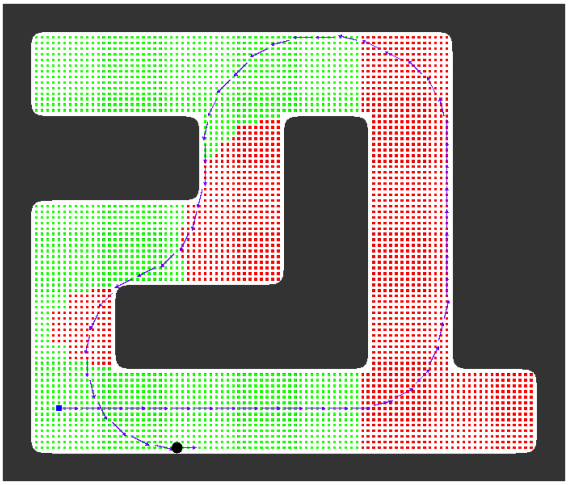
The robotic system starts at the bottom left of the workspace and at full speed. The control selection strategy for this scenario aims, at each time step, to minimize the change in the robot's velocity. This way the robotic system will try to keep moving in the same direction and the same speed as much as possible. The snapshots of Fig. 6.2 illustrate a typical simulation run. Although the reactive navigation scheme selects one action at a time, thanks to the underlying pre-computed VK and RM, the robot is able to perform farsighted maneuvers in order to preserve its long term viability. For example, it can be shown in Fig. 6.2a that it started to steer left well before reaching the dead-end at the bottom right.



(a) $\tau = 16, v_x = 6, v_y = 6$.



(b) $\tau = 39, v_x = -6, v_y = -6$.



(c) $\tau = 59, v_x = 8, v_y = 0$.

Figure 6.2: 2D viability kernel slices of the airplane scenario at different times.

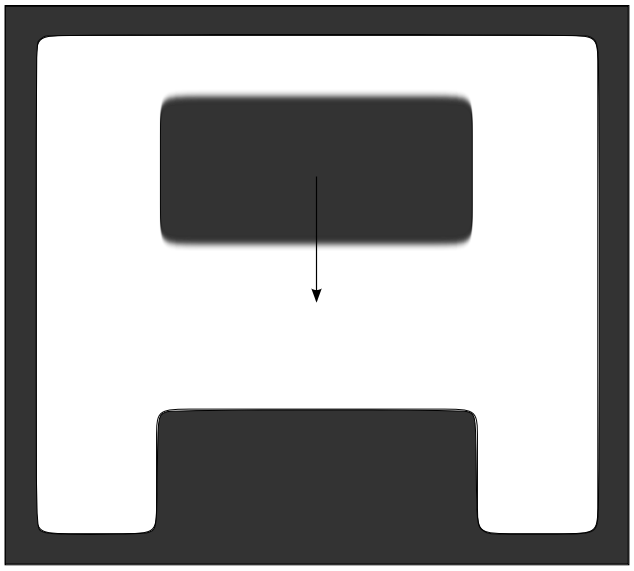
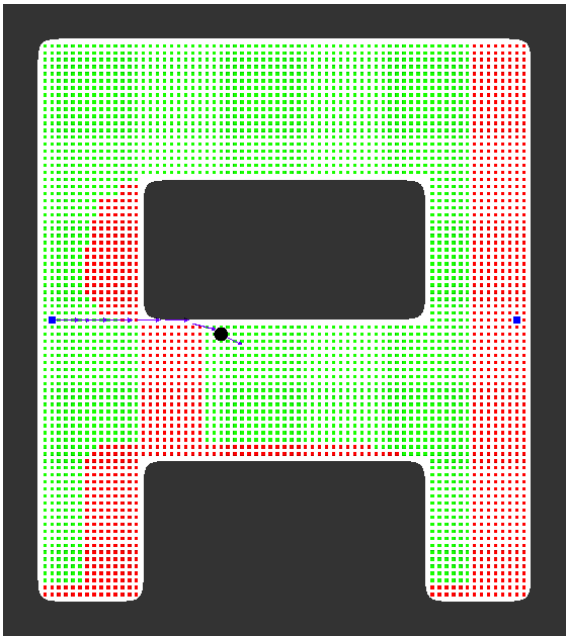


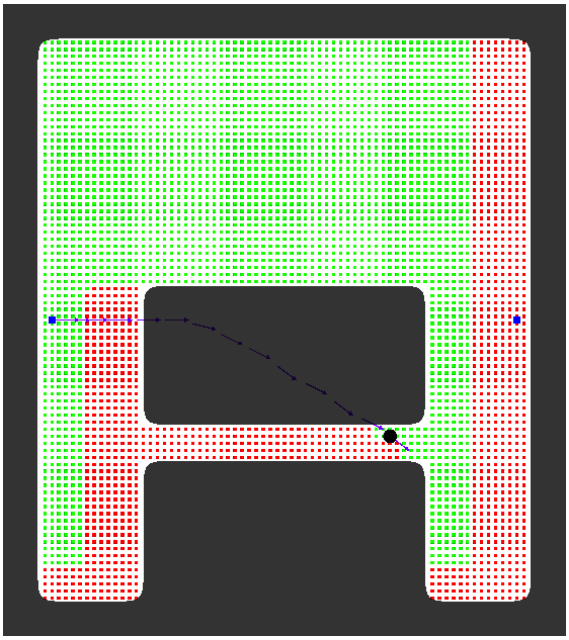
Figure 6.3: Test workspace for the compactor scenario

6.3 Freezing Workspace

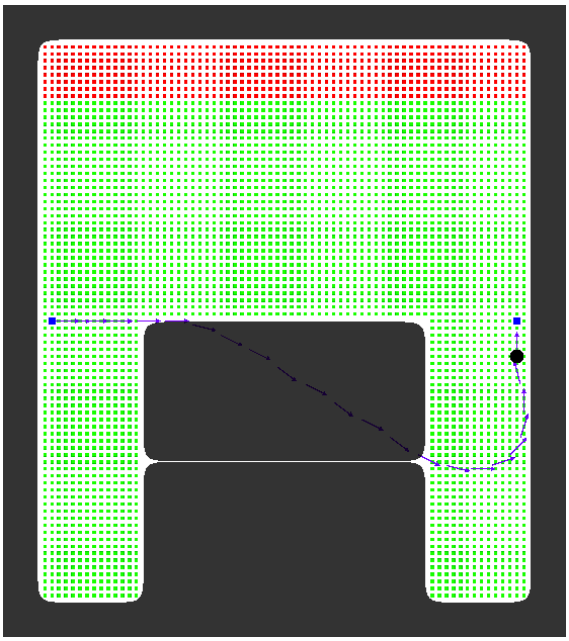
For this scenario, the workspace $\mathcal{W}$ contains one static obstacle region and one moving obstacle that moves downward until it makes contact with the static obstacle, a behavior resembling a trash compactor (Fig. 6.3). The viability constraints here are collision avoidance only. However, $\mathcal{A}$ has a goal now: it starts on the left side of the compactor and has to reach the right side. In this case, the choice of the control is goal-oriented, the navigation scheme selects the control that will drive $\mathcal{A}$ closer to its goal. This scenario is not as simple as it appears, it is similar to the one discussed in [19]. It is a case where the ICS-based approaches have a hard time finding the right set of evasive trajectories. For example, imitating trajectories [32] would have resulted in the whole region inside the compactor to be marked as ICS, which would have prevented the robot from passing it. Thanks to the pre-computed VK and RM, the robot was able to reach the goal *safely* using the viability-based reactive navigation strategy. The snapshots of Fig. 6.4 illustrate a typical simulation run.



(a) $\tau = 8, v_x = 8, v_y = -4$.



(b) $\tau = 14, v_x = 8, v_y = -6$.



(c) $\tau = 23, v_x = 0, v_y = 8$.

Figure 6.4: 2D viability kernel slices of the compactor scenario at different times.

6.4 Periodic Workspace

6.4.1 Collision Avoidance

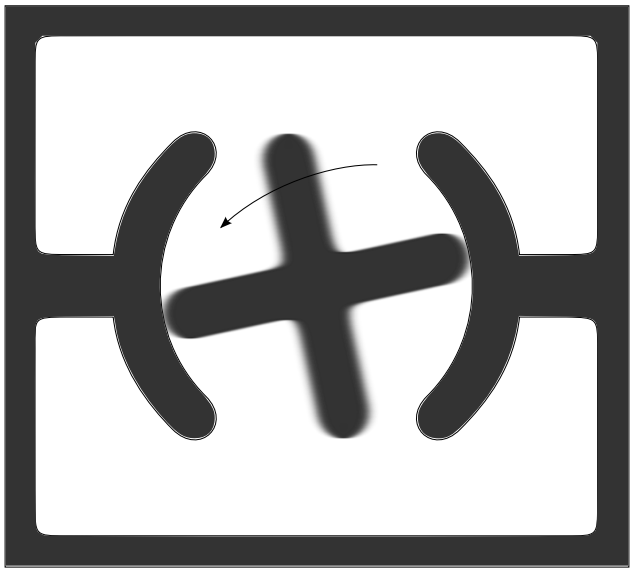
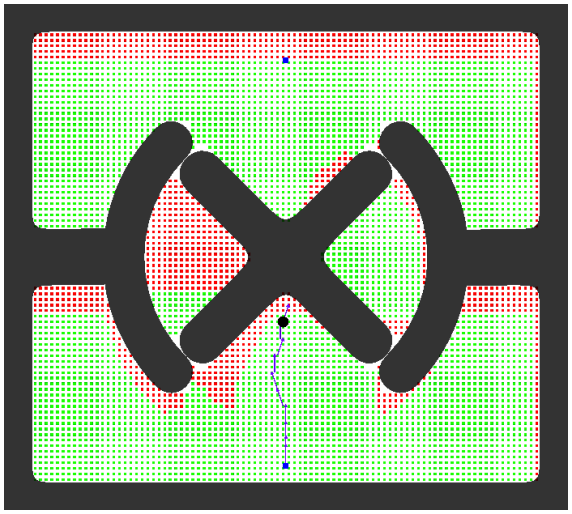
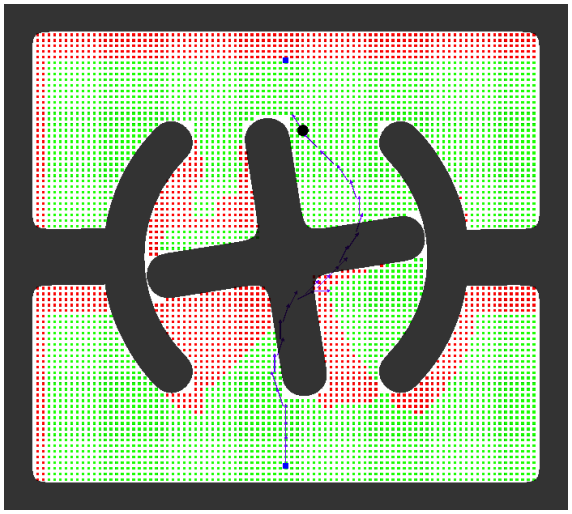


Figure 6.5: Test workspace for the revolving door scenario

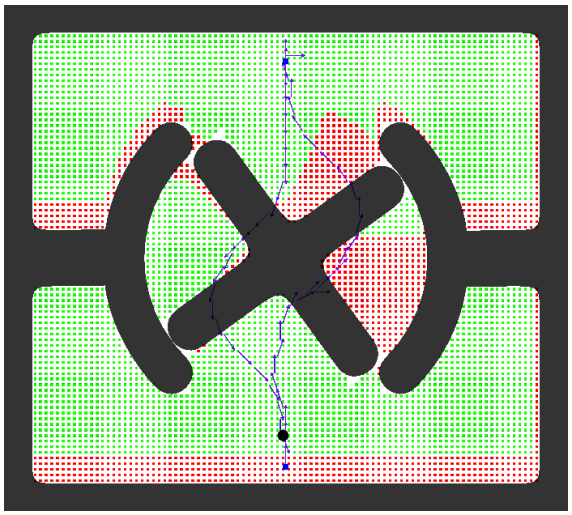
For this scenario, the workspace $\mathcal{W}$ contains both static and moving obstacles, it comprises two "rooms" and the only way to pass from one room to the other is to use a revolving door (Fig. 6.5). The revolving door has constant angular velocity and its behavior is periodic. The viability constraints here are collision avoidance only. Now, $\mathcal{A}$ has a task to accomplish which is to repeatedly pass from one room to the other. To that end, two goal positions are respectively defined in both rooms: when the current goal is reached, the other goal becomes the current goal and so forth. Using to the viability-based reactive navigation scheme, and a control selection strategy as simple as to minimize the distance to the goal, the robotic system was able to go back and forth many times without ever colliding with any of the workspace obstacles. In fact, it can provably continue doing so indefinitely. The snapshots of Fig. 6.6 illustrate a typical simulation run.



(a) $\tau = 10, v_x = 2, v_y = 6$.



(b) $\tau = 29, v_x = -4, v_y = 6$.



(c) $\tau = 59, v_x = 2, v_y = -6$.

Figure 6.6: 2D viability kernel slices of the revolving door scenario at different times.

6.4.2 Pursuit

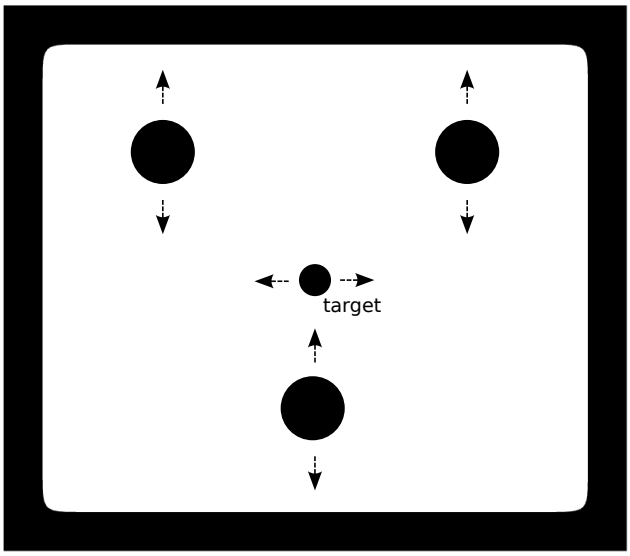
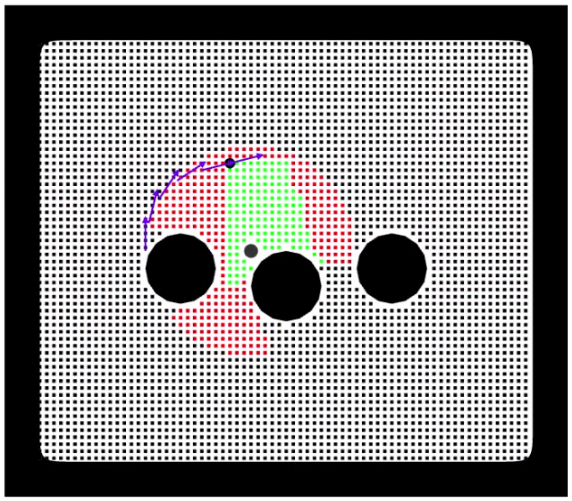


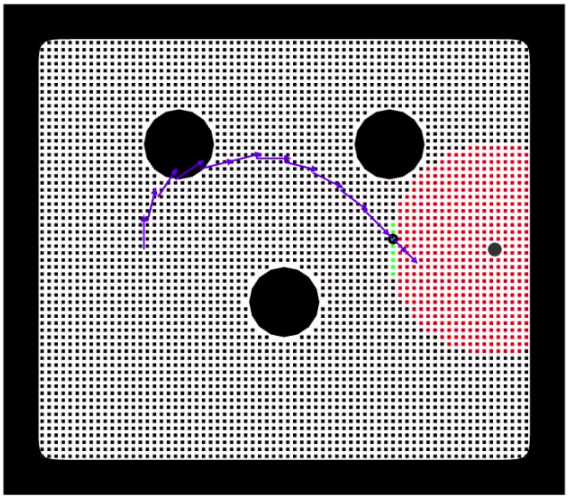
Figure 6.7: Test workspace for the pursuit scenario

For this scenario, the workspace $\mathcal{W}$ contains both static and moving obstacles. All moving obstacles are assumed to have a periodic behavior (Fig. 6.7). The system $\mathcal{A}$ is equipped with an omni-directional sensor and its task now is to keep one of the moving obstacles, the *target*, within its field of view at all times while avoiding collisions of course. The viability constraints here are collision avoidance and visibility. In this case, the navigation scheme was set to select the viable control maximizing the distance to the target. In a way, the robotic system is trying to be as less intrusive to the escorted target as possible. The snapshots of Fig. 6.8 illustrate a typical simulation run. It can be seen in Fig. 6.8b that the set of viable states can get very tight at times. This shows how challenging this scenario is and that there is so little room to error.

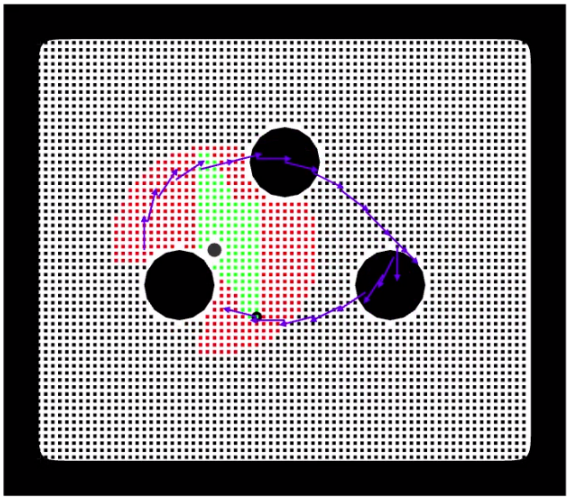
For an extra challenge, we have considered an additional constraint in the form of a lower-bound on the system's velocity. The corresponding results are depicted in Fig. 6.9.



(a) $\tau = 5, v_x = 8, v_y = 2$.

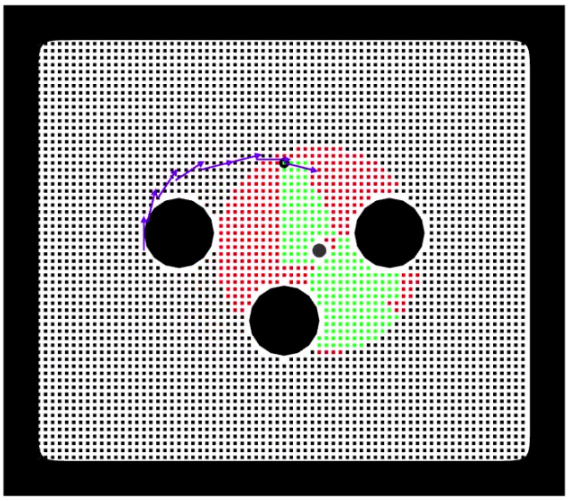


(b) $\tau = 12, v_x = 2, v_y = 2$.

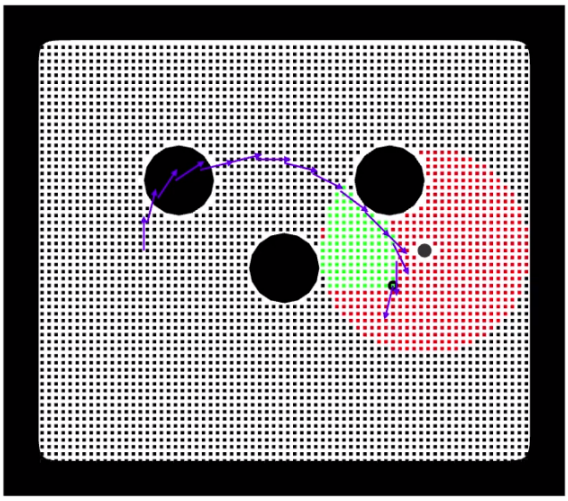


(c) $\tau = 20, v_x = 8, v_y = 2$.

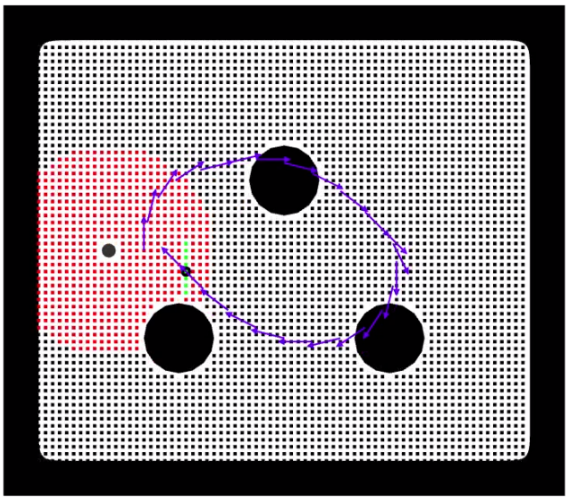
Figure 6.8: 2D viability kernel slices of the pursuit scenario at different times.



(a) $\tau = 7, v_x = 8, v_y = -2$.



(b) $\tau = 14, v_x = -2, v_y = -8$.



(c) $\tau = 23, v_x = -4, v_y = 4$.

Figure 6.9: 2D viability kernel slices of the unstoppable pursuit scenario at different times.

6.4.3 Evasion

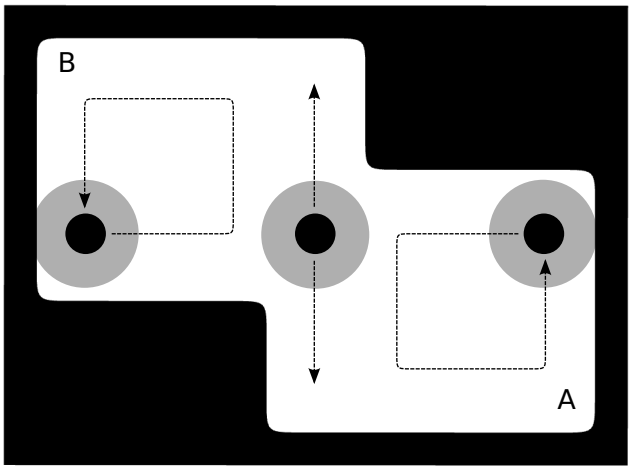
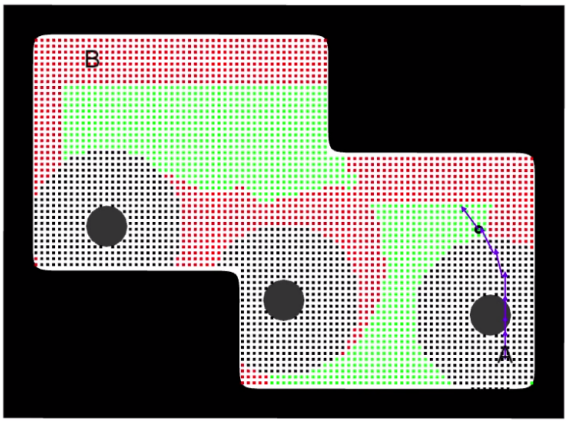


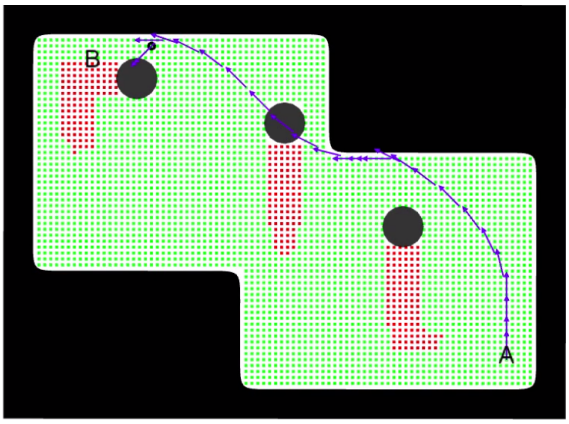
Figure 6.10: Test workspace for the evasion scenario

For this scenario, the workspace $\mathcal{W}$ contains both static and moving obstacles. The moving obstacles have a periodic behavior, they are assumed to be *sentinels* on patrol duty, they are equipped with omni-directional sensors with a limited field of view (Fig. 6.10). To make things more interesting, it is further assumed that the sensors can only detect moving objects. The system $\mathcal{A}$ is tasked to navigate from point A to point B and back without colliding with the workspace obstacles or being detected by the sentinels. The viability constraints in this case are collision avoidance, visibility, and velocity. The regulation map corresponding to this scenario allowed $\mathcal{A}$ to complete the task with success, even with a navigation policy as simple as choosing at each step the control that minimizes the distance to the goal. The snapshots of Fig. 6.11 illustrate a typical simulation run. Interestingly, note in 6.11b that the robot is taking advantage of the inability of the sentinel to detect moving objects by stopping until the path is clear.

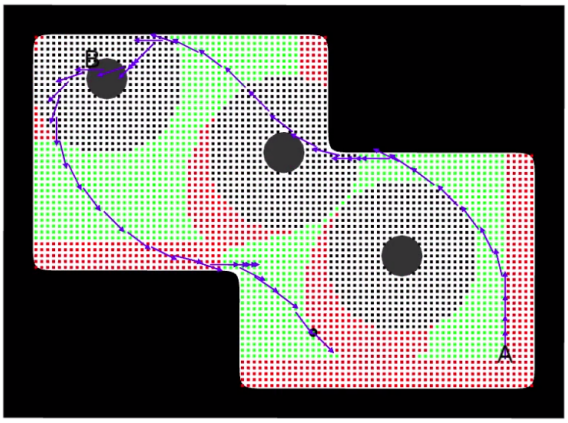
As with the pursuit scenario, the case where the velocity of $\mathcal{A}$ is lower-bounded has also been considered, the corresponding results are depicted in Fig. 6.12.



(a) $\tau = 6, v_x = -6, v_y = 8$.

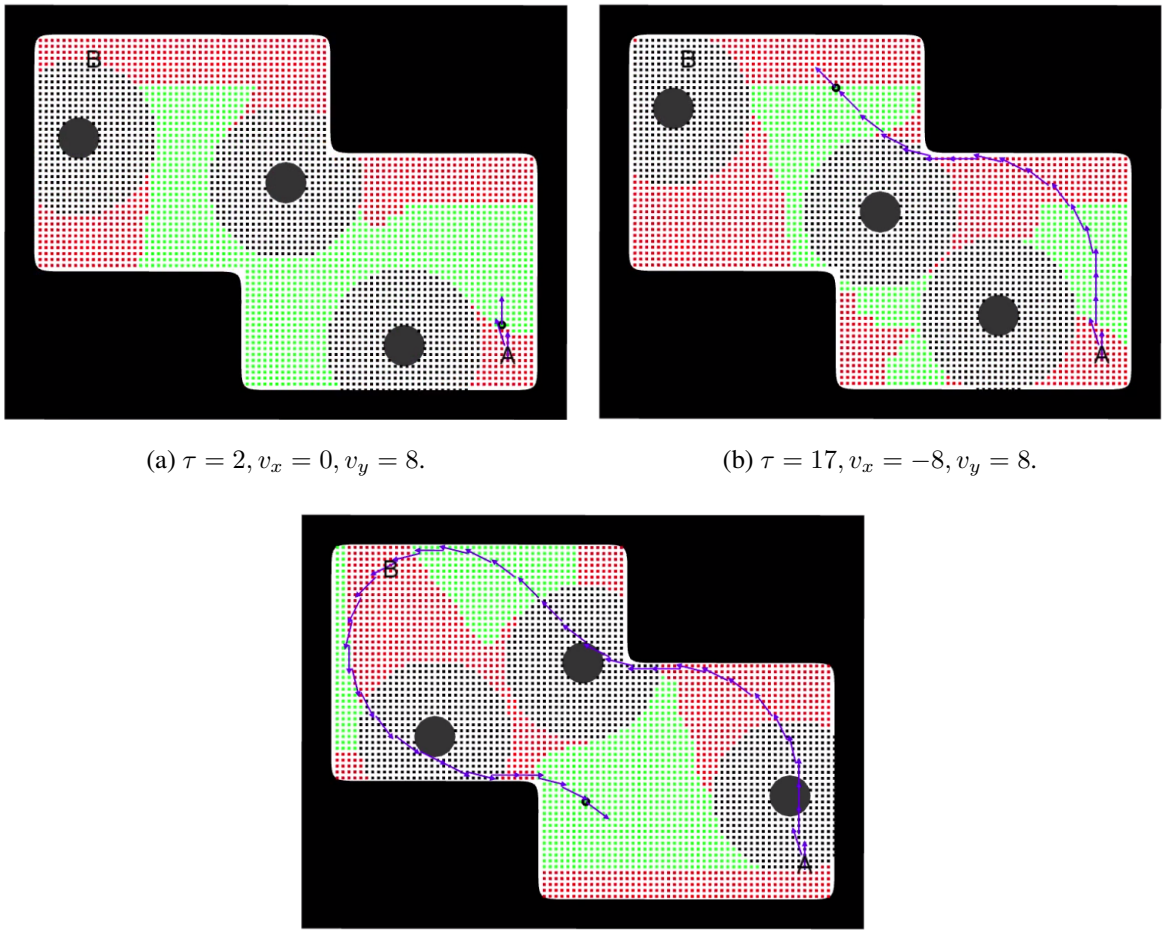


(b) $\tau = 30, v_x = 0, v_y = 0$.



(c) $\tau = 63, v_x = 6, v_y = -6$.

Figure 6.11: 2D viability kernel slices of the evasion scenario at different times.



(a) $\tau = 2, v_x = 0, v_y = 8$.

(b) $\tau = 17, v_x = -8, v_y = 8$.

(c) $\tau = 39, v_x = 8, v_y = -6$.

Figure 6.12: 2D viability kernel slices of the unstoppable evasion scenario at different times.

6.5 Discussion

We have seen, through the different scenarios presented above, the ability of a viability-based navigation scheme to achieve guaranteed motion safety for a robotic system under several constraints. These scenarios puts in evidence that viability framework is definitely an interesting alternative to ICS framework. In fact, they highlight the two key advantages of the conservative viability algorithm. The first one is its ability to handle different kinds of safety constraints in a unified manner (collision avoidance, visibility, velocity). The second one is that, unlike ICS-based methods, the quality of the conservative approximation is in no way dependent on the choice of an appropriate set of evasive trajectories. However, this feature comes at the cost of a greater computational complexity that prevents the on-line computation of the viability kernel and the regulation map. Depending on the task at hand, this may or may not be an issue. Note however that a more efficient implementation of the algorithm proposed could be obtained through parallel computing [11].

Chapter 7

CONCLUSIONS

7.1 Summary of the Contribution

Guaranteed motion safety is a substantial issue in many mobile robotics applications. This issue is usually tackled in the Inevitable Collision States (ICS) framework. In this framework, the key to guaranteed motion safety is to identify and avoid inevitable collision states at all times. In addition to the intricate identification of these states, the ICS framework is inherently limited to collision avoidance whereas the robot motion safety may often involve dealing with other types of constraints as well. This thesis has explored the use of the Viability framework to address the more general problem of guaranteed safe robot motion when it involves more than mere collision avoidance. The primary contribution of this thesis is the Conservative Viability Algorithm (CVA). It consists of an adaptation of the viability kernel approximation algorithm presented in [44]. This adaptation is designed to make the algorithm: (i) *conservative*, and (ii) able to handle *time-varying viability constraints* such as moving obstacles. These two features are essential to address the issue of guaranteed safe motion in dynamic environments.

The purpose of the CVA is to compute the Viability Kernel (VK) and the corresponding Regulation Map (RM) for a robot in a given scenario. CVA pre-processes the state space of the robot in order to compute VK+RM. Computing VK+RM is the key to the design of control schemes for which motion safety can be guaranteed. To that end, the robot must always remain within VK and this is easily achieved by selecting controls from RM so as to always remain inside VK and there-

fore always satisfy the motion safety constraints. Because of its complexity, CVA operates off-line but it is important to note that it is run only once. To demonstrate the usefulness of the VK+RM computed off-line by CVA, they are used inside a basic on-line reactive navigation scheme that proved able to control the robot in several different scenarios without ever violating the motion safety constraints at hand (collision, visibility, velocity). Note that the computed VK+RM could just as well be used inside more advanced navigation methods, *e.g.* with goal-reaching abilities.

7.2 Discussion

The Viability framework is definitely an interesting alternative to the ICS framework. The proposed conservative viability algorithm presents several advantages. First, it is able to handle different motion safety constraints in a unified manner. It has been shown for instance how a spy robot with a lower bound on its velocity can reach its goal while avoiding collision and staying out of the sight of a patrol at the same time. Second, unlike ICS-based methods, the quality of the viability kernel approximation is no way dependent on the choice of a set of evasive trajectories. This has enabled to obtain a practical solution to several safe motion cases where the set of appropriate evasive trajectories is not available. Third, the proposed algorithm is able to handle time-varying motion constraints such as moving obstacles. Although it is in general impossible to compute a non-empty viability kernel in the presence of moving obstacles, two classes of dynamic environments have been identified, freezing and periodic, for which it is possible to compute a valid viability kernel. Accordingly, it becomes possible to guarantee motion safety in the presence of moving obstacles for these two classes of environments. Although guaranteed collision avoidance has already been demonstrated for freezing environments using Inevitable Collision States, it is the first time that a similar result is achieved for periodic environments.

The above-mentioned features come at a cost though. The computational complexity of the

conservative viability algorithm is exponential in the size of the state and control spaces of the system. Depending on the task at hand, this may or may not be an issue. It should be kept in mind however that the computation of the viability kernel and the regulation map is done off-line and only once. They could then be used on-line within a reactive navigation scheme as well as a motion planner. Nevertheless, there is still room for improvement when it comes to efficient implementation of the algorithm especially through parallel computing [11].

7.3 Future Works

Based on the results obtained in this thesis, different research directions can be pursued. The efficiency issue might be worked out to obtain a faster implementation of the algorithm namely through parallel computing. One could also explore how the Conservative Viability Algorithm whose formulation is general can handle alternative robotic systems other than the fully-actuated double integrator. At a more fundamental level, further investigation into non-freezing and non-periodic environments should be carried out in order to determine whether viability can nonetheless be useful when it comes to guaranteed motion safety in arbitrary dynamic environments. Then, it would be of interest to investigate other potential connections between the viability theory and robots motion safety. For instance, the concepts of *invariance kernel* and *restoring viability* were subject to study in the viability theory, and it could be possible to benefit from the results obtained therein.

BIBLIOGRAPHY

[1] Matthias Althoff and John M Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

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

[3] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4), 1993.

[4] K. Bekris. Avoiding inevitable collision states: Safety and computational efficiency in replanning with sampling-based algorithms. In *Workshop on "Guaranteeing Safe Navigation in Dynamic Environments" in association with IEEE Int. Conf. on Robotics and Automation (ICRA)*, Anchorage (US), May 2010.

[5] K. Bekris and L. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, Roma (IT), 2007.

[6] Michael Blaich, Simon Weber, Johannes Reuter, and Axel Hahn. Motion safety for vessels: An approach based on inevitable collision states. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Hamburg (DE), October 2015.

[7] N. Bohórquez, A. Sherikov, D. Dimitrov, and P.-B. Wieber. Safe navigation strategies for a biped robot walking in a crowd. In *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, Cancun (MX), November 2016.

[8] M. Bouguerra, T. Fraichard, and M. Fezari. Safe motion using viability kernel. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, Seattle (US), May 2015.

[9] Mohamed Amine Bouguerra, Thierry Fraichard, and Mohamed Fezari. Viability-based guaranteed safe robot navigation. *Journal of Intelligent & Robotic Systems*, Nov 2018.

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

[11] Antoine Brias, Jean-Denis Mathias, and Guillaume Deffuant. Accelerating viability kernel computation with cuda architecture: application to bycatch fishery management. *Computational Management Science*, 13(3):371–391, Jul 2016.

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

[13] H. Chitsaz and S. M. LaValle. Time-optimal paths for a Dubins airplane. In *Proceedings IEEE Conference Decision and Control*, New Orleans, LA (US), December 2007.

[14] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. In *AHS 70th Annual Forum, Montreal, Quebec, Canada*, volume 1, pages 3–1, 2014.

[15] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5), 1993.

[16] T. Fraichard. Trajectory planning in a dynamic workspace: a state-time space approach. *Advanced Robotics*, 13(1), 1998.

[17] T. Fraichard. A short paper about motion safety. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, Roma (IT), April 2007.

[18] T. Fraichard and H. Asama. Inevitable collision states. a step towards safer robots? *Advanced Robotics*, 18(10), 2004.

[19] Th. Fraichard and T. Howard. Iterative motion planning and safety issue. In A. Eskandarian, editor, *Handbook of Intelligent Vehicles*. Springer, 2012.

[20] Thierry Fraichard. Will the driver seat ever be empty? Research Report RR-8493, INRIA, March 2014.

[21] E. Frazzoli, M. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1), 2002.

[22] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Int. Journal of Robotics Research (IJRR)*, 21(3), 2002.

[23] M. Kalisiak and M. van de Panne. Approximate safety enforcement using computed viability envelopes. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, New Orleans (US), April 2004.

[24] M. Kalisiak and M. van de Panne. Faster motion planning using learned local viability models. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, Roma (IT), April 2007.

[25] S. Kaynama, J. Maidens, M. Oishi, I. Mitchell, and G. Dumont. Computing the viability kernel using maximal reachable sets. In *ACM Int. Conf. on Hybrid Systems: Computation and Control*, Beijing (CN), April 2012.

[26] M. Korda, D. Henrion, and C. Jones. Convex computation of the maximum controlled invariant set for polynomial control systems. *SIAM Journal on Control and Optimization*, 52(5), 2014.

[27] A. Liniger and J. Lygeros. Real-time control for autonomous racing based on viability theory. *arXiv preprint arXiv:1701.08735*, 2017.

[28] Stefan B Liu, Hendrik Roehm, Christian Heinzemann, Ingo Lütkebohle, Jens Oehlerking, and Matthias Althoff. Provably safe motion of mobile robots in human environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1351–1357. IEEE, 2017.

[29] J. Lygeros. On reachability and minimum cost optimal control. *Automatica*, 40(6), 2004.

[30] K. Macek, D. Vasquez, T. Fraichard, and R. Siegwart. Towards safe vehicle navigation in dynamic urban scenarios. *Automatika*, 50(3-4), November 2009.

[31] J. Maidens, S. Kaynama, I. Mitchell, M. Oishi, and G. Dumont. Lagrangian methods for approximating the viability kernel in high-dimensional systems. *Automatica*, 49(7), July 2013.

[32] L. Martinez-Gomez and T. Fraichard. An efficient and generic 2d inevitable collision state-checker. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Nice (FR), September 2008.

[33] M. McNaughton, C. Urmson, J. Dolan, and J.-W. Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[34] I. Mitchell, A. Bayen, and C. Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Trans. on Automatic Control*, 50(7), 2005.

[35] S. Mitsch, K. Ghorbal, and A. Platzer. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In *Robotics: Science and Systems (RSS)*, 2013.

[36] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *The International Journal of Robotics Research*, 36(12):1312–1340, 2017.

[37] D. Monnet, J. Ninin, and L. Jaulin. Computing an inner and an outer approximation of the viability kernel. *Reliable Computing*, 22, 2016.

[38] World Health Organization. Global status report on road safety 2018. Geneva, 2018. Licence: CC BY-NC-SA 3.0 IGO.

[39] M. Owen, R. Beard, and T. McLain. Implementing dubins airplane paths on fixed-wing uavs. In *Handbook of Unmanned Aerial Vehicles*. Springer, 2014.

[40] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi. Motion planning through symbols and lattices. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, New Orleans (US), April 2004.

[41] Stéphane Petti and Thierry Fraichard. Safe motion planning in dynamic environments. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2215. IEEE, 2005.

[42] M. Pivtoraiko, R. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3), February 2009.

[43] M. Rufli and R. Siegwart. On the design of deformable input-/state- lattice graphs. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010.

[44] P. Saint-Pierre. Approximation of the viability kernel. *Applied Mathematics and Optimization*, 29(2), March 1994.

[45] G. Savino, F. Giovannini, M. Fitzharris, and M. Pierini. Inevitable collision states for motorcycle-to-car collision scenarios. *IEEE Trans. on Intelligent Transportation Systems*, 17(9), 2016.

[46] T. Schouwenaars, J. How, and E. Feron. Receding horizon path planning with implicit safety guarantees. In *American Control Conference*, Boston (US), Jun 2004.

[47] M Seder and I Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2007.

[48] Z. She and B. Xue. Computing an invariance kernel with target by computing lyapunov-like functions. *IET Control Theory and Applications*, 7(15), 2013.

[49] Z. Shiller, O. Gal, and A Raz. Adaptive time horizon for on-line avoidance in dynamic environments. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, San Francisco (US), September 2011.

[50] S. Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. (traffic safety facts crash•stats. report no. dot hs 812 506), Washington, DC: National Highway Traffic Safety Administration, March 2018.

[51] J. Ziegler and C. Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *IEEE/RSJ International Conference on Intelligent Robots and Sys- tems.*, 2009.