HAL
open science

Safe Navigation for Autonomous Vehicles in Dynamic Environments: an Inevitable Collision State (ICS) Perspective

Luis Martinez-Gomez

To cite this version:

Luis Martinez-Gomez. Safe Navigation for Autonomous Vehicles in Dynamic Environments: an Inevitable Collision State (ICS) Perspective. Automatic. Université de Grenoble, 2010. English. NNT : . tel-00600578

HAL Id: tel-00600578

https://theses.hal.science/tel-00600578v1

Submitted on 15 Jun 2011

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

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

Université de Grenoble

Thèse

Pour obtenir le grade de

Docteur de l'Université de Grenoble

Spécialité : Informatique

Arrêté ministériel : 7 août 2006

Présentée et soutenue publiquement par

Luis Alfredo Martínez Gómez

le 5 novembre 2010

Safe Navigation for Autonomous Vehicles in Dynamic Environments: an ICS Perspective

---

Thèse dirigée par Thierry Fraichard

Jury

M.	Augustin Lux	Président
M.	Rachid Alami	Rapporteur
M.	Luis Montano	Rapporteur
M.	Michel Parent	Examinateur
M.	Thierry Fraichard	Examinateur

Thèse préparée au sein du Laboratoire d'Informatique de Grenoble et l'INRIA Grenoble, dans l'École Doctorale de Mathématiques, Sciences et Technologies de l'Information, Informatique

Abstract

This thesis deals with the problem of safe navigation for autonomous vehicles in dynamic environments. Motion safety is defined by means of Inevitable Collision States (ICS). An ICS is a state for which, no matter what the future trajectory of the vehicle is, a collision eventually occurs. For obvious safety reasons, the autonomous system should never ever find itself in one of such states. To accomplish this objective the problem is addressed in two parts. The first part focus in determine which states are safe for the vehicle (non-ICS). The second part concentrates in how to select a valid control to move from one safe state to the other. Once its found, the vehicle can apply it to successfully navigate the environment. Simulations and experimental results are presented to validate the approach.

Résumé

Cette thèse traite le problème de navigation sûre pour les véhicules autonomes en environnement dynamique. La sûreté est définie par le concept des Etats de Collisions Inévitables (ICS). Un ICS est un état dans lequel, quelque soit le contrôle appliqué au système robotique étudié, celui-ci entre en collision avec un obstacle. Pour sa propre sécurité et celle de son environnement, il est impératif qu'un système robotique n'entre donc jamais dans un tel état. Ce problème est traité en deux parties. La première partie est consacrée à caractériser les etats de collisions inévitables. La deuxième partie à la création d'un système de navigation permetant d'éviter telles etats. Des résultats en simulation et sur une plateforme expérimentale sont présentés pour valider l'approche.

Contents

1 Introduction **11**
1.1 Solving motion safety difficulties with an ICS perspective . . . 14
1.2 Contribution . . . 15
1.3 Document organization . . . 16

2 Motion Safety- State of the Art **19**
2.1 Motion Safety Analysis . . . 20
2.2 Navigation Methods . . . 24
2.2.1 Deliberative Methods . . . 24
2.2.2 Reactive Methods . . . 28
2.2.3 Alternative Methods . . . 39
2.3 Conclusion . . . 41

3 Conceptual framework **43**
3.1 Notations . . . 43
3.2 ICS definition . . . 44
3.3 ICS and Motion Safety Criteria . . . 45
3.4 Motion Safety Definition . . . 46
3.5 Motion Safety Level Achievable by ICS . . . 48
3.6 ICS properties . . . 49
3.7 Conclusion . . . 52

4 Determining Safe States **53**
4.1 Preliminaries . . . 53
4.1.1 Evasive Manoeuvres . . . 53
4.1.2 Braking and Imitating Manoeuvres . . . 55
4.1.3 General ICS Checking Algorithm . . . 57
4.2 Ics-Check: a 2D ICS Checking Algorithm . . . 58
4.2.1 2D Reasoning . . . 58
4.2.2 Valid Lookahead . . . 60
4.2.3 Ics-Check Algorithm . . . 61

4.2.4 Complexity Analysis 61
4.3 Ics-Check: an Efficient Implementation 62
4.3.1 Exploiting Graphics Rendering Techniques 62
4.3.2 Precomputing As Much As Possible 64
4.4 Ics-Check At Work 64
4.4.1 Robotic Systems 64
4.4.2 Workspace Model 66
4.4.3 Car-Like System Case Study 67
4.4.4 Other Examples 71
4.4.5 Ics-Check Performances 72
4.5 Conclusion 72

5 Navigation 75
5.1 ICS-Avoid 76
5.1.1 Overview 76
5.1.2 Safe Control Kernel 77
5.1.3 Augmenting Ics-Check 78
5.1.4 Ics-Avoid Algorithm 79
5.1.5 Sampling strategies 80
5.2 Benchmarking Ics-Avoid with Other Navigation Methods . . 82
5.2.1 Simulation Setup 83
5.2.2 Conclusion Benchmarking 86
5.3 Conclusion 87

6 Experimental Results 89
6.1 Experimental Platform 90
6.1.1 Hardware 90
6.1.2 Software Architecture 91
6.2 Navigation Components 93
6.2.1 Organization of Components 93
6.2.2 Robot System 94
6.2.3 Mapping 95
6.2.4 Localization 95
6.2.5 Detection and Tracking of Moving objects 96
6.2.6 Model of the Future 96
6.2.7 ICS-Check 96
6.2.8 ICS-Avoid 97
6.3 Evaluation 97
6.3.1 Experimental Conditions 97
6.3.2 Collision With an Object 99
6.4 Analysis 102

7 Conclusions **105**
7.1 Contributions 106
7.2 Discussion 107
7.3 Future Work 107

Chapter 1

Introduction

If we take a moment and look around us, we won't find ourselves in one of those old movies' scenes where robots are everywhere, helping humans in all sort of tasks. For something like that happens, it will be necessary first to overcome some challenges. Issues that have prevented us to confront our robots with the real world, away from the ideal conditions found in the comfort of our labs. One such challenge is giving our robots the ability to safely navigate among the persons and objects that populate the environments they will be confronted with. Safe navigation means the ability to go from one location to another while avoiding dangerous situations, such as collisions. The challenge in all this can easily be verified if we take another moment and look once more around us. We will observe that some objects and humans don't remain static, all the contrary, they move. That is to say, we are dealing with a dynamic environment.

Robot navigation in dynamic environments has been studied extensively by the robotics community and some of their results were illustrated rather brilliantly by the 2007 DARPA Urban Challenge[1]. The challenge called for autonomous car-like vehicles to drive 96 kilometers through an urban environment amidst other vehicles. Six autonomous vehicles finished successfully the race thus proving that autonomous urban driving could become a reality. Note however that, despite their strengths, the Urban Challenge vehicles have not yet met the challenge of fully autonomous urban driving (how about handling traffic lights or pedestrians for instance?). Moreover, several collisions took place in the competition (Fig. 1.1). The accidents put in evidence that *motion safety* (the ability for an autonomous robotic system to avoid collision with the objects of its environment) remains an open problem in mobile robotics.

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

(a) Talos (MIT) & Skynet (Cornell) (b) Terramax (Oshkosh)

Figure 1.1: Collisions in DARPA Urban Challenge.

Motion safety is a basic requirement for any mobile robot application. However it takes a major role when dealing with those applications where human lives are susceptible of risk. Whenever the robots' size and dynamics are considerable special attention must be made to guarantee the robot will not harm the people around it.

One example of such applications is **driverless vehicles** (Figure 1.2), *i.e.,* an autonomous vehicle that can drive itself without the assistance of a human driver. The main idea behind automated driving is to take the persons out of the vehicle's control loop. The reason is that a great percentage of traffic accidents (around 90%) are attributable to human mistakes: errors in judgment of the road and car conditions, inattentiveness or simply taking the wrong action [RAH+09]. Besides the human losses, medical costs and property damage, accidents have also a negative impact in the environment. They are one of the main reasons of traffic congestion and its consequent increment in wasted fuel and air pollution. Driverless cars have the potential to transform the transportation industry by eliminating the accidents and drastically reducing their adverse effects. The first steps to achieve this goal have already been taken. Research projects funded by government agencies (HAVEit [HAK+08], SPARC [HBK+05], etc.) and the technology developed lately in the automotive industry (radar-based cruise control, lane-change warning devices, precrash systems, etc.) are a clear indication of this trend. However, as was evidenced by the DARPA Urban Challenge, there are still significant challenges to meet. Notably, guaranteeing the motion safety of the vehicle.

(a) INRIA Cycab (b) CMU Chevy Boss

Figure 1.2: Driverless vehicles.

Service robotics is an other example of the applications where motion safety is critical. As its name suggest, a service robot operates autonomously to perform tasks or services which are useful to the well-being of human beings. They are increasingly becoming a possible answer to solve some of the challenges posed by the demographic changes seen in industrialized countries. These changes have altered the balance of age groups causing a significant increase in the elderly population of the concerned nations. This means that a diminishing younger population is now faced with the challenge of providing solutions for the needs of its older fellow citizens. Needs that ranges from basic household tasks, such as cleaning, to more sophisticated ones, like filling the fridge with groceries (Fig. 1.3a) or assisting persons in their mobility. One example of the type of technology that provides this type of assistance is the autonomous wheelchair (Figure 1.3b). These devices increase considerably the range of action of a classic wheelchair. They improve the independence, convenience and mobility freedom of their users. They are also designed to respond to different levels of disabilities by adjusting the amount of user involvement in controlling the device. If required, they can operate in full autonomous mode, taking their users from one place to another through different kinds of settings (indoors or outdoors) in challenging dynamic environments as those found in homes or public spaces. Areas where the robotic devices will need to interact constantly with humans. As opposed to the small robots found nowdays in homes (*e.g.,* vacuum cleaner robots) the next generation of service robots will need to address more seriously the motion safety of their actions to guarantee that their movements will not cause harm. If a small vacuum cleaner robot collides with a piece of furniture is not big deal; if an autonomous wheelchair carrying a person collides and someone get hurt is quite a different matter.

(a) Mobile Manipulator (b) Autonomous Wheelchair.

Figure 1.3: Service Robots.

1.1 Solving Motion Safety Difficulties with an Inevitable Collision State Perspective

Dynamic environments are challenging. Specially when it comes to solve the difficulties associated with the motion safety of an autonomous robotic system. The complexity stem from one inherent characteristic of these changing environments: *time*.

To begin with, there is a real-time decision constraint. A robotic system cannot safely remain passive in a dynamic environment as it risks to be collided by a moving object. It has only a limited amount of time to come up with a decision that allows it to avoid a possible collision. If it takes too much time to make the decision it may find itself in a situation which can be catastrophic for its own safety.

Furthermore, avoiding collisions in dynamic environments requires to explicitly reason about the future. This allows to account for other time-dependent constraints such as the robot system's dynamics and moving objects future behaviour. Failure to do so yields navigation strategies whose motion safety is not guaranteed (in the sense that situations where collisions will eventually occur can happen). In general, the system's dynamics are usually known a priori and thus they can be used to reason about the

robot future behaviour because they allow to accurately predict its future states. Now, to account for the moving objects two issues arise. The first one concerns the determination of a description or model of their future behaviour. In certain cases, this knowledge may be available beforehand, *e.g.,* space applications. In most cases however, it will be necessary to estimate this future behaviour (in a deterministic or probabilistic manner) using whatever information available, typically sensor data. 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 is once a model of the future is available how to reason about it in order to produce safe navigation strategies. In essence, the process to arrive to a decision that guarantee the motion safety of the system with respect to a given model of the future. This thesis propose that this decision is taken by reasoning with the perspective given by the *Inevitable Collision States* (ICS) concept.

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. They are particularly well suited for navigation in dynamic environments since ICS take into account both the dynamic constraints of the robotic system and the future behaviour of the moving objects. Through the perspective given by ICS, the motion safety difficulties can be addressed in an incremental way. Starting from a theoretical basis to formally define what motion safety is in the context of dynamic environments to the implementation of the tools needed in a safe navigation strategy (if a robotic system doesn't wants to be involved in a collision it should never ever end up in an ICS). 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 Contribution

The contribution of the thesis is three-fold:

1. It explores key issues that have an impact in the motion safety of robotic systems operating in dynamic environments.

2. It furthers the study of the Inevitable Collision States from a theoretical point of view.

3. It lays out the foundations of a practical solution to the problem of motion safety in dynamic environments from an Inevitable Collision States perspective. Specifically it address two main points:

   (a) The characterization of the ICS set for a given robotic system with an algorithm or ICS-Checker that determines whether a given state is an ICS or not. This is an intricate problem since characterizing the ICS set requires in theory to reason on the state-time space of the robotic system at hand, and above all to consider all possible trajectories that the robotic system can follow from any given state. Similar to a Collision-Checker that plays a key role in path planning and navigation in static environments, it could be argued that an ICS-Checker is a fundamental tool for motion planning and navigation in dynamic environments. Like its static counterpart, an ICS-Checker must be computationally efficient so that it can meet the real-time decision constraint imposed by dynamic environments.

   (b) The determination of a control that takes the robotic system from one non-ICS state to another within a collision avoidance scheme. This decision-making module has as objective to keep the robotic system safe. By preventing the system to fall in an ICS is possible to guarantee its motion safety with respect to the model of the future which is used. The guarantee is derived from the ICS definition. When the robotic system's state is not an ICS it means that at least one collision-free trajectory exists and in consequence that is possible to follow it to avoid collision.

1.3 Document organization

The document is organized as follows. Chapter 2 presents first an analysis of the motion safety problem identifying the key aspects that must be used in the evaluation of any navigation method. Armed with such key issues or safety criteria a literature review is presented to locate this work in the context of what it has been done in the research community. Chapter 3 lays the foundations for the rest of the document. It presents a definition of what it is understood as motion safety and introduces the necessary notations to formally define the Inevitable Collision States (ICS) and its properties.

Chapter 4 presents Ics-Check. A generic and efficient way of determining the safety of a given state for planar robotic systems with arbitrary dynamics moving in dynamic environments. Chapter 5 explains Ics-Avoid, the ICS-based collision avoidance scheme, which takes the safe states, verified with the algorithms presented in the previous chapter, and use them to safely navigate through a dynamic environment. The principle which allows to guarantee safe transitions between non-ICS states is also introduced. Chapter 6 shows the results from simulation and real experiments: a wheelchair in indoor environments. The approach is shown to be tractable and an evaluation of its weak points is presented. Finally, Chapter 7 summarize the approach, the contributions of the work and outline the lines of research for future work.

Résumé

Dans le Chapitre 1, nous avons présenté une introduction générale de la thèse, les motivations et objectifs du travail ainsi que les principales contributions du manuscrit.

La motivation principale du travail est le problème de navigation sûre pour les véhicules autonomes en environnement dynamique. Ce problème prend une importance quand les robots sont opérés dans des environnement où les personnes sont présentes. Des exemples des dites applications sont les voitures automatisées et les robots de service.

La difficulté principale dans la détermination d'un mouvement sûr dans des environnements dynamiques vient du fait qui dans les dits environnements existe le facteur du temps. En effet, le temps impose des restrictions dans les décisions du robot et dans le modèle utilisé pour représenter les caractéristiques du robot et les objets de l'environnement.

Une manière d'aborder ce problème est par le concept des Etats de Collisions Inévitables (ICS). Un ICS est un état dans lequel, quelque soit le contrôle appliqué au système robotique étudié, celui-ci entre en collision avec un obstacle. Donc le principales contributions du manuscrit sont:

- Approfondir l'étude du concept d'ICS et son impact dans la sécurité de mouvement
- La conception d'algorithmes pour une solution pratique du problème de navigation sûre dans des environnements dynamiques par l'emploi d'ICS.

Chapter 2

Motion Safety- State of the Art

Since the early days of mobile robotics, researchers have seek to give to their robots the power to successfully navigate through their environment. This task, so natural in some of the simplest creatures, has proved to be a challenging one. Part of the challenge comes from the large spectrum of techniques that must work together to achieve the desired result. Among them are techniques for localization, mapping, path planning, obstacle avoidance and motion control. This work will concentrate in what from now on will be called "navigation methods" (methods at a decisional level that focus in determining the robot future course of action). Although a huge diversity and variety of these methods can be found in the literature, this chapter won't opt to make an exhaustive enumeration and description of them (that will be more appropriate for another kind of endeavour, like writing a book in the subject). Instead, an evaluation of a selection of the most prominent ones will be made. The evaluation is based in key issues that appear in an analysis of the notion of *motion safety* for robotic systems. These issues or safety criteria will turn out to be useful in determining the limits and the appropriate conditions to employ a given method. With the light given by them it will be easier to understand how the motion safety problem has been addressed in the robotics community and see, that in fact, many assumptions made by the navigation methods have been accepted without challenging them or validating their impact in the motion safety of the robotic system. As consequence, the applications where the navigation methods are employed may be regarded as safe when in reality they are not. This may become critical specially with the applications where human lives are involved.

2.1 Motion Safety Analysis

When looking to the reasons why a robot system collides in a dynamic environment is possible to make a distinction between two kinds. The first kind are those who have nothing to do with the way in which the motion safety problem is addressed. The second kind are those who do. Examples of the first ones are easy to list. They include robot hardware failure, errors or bugs in the software, misperception of the environment (*e.g.,* an erroneous interpretation of the sensor data), etc. The latter, on the contrary, are much harder to enumerate. Fortunately, analysis like the one done in [Fra07] have explored the motion safety issue at an abstract level and have laid down "safety criteria" whose violation by a navigation method is likely to yield collisions. The safety criteria are associated with how the key aspect of time is handled by a robotic system when deciding its future course of action. Specifically, a robotic system in a dynamic environment needs to:

1. Reasons about the **future**,

2. while respecting a **decision time** constraint,

3. with the **appropriate lookahead**.

To illustrate them a simple example called the "compactor scenario" will be employed (Figure 2.1). In this example we have a point robotic system (denoted as $\mathcal{A}$) which is placed between two plates. One plate is moving ($\mathcal{B}_m$) and the other is static ($\mathcal{B}_f$).

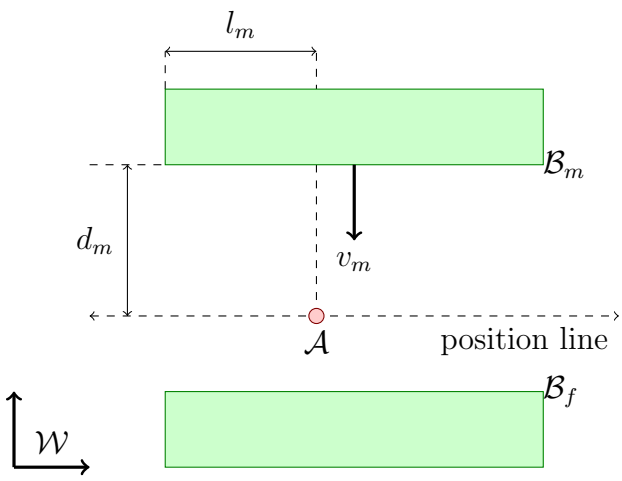


Figure 2.1: Compactor scenario.

The moving plate close the gap between it and the static one as it advances towards the latter with a velocity v_m. If the system doesn't move out of the way it will be crush between the plates. To ease the explanation, lets suppose that the system can only move right or left along a horizontal line (henceforth called the *position line*). In this way the robot state (the set of variables that adequately describe the condition of the system) is 1D and determined by its scalar position on the position line. Assuming the robotic system is controlled by its velocity (limited to a maximum) $v \in [-v_{max}, v_{max}]$ then the dynamics of $\mathcal{A}$ is given by $\dot{s} = v$. Let d_m denotes the distance between $\mathcal{A}$ and the moving plate $\mathcal{B}_m$, the time to collision can be easily computed: $t_c = d_m/v_m$. Let the distance to reach the nearest side of the plate is l_m (on the left side in this case), a collision is inevitable if the minimum time to escape (the time it takes to traverse l_m) is greater than the time to collision: $t_e = l_m/v_{max} > t_c$. Assume that the robot is not in that situation and can escape a collision. Now, let's see what happens when the robot needs to decide its future course of action.

If the system starts its decision process by considering the first point of the safety criteria then it will need to ***reason about the future***. One way of doing it is to add the time dimension ($\mathcal{T}$) to the configuration ($\mathcal{C}$) or state space ($\mathcal{S}$) as in [ELP87, Fra98]. In this representation the system's dynamics and the future behaviour of the objects can be considered simultaneously. Figure 2.2 shows the compactor scenario in $\mathcal{S} \times \mathcal{T}$. There the State×Time space of the system $\mathcal{A}$ is 2D: position×time. The robot's dynamics are represented with the concept of reachable states, in this scenario is an upside-down infinite cone whose apex is the current position of $\mathcal{A}$ and whose aperture is a function of v_{max}. Now, the future behaviour of the moving plate $\mathcal{B}_m$ is represented with states which are forbidden at a specific time (collision states CS). During its motion, $\mathcal{B}_m$ sweeps across the position line from time t_c for a duration depending upon the width of $\mathcal{B}_m$ and its velocity v_m. This yields a rectangular set of state-time (p, t) wherein $\mathcal{A}$ is in collision with $\mathcal{B}_m$ (the rectangle labeled CS in Fig. 2.2). CS is a forbidden region that $\mathcal{A}$ must avoid. A future course of action of the system must be in such way that the state trajectory of the system in $\mathcal{S} \times \mathcal{T}$ doesn't intersect the collision states and is inside the system's reachable states. The space-time model clearly captures the fact that, if $\mathcal{A}$ stays put, it eventually enters CS and a collision occur. It also shows that if the future evolution of $\mathcal{B}_m$ is not taken into account (*i.e.,* if $\mathcal{B}_m$ is treated like a fixed object), the region CS does not appear in the space-time and $\mathcal{A}$ cannot be aware of the upcoming collision risk (hence the importance of modeling and reasoning about the future evolution of the moving objects).

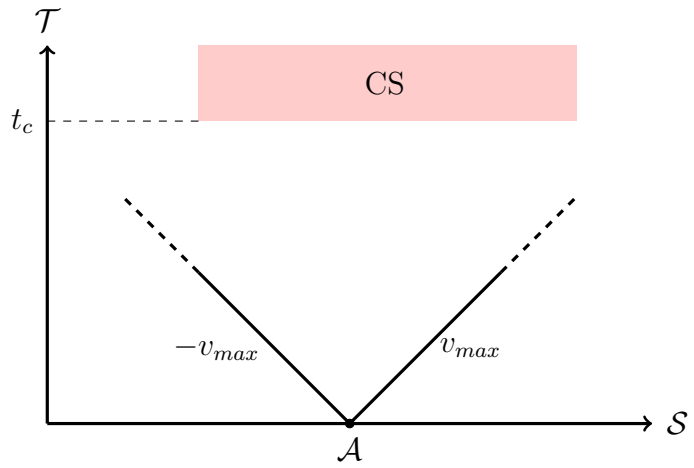


Figure 2.2: Reasoning about the future.

Furthermore, the robotic system cannot take much time to make a decision if it wants to avoid being crushed by the compactor. It has to respect a ***decision time constraint***. In the compactor scenario is quite obvious that if the robot takes more time than the time to collision $t_c = d_m/v_m$ to come up with a decision then a collision will have occurred before the system has even decided what it would do next. Now, this is not enough, the system $\mathcal{A}$ has to move to the right or to the left until it exits the compactor in order to avoid a collision. Thus, the decision time value must be selected in such way that it leaves enough time to the system to escape. If $\mathcal{A}$ decides to move to the left, it takes at least time t_e to exit the compactor which means that $\mathcal{A}$ should start moving to the left at least before time $t_l = t_c - t_e$, otherwise it does not have the time to exit the compactor on the left side. Likewise, there is an upper bound t_r on the time where $\mathcal{A}$ should start moving to the right (Figure 2.3). The maximum time that $\mathcal{A}$ has in order to make a move is $t_c - \max(t_l, t_r)$ whose value will depend if the system is closer to the left or to the right side of the compactor. Lets assume that t_d is the time it takes $\mathcal{A}$ to decide its future motion. If t_d is greater than $t_c - \max(t_l, t_r)$ then $\mathcal{A}$ is doomed, a collision will be inevitable. Here is the decision time constraint mentioned above. Note how the value of t_d depends on the state and of the environment where the robotic system is immersed (through the position, size and velocity of $\mathcal{B}_m$). Its value can become arbitrary small, just consider that the moving plate $\mathcal{B}_m$ is closer to $\mathcal{A}$, *i.e.*, $d_m \to 0$, then the time to collision tends to zero $t_c \to 0$ and in consequence the decision time $t_d \to 0$.

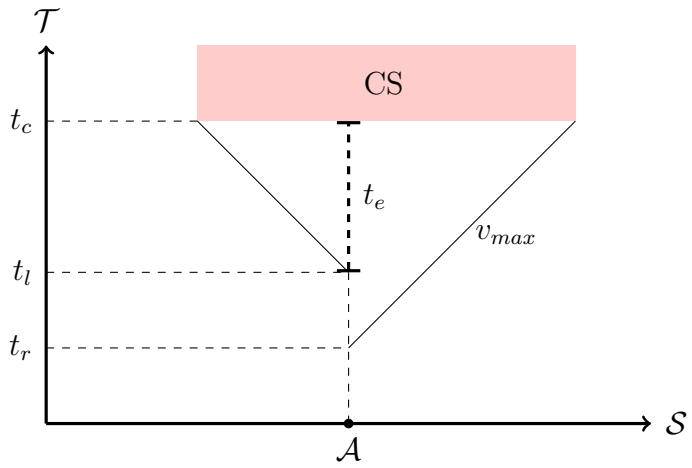


Figure 2.3: Decision time constraint.

It has been shown that modeling the future evolution of the environment and reasoning about it is necessary for the safety of the robot. Now the question is: with what ***lookahead***?. In other words, how far into the future should the modeling/reasoning go? Following with the compactor example, the answer is straightforward: the lookahead time t_{la} must be greater than $t_e + t_d$. If not, by the time $\mathcal{A}$ becomes aware of the risk caused by $\mathcal{B}_m$, it no longer has the time to decide that it should move to the left and execute this motion. Similar to the decision time constraint, the lookahead depends on the environment considered and in fact can become arbitrary large. Consider in this example that $\mathcal{B}_m$ is very long and very slow, *i.e.*, $l_m \to \infty$ and $v_m \to 0$ then $t_{la} \to \infty$.

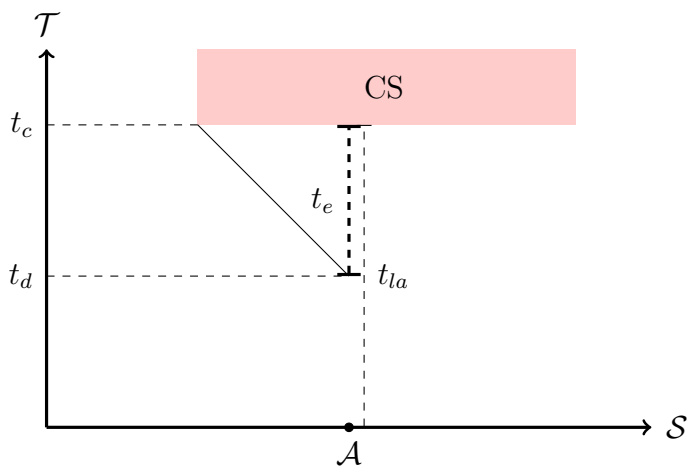


Figure 2.4: Appropriate lookahead.

In summary, the motion safety requirements for a robotic system boil down to three rules:

1. Reasoning about the future to consider: its own dynamics and the environment objects' future behaviour.

2. Decision time constraint: upper-bounded decision time t_d.

3. Appropriate lookahead: lower-bounded lookahead t_{la}.

These three safety criteria are related to time. In a dynamic environment, the time dimension is the key aspect. In the compactor scenario, the bounds in t_d and t_c are: $t_d < t_c - t_e$ and $t_{la} \geq t_e + t_d$ with $t_c = d_m/v_m$ and $t_e = l_m/v_{max}$.

If a method neglect or relax the safety criteria then the system's motion safety will be impacted. This assertion will become clearer as a presentation of a selection of methods which are considered relevant is made in the next section. Each method will be first approached by a description of the principles and techniques in which they are based. Then, their capacity to guarantee collision avoidance will be analyzed. In doing so, it will be shown how the assumptions made by them that do affect the motion safety of the robotic system are related to the safety criteria presented in this section.

2.2 Navigation Methods

When reviewing and classifying the extensive body of work present in the literature one difficulty comes from the varied ways the navigation problem can be conceptualized and decomposed in smaller parts. However, this variety tends to fade away when viewed through the "classic" perspective of deliberative/reactive levels.

2.2.1 Deliberative Methods

The deliberative or global methods intend to produce a complete set of actions. The solution, when found, is commonly known as a global or long term plan. Given some general knowledge about the environment (usually a priori information in the form of a map) and a goal to reach in it, the methods on this level produce a path that when executed will cause the robot to attain its target.

Motion planning algorithms

Motion planning algorithms have a long and prosperous history by now. They origin can be traced back to the the classic definition of the "Piano Mover's Problem" [Rei79] but it is the seminal work in the Configuration Space ($\mathcal{C}$) [LP83] that laid the foundations of the field. A configuration of a robotic system is the specification of the position and orientation of the system's reference frame with respect to the workspace. The configuration space is simply the set of all possible configurations of the robotic system. The beauty of the configuration space is that the robotic system (denoted as $\mathcal{A}$ from now on) is represented as a point, regardless of its actual shape. Static obstacles are mapped to forbidden regions (the set of system configurations where an intersection in the workspace occurs). Accordingly, the configuration space is divided into disjoint subsets of the configuration space (free space $\mathcal{C}_{free}$, obstacle space $\mathcal{C}_{obs}$ and contact space $\mathcal{C}_{contact}$). The basic motion planning problem can be stated as:

Given an initial configuration $\mathbf{q}_{initial}$ and a goal configuration $\mathbf{q}_{goal}$ is it possible to compute a collision-free motion between them?

The search for an answer to this question has derived in a large number of algorithms which can be roughly classified in:

- roadmap methods
- cell decompositions
- sampling-based methods

The reader is referred to one of the books in the subject [CLH+05, Lat91, LaV06] to find a detailed description of the methods and the techniques employed. Next, a brief presentation of their characteristics is done.

The basic idea behind the ***roadmap methods*** is to capture the connectivity of the $\mathcal{C}_{free}$ by building a graph or "roadmap". In doing so, these methods reduce the dimensionality of the problem which is why they are also known as "retraction" methods. Once the graph has been built the problem is reduced to first connect the initial and final configuration to the graph and then use a graph search algorithm (such as Dijkstra[Dij59] or A*[Dij68]) to find a path that connects $\mathbf{q}_{initial}$ to $\mathbf{q}_{goal}$. These methods are complete, *i.e.,* they will always find a path in finite time when one exists, and will let us know in finite time if no path exists. The hard part of these methods consist

in building the graph. Among the strategies that have been proposed to do that are: visibility graph [LPW79], voronoi roadmap [CD88, CB00] and silhouette method [Can88].

These methods were originally designed to deal with static environments, however, they can also be used in dynamic ones. There are basically two ways of doing this. The first is to represent the dynamic environment by adding the time dimension to the configuration or the state space as in [ELP87, Fra98]. In doing so, the dimensionality of the problem increase. The second option is simply to replan from scratch each time new information arrives. Given an updated graph or roadmap, a new path can can be planned from the current configuration of the robot to the goal configuration. Unfortunately the two options are not viable if we consider the safety criteria reviewed in the previous section. In particular we have the time decision constraint. The complexity of a general solution for these methods is too high (PSPACE-hard [Can88]) for either increasing the dimensionality of the problem or plan from scratch frequently.

Cell decompositions methods consist in decomposing the free space into a number of disjoint sets called cells. A connectivity graph that represents the adjacency relation between the cells is employed to search a path between two configurations. Each cell is represented as node in this graph. Two nodes are connected in the graph if and only if the two corresponding cells are adjacent. There are two types of methods: exact (*e.g.,* trapezoidal [Cha87], critical-curve [SS83a], cylindrical algebraic [SS83b] and connected balls [BK01, VKA05]) and approximate (*e.g.,* rectanguloid[Elf89] and 2^m tree). The main difference (as their names indicate) is that the exact methods generates an exact decomposition (*i.e.,* the union of the cells is exactly the free space) whereas the approximate methods try to approximate the structure of $\mathcal{C}$ with cells that have a simple shape like, for example, rectanguloids.

Similar to the previous approaches the cell decomposition methods were conceived for static environments. Furthermore, most of them operate in low dimensional spaces (the complexity of the subdivision algorithm increase exponentially with the dimension of the space). This characteristic make impractical to use a $\mathcal{S} \times \mathcal{T}$ representation to account for dynamic environments. Furthermore, the time decision constraint of the safety criteria makes infeasible to plan from scratch each time a change in the environment is detected: a complete plan from the current configuration to the goal configuration is computationally expensive. One alternative option to comply with the decision time constraint is to repair the path only in the portions that are affected by a detected change in the environment. This

means that the subdivision algorithm must be run in the affected regions of $\mathcal{C}$ or $\mathcal{S}$, the adjacency graph repaired accordingly and a valid path in the renewed graph found. Replanning algorithms such as [Ste95, KL02, LFG+05] are fast enough for finding a new path once the adjacency graph has been built, however performing the space decomposition and repairing the adjacency graph still remains in practice limited to low dimensional spaces. This is the reason why this kind of replan strategy use at best $\mathcal{S}$ and not $\mathcal{S} \times \mathcal{T}$. The consequence is that the representation of the environment is usually only a time slice and not a complete picture of the dynamic environment with its time dimension. As seen in the motion safety analysis, reasoning without considering the future behaviour of obstacles has an adverse impact in the motion safety of the robots.

As opposed to classic roadmap methods, ***sampling based methods***, avoid to explicitly build a representation of the free space (or equivalently a construction of $\mathcal{C}_{obs}$). Instead they conduct a search that probes $\mathcal{C}$ with a "sampling" scheme. To that end, a collision checker is in charge of verify if the given sample belongs to the occupied or free space. These type of methods have been demonstrated to work well in high dimensional spaces where is difficult to do a discretization ({eg cell decomposition) or to use roadmap methods efficiently. They satisfy a weaker form of completeness (as many of them are based in random sampling they are said to be *probabilistically complete i.e.,* the probability to found an existing solution converges to one as the number of samples increase). Among the examples of this kind of methods are: randomized path planner (RPP) [BJ91], Ariadne's clew [BATM93], probabilistic roadmap planners (PRM) [KSLO96] and rapidly-exploring random trees (RRT) [LaV98, LK01].

As these planners can work with high dimensional spaces they can operate in $\mathcal{S} \times \mathcal{T}$ to represent a dynamic environment. This is why navigation schemes that deal explicitly with dynamic environments have been proposed based in these methods. The method in [HKLR02] is a PRM based planner that encodes the kinematic and/or dynamic motion constraints of the robot with a control system that samples the robot's state-time space by picking control inputs at random and integrating its equations of motion which results in a probabilistic roadmap. The roadmap is not precomputed but instead a new roadmap that connects the initial and goal state is constructed from scratch at each planning query. In [vdBO05] a roadmap is precomputed for the static part of the environment without considering neither the dynamic obstacles nor the time dimension. In the query phase, the method only needs to deal with the dynamic obstacles when searching for a trajectory between the start and goal configurations. To find the trajectory there is a

local level where trajectories on single edges of the roadmap are found in a grid in state-time space and a global level where the local trajectories are coordinated using an A* based search to find a global trajectory in the entire roadmap. The work of [BV03] is an example of extending RRTs to interleave planning and execution. They introduce two additions to the planner: the waypoint cache for replanning and adaptive cost penalty search. The waypoint cache serves to use a plan that was found in a previous iteration as a guide for the current iteration. The adaptive cost penalty search is based in the idea that having a plan that could be a not very good one is better than no plan at all, and once a plan is in the cache the search is biased toward improving it. Finally in Anytime RRT [FS06] the principle is to locally repair a plan computed with the classical RRT by deleting the invalidated nodes and performing a new search to add new nodes that preserve the path between the initial and final configurations.

All this methods can take into consideration the safety criteria of reasoning about the future and appropriate lookahead. However, as the running time of a randomized technique cannot be upper bounded it can be argued that given the intrinsic complexity of motion planning in dynamic environments it seems unlikely that a hard decision time constraint could ever be met in realistic situations.

2.2.2 Reactive Methods

Reactive methods foresee for the immediate time and return a single action to be performed right away. This level deals with the unexpected events encountered during the execution of a previous conceived plan (normally coming from the deliberative level). The methods take constant updated information (frequently provided by sensor readings) and modifies the current plan with local modifications in order to avoid the detected objects. The reactive methods are relatively simple techniques in nature which make them suitable for execution in real-time.

Potential Field Methods

In the potential field methods the robotic system is considered as a particle immersed in an artificial potential field where is attracted to the goal and repulsed away from obstacles. Therefore, the model of the environment is specified with a potential function that determines the forces exerted in the robotic system. According to the source of the available information about the environment, the forces can be computed off-line (when the information is known a priori) or on-line (relying in sensor readings detecting close obstacles

during execution). The robot choose in an iterative fashion the movement to take by selecting a direction which is conventionally pointed by the negative gradient of the sum of forces. The procedure continues until the robot reaches the goal configuration if successful.

Conventional potential field methods [BK89, Kha86] are subject to problems like the local minima, that, as its name suggests, is a local minimum point in the artificial field which is not the goal and where the robot system is taken when following the direction pointed by the gradient. Two main techniques are used to overcome this problem. The first one is to replace the gradient descent strategy for a guided search (if the robot is trapped in a local minimum heuristics such as random walk or search algorithms such as depth-first, best-first or A* are employed to attempt to find a way out). The second one is to produce potential fields which are navigation functions in the sense of [RK92] (*i.e.,* they are smooth and have only one local minimum situated at the goal thus eliminating the local minima problem, an example, which can be used when complete knowledge of the environment is available, is the non-optimal navigation functions generated using harmonic potential functions [KK92]).

One of the most relevant assumptions from a motion safety point of view is that all these methods consider that the surrounding environment is static. The methods make the assumption that the distance to obstacles (a main parameter in the computation of the repulsive forces) remains constant during the decision time step. This assumption doesn't hold in dynamic environments. As explained in the motion safety analysis one safety criteria when dealing with dynamic environments is the need to reason about the future behaviour of the obstacles. Recent works [GC02, Hua08] propose an extension more appropriate for dynamic environments. They account for the motion of obstacles by defining a potential function which takes into consideration not only the distance to the obstacles but also their instantaneous velocities. However, problems of local minima do exist in the resulting potential field which are worked out by heuristics or guided search that don't give guarantees that a solution will be found. Another feature of the potential field methods that can have an impact in the motion safety of the robot is that the result is expressed as a force that indicates the direction where the robot should move. This is alright for holonomic robots that can move freely in the space but the great majority of robotic models have kinematic and dynamic constraints that impede that the direction indicated by the method be followed right away. As a result a low-level controller is required to attain the desired direction and the motion before convergence is reached can at times be quite different from the expected one. In a sentence, these methods don't reason about the future of the system by neglecting its

own dynamics. This situation can result in problems with the motion safety of the system.

Vector Field Histogram Methods

Among these methods are ***Vector Field Histogram (VFH)*** [BK91] and its extensions VFH+ [UB98] and VFH* [UB00]. These methods create a local map of the environment around the robot. They use a polar histogram grid. Similar to the occupancy grids, each cell of the histogram grid stores the probability value that an obstacle is present in the direction associated with the cell. From this histogram grid the direction to where the robot should move is calculated. The procedure starts by finding all openings large enough in the histogram where the robotic system is capable of passing through. Then, each opening that is found is evaluated with a cost function. The cost function weights factors such as target direction, wheel direction and previous direction.

VFH+ extended the method by changing to a robot model which could take into consideration non-holonomic constraints. To find an opening in the histogram grid, instead of only using straight line paths, VFH+ also use circular arcs paths that describe more accurately the robotic system capabilities. Another improvement was the capacity to consider robots of different sizes.

Finally, the most recent extension ***VFH**** improves some of the problems inherent to local navigation methods. The main contribution of this method is that it verifies that the direction chosen by the method can guide the robot around an obstacle without getting trapped. This verification is done by combining an A* search algorithm with appropriate cost and heuristic functions.

All these methods assume implicitly an static environment. Their histogram grids are not capable of capturing appropriately the information concerning the motion of the obstacles present in a dynamic environment (although they are robust in the sense that they accumulates in a probabilistic fashion the sensor readings). In consequence the time dimension cannot be taken into consideration. This limitation impacts the motion safety of the robotic system because the methods fail to comply with one of the safety criteria presented in Section 2: reason about the future behaviour of the moving objects of its environment. Even if the methods operate at high rates a direction that has been chosen by them can become quickly invalid and even dangerous for the robot's motion safety if an unseen obstacle's block it in the future.

Velocity Space Methods

Of the many reactive methods at hand, worthy of special mention are those which operates in the Velocity Space (V-Space). The V-Space represents the set of all the velocities that are achievable by the robotic system. Two types of constraints are usually imposed by the navigation methods to the admissible set in the V-Space (the set of velocities from where the robot can select one command to apply at each time step). First, those derived from the system limitations (kinematic or dynamic) and, second, constraints of the physical environment coming from obstacles blocking certain velocities values due to their positions (if the robot select such velocity a collision would take place).

The ***Curvature Velocity Method (CVM)*** [Sim96] operates in the V-space composed of linear and angular velocities (v,ω respectively). The objects present in the environment are assumed to have a circular shape to allow an easier computation of the distance that the robot system will traverse from its position to the obstacle. The distance is computed assuming the robot follows a constant curvature path (where the curvature is given by $\kappa = \frac{\omega}{v}$). Only curvatures that lie inside the kinematic capabilities of the robotic system are considered valid for an objective function that serves to make the selection of the final command to apply to the robotic system. The objective function takes into consideration the distance to obstacles (giving preference of traveling longer distances without colliding with obstacles), the speed (preferring to travel at faster speeds) and heading (to bias the progression of the system towards the goal).

Among the assumptions reproached to the CVM are that obstacles need to be circular (which may be acceptable for some environments but not for others), the approximation of the robot movement only with circular arcs (when in fact the robot can change direction many times and draw different paths) and finally and probably the one with more impact in the motion safety of the system: that the environment is static. Once more, the safety criteria concerning reasoning about the future behaviour of obstacles is neglected.

The ***Lane Curvature Method (LCM)*** [NS98] is an extension to CVM to address some of its problems. In particular those derived from the assumption that the robotic system moves only along paths of circular arcs. The method divides the environment in a set of lanes that are built considering the maximum distance to obstacles along a desired goal heading and merging lanes when the distance between adjacent lanes is similar. The

most promising lane is chosen with the help of an objective function. The local heading of the robotic system is set appropriately to change lane if the selected one is not the same as the one where the robot is. Although the extension allow more flexibility in the paths described by the robots it doesn't address the assumption of a static environment. In a sentence, it improves the reasoning about the future of system by having a more accurate model of the robot's dynamics but completely ignores the future behaviour of the obstacles.

The ***Dynamic Window Approach (DWA)*** [FBT97] is one the most representative reactive avoidance methods using a V-Space representation. Its search space is also composed by all possible pairs of linear and angular velocities. A "dynamic window" enclose the set of reachable velocities (V_r) around the current velocity vector (v_c). It is computed for a short time interval (Δt) by taken into consideration the constraints of the robotic system in its translational and rotational acceleration/deceleration capabilities:

$$V_r = \{(v, \omega) | v \in [v_c - \dot{v_d}\Delta t, v_c + \dot{v_a}\Delta t] \wedge \omega \in [\omega_c - \dot{\omega_d}\Delta t, \omega_c + \dot{\omega_a}\Delta t]\} \quad (2.1)$$

A velocity command is included into the admissible set (V_a) if the system is capable of coming to a stop before colliding with a detected object in the environment:

$$V_a = \{(v, \omega) | v \leq \sqrt{2\rho_{min}(v, \omega)\dot{v_b}} \wedge \omega \leq \sqrt{2\rho_{min}(v, \omega)\dot{\omega_b}}\} \quad (2.2)$$

were $\rho_{min}(v, \omega)$ represents the distance to the closest obstacle in the circular path traced by the system when applying the controls (v, ω). Figure 2.5 illustrate the working principle, it shows the V-Space of the considered system with the admissible velocities noted as V_a, the forbidden velocities V_a^C (those which would take the system to collision, shown in red) and the set of reachable velocities belonging to the dynamic window (built around the current velocity vector). Choosing one velocity among those in the admissible set boils down to the optimization of an objective function. The objective function favours three items: the progression towards the goal, the clearance to obstacles and fast forward motion.

The DWA exhibits some limitations. To begin with, it is susceptible to local minima unless a mechanism to incorporate information about the connectivity of the free space is employed (Global Dynamic Window) [BK99]. It also assumes that the system moves only in circular arcs paths (at least during the considered Δt) which simplifies the computation of the distance to an object but which reduce the space of solutions by not considering a wider variety of paths. Finally, only the velocities that fall at the

interior of the dynamic window are considered to choose the instantaneous command to execute. Stated differently, an obstacle which is detected but that falls outside of the dynamic window when mapped to a velocity will be ignored. This implies that it discards potential valuable information which could help to achieve better performance. As shown in the motion safety analysis having an appropriate lookahead is critical for the motion safety of the robotic system. DWA simply choose to have a lookahead that has an arbitrarily value (the duration of the decision time step) and that probably is too shortsighted because it does not consider the particularities of the environment. Additionally, DWA also makes the assumption that the environment remains frozen during the decision time step and thus, it doesn't reasons about the future behaviour of the objects.

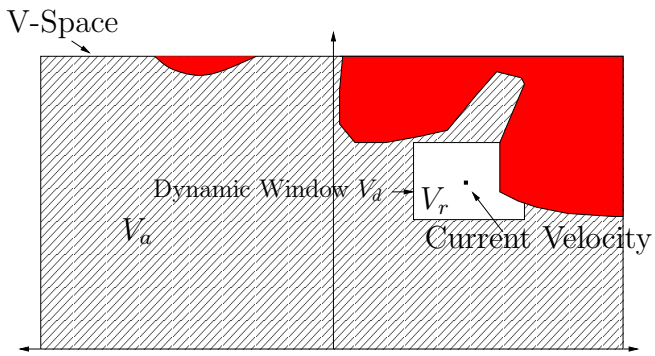


Figure 2.5: Dynamic Window.

Time Varying Dynamic Window (TVDW) [SP07] extends the classic Dynamic Window Approach by calculating at each time instant a set of immediate future obstacles trajectories in order to check for collision in the short term. In this respect TVDW is superior to DWA because it considers the safety criteria related to reasoning about the future behaviour of the obstacles and don't simply assume that the environment is static. The approach is not limited to a particular obstacle shape but instead assume that the representation of the environment is given as an occupancy grid map. The cells on the grid should be classified as free or occupied, and for the occupied cells a further important distinction is required: the identification of those that move (the ones that belong to the dynamic objects). For each moving cell (MC) its velocity vector with linear and angular velocities (v_{mc}, ω_{mc}) and motion heading θ_{mc} is assumed to be known. With this information, the set of predicted obstacle trajectories is generated. Each one starting from a MC and drawing the path resulting from applying the known (v_{mc}, ω_{mc}) during a Δt interval. The lookahead in TVDW is set equal to the time it takes to the robotic system to stop when traveling at maximum speed, so

usually is much longer than the DWA time step. A robot velocity pair (v, ω) is considered admissible if no collision occurs between the MC trajectories and the DW trajectory corresponding to that velocity vector. Figure 2.6 illustrates the method. It shows a circular robot system $\mathcal{A}$, its set of DW trajectories (resulting from different values for the tuple (v, ω)), the moving cells and their trajectories. The set of admissible controls for the system are those which their trajectories don't produce a collision (shown in green). Similar to DWA, this method also use a cost function to select one velocity control from the admissible set.

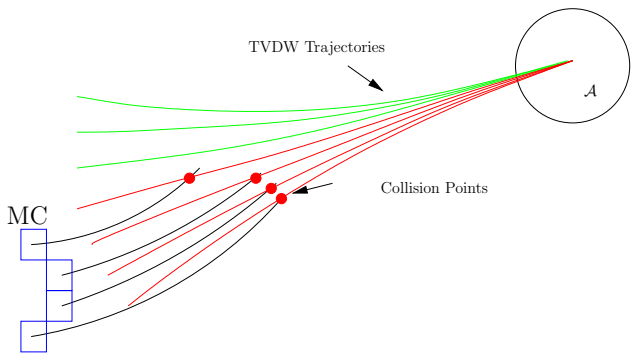


Figure 2.6: Time Varying Dynamic Window.

One of the assumptions made by TVDW (and also DWA) that have an impact in the motion safety of the system is related to what they consider to be a safe state. For those methods the robotic system has reached a safe state when it is static. Is not hard to come up with a situation when this can be extremely dangerous (*e.g.,* an autonomous car stopping in the train tracks when the train is approaching). For some systems this may even not be possible (a fixed wing AUV can't just stop in the middle of the air without crashing). Other assumption made by these methods its the type of paths that can be described by the robotic system (only circular arcs). Those type of paths are only a gross approximation of what the system can achieve, *i.e.,* the dynamics of the robot.

Dynamic Velocity Space (DVS) [OM05] is an interesting method that builds a velocity space that captures the dynamicity of the environment. It employs the concept of estimated arriving time to compute the times to potential collision and potential escape (assuming that the obstacles move with constant linear velocity and that the robotic system moves only in straight or circular arcs). This time-related information is added to a surface in the velocity-time space ($DOVS$) combining the collision and escape

information of individual obstacles. A 2D projection V_{DOVS} of the $DOVS$ surface is used to characterize the set of forbidden velocities. Dynamic constraints are considered through a window of admissible velocities (which can span several sampling periods). The selected velocity is such that it is close to the goal (which is mapped to a velocity in the 2D projection considering the robot heading and given preference to high velocities), belongs to admissible velocities and is not inside of the forbidden set.

From the safety criteria point of view this method does reasons about the future behaviour of the obstacle, unfortunately it does it in a simplified way by considering their movement only as constant linear velocities. Is not clear how the collision and escape times could be computed in more general models of motion. Furthermore, the reasoning about the future of the robotic system is limited to straight and circular arcs which may be restrictive for some applications. However, an interesting characteristic of the method is its ability to cope with selectable lookahead values by adjusting the span of sampling periods to characterize the set of admissible velocities.

The ***Velocity Obstacles (VO)*** family of methods are others simple but effective navigation methods well suited for dynamic environments. In its simpler and original form [FS98], VO is a reactive approach that operates also in the V-Space of the robotic system considered (here the V-Space is composed of linear velocities $\mathbf{v} = (v_x, v_y)$). VO takes into account the future behaviour of the moving objects. The hypothesis is that the obstacle will maintain its current linear velocity and thus its future trajectory is a constant linear one. Each object in the environment yields a set of forbidden velocities whose shape is that of a cone (*cf* Fig.2.7 depicts the linear velocity space of the robotic system, the red conical region on the right is the set of forbidden velocities that would yield a collision between the robot $\mathcal{A}$ and the moving object $\mathcal{B}$). Should the robotic system select a forbidden velocity, it would collide with the moving object at a later time (possibly infinite) in the future. Formally:

$$VO = \{\mathbf{v} | \exists t \ > 0, (\mathbf{v} - \mathbf{v}_{\mathcal{B}})t \in D(\mathbf{x}_{\mathcal{B}} - \mathbf{x}_{\mathcal{A}}, r_{\mathcal{A}} + r_{\mathcal{B}})\} \tag{2.3}$$

where the robot system $\mathcal{A}$, has position $\mathbf{x}_{\mathcal{A}}$ and radius $r_{\mathcal{A}}$; and obstacle $\mathcal{B}$ has position $\mathbf{x}_{\mathcal{B}}$, velocity $\mathbf{v}_{\mathcal{B}}$ and radius $r_{\mathcal{B}}$. In practice, velocities yielding a collision occurring after a given time horizon (t_H) are considered as admissible.

One issue (often overlooked) with VO is that, in a closed environment, every velocity is forbidden since it eventually yield a collision. For that

reason, the lookahead time in VO cannot be arbitrarily large which means that in some circumstances will not be possible to set an appropriate value as indicated by the safety criteria. Other assumption made by VO is that the shape of the system and obstacles must be discs (an non circular obstacle must be approximated by a patch of contiguous discs). Furthermore, the assumption that the future behaviour of obstacles is a linear constant velocity can be very restrictive in some situations (imagine a car entering a curve in a highway).

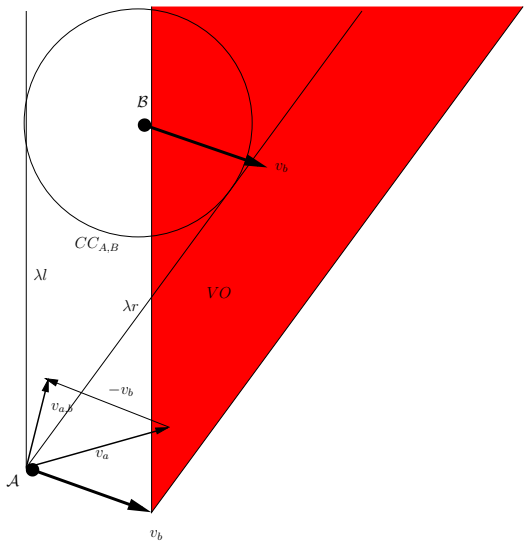


Figure 2.7: Velocity Obstacles.

VO was extended by ***Non-Linear Velocity Obstacles (NLVO)*** [LLS05] to consider known arbitrary velocity profiles for the moving objects, *i.e.,* object trajectories that are not necessary constant linear ones. NLVO consist of all velocities of $\mathcal{A}$ at t_0 that would result in collision with $\mathcal{B}$ at any time $t_0 \leq t \leq t_H$. Geometrically (Fig.2.8), $NLVO(t)$ is a scaled $\mathcal{B}$, bounded by the cone formed between $\mathcal{A}$ and $\mathcal{B}(t)$, thus, NLVO is a warped cone with apex at $\mathcal{A}$ and formally defined as:

$$NLVO = \bigcup_{t_0 \leq t \leq t_H} \frac{\mathcal{B}(t)}{t - t_0} \tag{2.4}$$

Similar to the original VO, NLVO cannot have an arbitrary large lookahead time and only considers circular shaped obstacles. A recent extension Finite Velocity Obstacle (FVO) [GCK+09] explicitly accounts for the lookahead by defining a discrete time interval and truncated cone to impose additional constraints that guarantee collision avoidance during a discrete time interval.

Other extensions like Generalized Velocity Obstacles (GVO) [WvdBM09] take into account the non-holonomic constraints of the robotic system which is required to correctly reason about the future behaviour of the system.

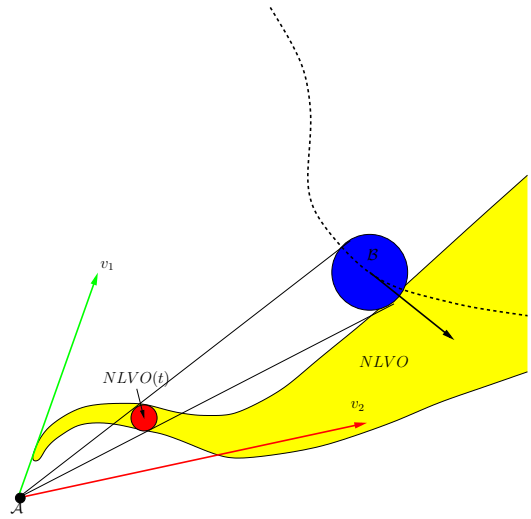


Figure 2.8: Non Linear Velocity Obstacles.

Trajectory Parameter Space Methods

Recently, other reactive navigation methods in the Trajectory Parameter Space (TP-Space) have been proposed to decouple the problem of kinematic restrictions and obstacle avoidance for an any-shape robot system [MM06, BGF08]. A TP-Space is a two dimensional space where each polar coordinate (α, d) maps to a robot configuration (x, y, θ) in a sampling surface of the C-Space. The sampling surface can be visualized as the surface resulting of joining the set of pose trajectories drawn by the system while applying a generating function (a control function that when executed by the system draws a family of path models). Examples of generating functions can produce circular arcs, spiral segments, asymptotically heading trajectories, etc. A valid generating function depends in one control parameter: α, which paired with the distance value d along the generated trajectory should define an unique point in the C-Space (note that the measurement of the distance is done in the C-Space and thus combines linear and angular values). Figure 2.9 shows an example of one such C-Space sampling surface.

The sampling surface then, is simply a representation of what the robotic system is capable of do given its kinematic constraints and a control function. To "straighten out" the sampling surface into the two dimensional TP-Space

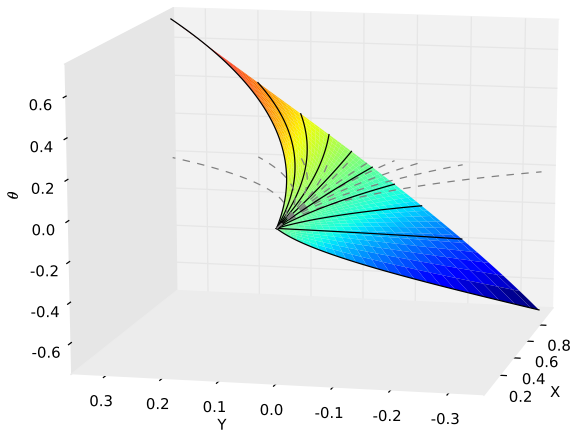


Figure 2.9: C-Space sampling surface resulting from a generating function of circular-arcs (shown in dashed-lines). Solid black trajectories in the surface correspond to different values of α.

all that must be done is to consider tuples (α, d). The tuple still stands for a C-Space point but the problem is now posed in a low dimensional space which is easier to handle. Finally, a mechanism to incorporate obstacle information in the TP-Space is also needed, that is, getting TP-Obstacles which involves a transformation of the intersection points of the pose trajectories composing the sampling surface and the C-Obstacles (the obstacles in the C-Space). As this procedure is computationally expensive the use of pre-computed look-up tables is commonly employed. The procedure to fill the look-up tables starts by making a discretization of the physical space around the robot in a rectangular grid where each cell in the grid store their associated TP-Obstacle. If the shape of the robot (which can be anything) along a pose trajectory generated by one pair (α, d) touches the cell, then that pair will be stored as a TP-Obstacle in the cell. Different pairs of (α, d) can produce pose trajectories where the shape of the system touch the same cell, so, in general, each grid cell can contain several (α, d) pairs. The advantage of the transformation to the TP-Space is three-fold: the transformation allow to consider the robot as a free-flying point in the TP-Space (since its shape and kinematic restrictions are already taken into account); second, any previous "classic" obstacle-avoidance method constrained to holonomic point or circular robots can be applied to an any-shape robot in the TP-Space; and finally, a wider range of paths compatible with the system kinematic

restrictions (a superset of the typical circular arcs) can be employed to avoid obstacles, increasing the space of solutions to situations that were harder or impossible to solve with other methods.

Although these methods have been shown to solve difficult problems where other methods have failed (given that they can use a greater path diversity) they make a big assumption that impacts the motion safety of the robotic system. They assume that the surrounding environment is static (at least during the decision timestep), they don't reason about the future behaviour of obstacles. As explain before, in dynamic environments an occupancy grid is not enough to represent the environment (as it will constantly change according to new observations) and the static mapping to TP-Obstacles may very well become dangerous if the speed of the obstacle is considerable.

2.2.3 Alternative Methods

Intermediate methods are all those that do not quite fit in the previous classification on deliberative/reactive methods. As its name suggests, they combine some features of both to address the problems that with a different perspective.

Deformation Methods

Deformation methods can be traced back to the Elastic Band concept [QK93]. They operate in a first stage by using a motion planning algorithm to give to the robotic system a collision-free path connecting the initial to the final configuration. The path is based in whatever a priori knowledge of the environment is available. Then, at the execution stage the path is deformed as new information of the environment is discovered. Two types of forces are exerted in the path. Those coming from obstacles that push it away and those that comes from internal constraints aimed at maintaining the connectivity of the path, that is to say, that following the path remains feasible for the robotic system given its dynamic constraints. The first works considered only holonomic systems but extensions like [KJCL97, LBL04] extended the method to nonholonomic ones. However, one drawback of these path deformation methods is that the deformation doesn't consider the time dimension. Things like deforming excessively the path when an obstacle is cutting through can be avoided by simply stopping and letting the obstacle pass by. Promising trajectory deformation methods that operate in the $\mathcal{C} \times \mathcal{T}$ or $\mathcal{S} \times \mathcal{T}$ have been proposed to address this type of issues for holonomic [KF07] and non-holonmic systems [DF08]. However,

maintaining the connectivity of the trajectory in the $\mathcal{S} \times \mathcal{T}$ requires the use of relatively complex trajectory generation algorithms that are still an open research problem. These algorithms need to be efficient to account for the safety criteria such as the time decision constraint imposed by dynamic environments. Furthermore, no upper time bounds in the computation of the initial trajectory (or path) connecting the initial to the goal needed by the deformation methods are imposed.

Partial Motion Planning

Partial Motion Planning (PMP) [PF05] is a motion planning scheme that fits perfectly in the intermediate methods. It doesn't compute a complete sequence to the goal as deliberative methods do. Nor calculate the next command at each time step. Instead it considers the time decision constraint and compute as many steps as possible to the goal within the available time. When the decision time is up, PMP returns the best partial motion to the goal computed so far. In a sentence is an anytime motion planning method. The method operates in stages that are performed at each time step. The first is the execution of the plan from the previous step. In parallel the model of the future is updated based on new observations. Then a motion planning algorithm is executed to progress the plan towards the goal. If the goal is not reached, the path closest to the goal is set as the current plan.

Replanning Methods

Replanning methods compute a new plan frequently to react to unexpected events that appear in a dynamic environment. The use of replanning is justified by the fact that in unknown environments and in environments with moving obstacles the information known a priori in which an initial plan is based is very likely to change during the execution of the plan. Algorithms like the one presented in [BH95], Focused Dynamic A* (D*) [Ste93] and its simpler version D*-Lite [KL02] are able to quickly replan based on the latest observations. Every time a moving obstacle invalidate part of the plan, the algorithm search an alternative path that repairs the solution towards the goal. In this way the previous plan is discarded and a new one starting from the current state is created. Although these planners respect the time decision constraint during its execution they assume an initial plan is given, however no upper bounds are set to the time it can take to come up with this initial plan.

2.3 Conclusion

When dealing with dynamic environments a navigation method must take into consideration all the safety criteria presented in the first section of this chapter. Failure to do so result in navigation strategies which are not safe for the robotic system. Surprisingly, the description of the navigation methods have shown that this is hardly the case. The design choices and assumption made by the navigation methods put in evidence that safety guarantees cannot be provided if they are confronted with dynamic environments. However, these same methods may work perfectly fine in less challenging environments where the assumptions made are valid. Unfortunately, the real world is a dynamic environment and where human lives are at sake safety guarantees must be given.

Résumé

Dans le Chapitre 2, nous avons présenté une analyse de l'état de art dans le domaine de la navigation des robots. Nous avons identifié les aspects essentiels qui doivent être considérés du point de vue de la sécurité de mouvement. Avec ces critères nous avons évalué les différents algorithmes présents dans la littérature et nous avons trouvé que tous ignorent un ou plusieurs des dits critères de sécurité. En conséquence les méthodes existantes de navigation ne sont pas appropriées pour garantir la sécurité de mouvement d'un robot dans un environnements dynamique.

Chapter 3

Conceptual framework

This chapter lays down the foundations of the approach taken in this work. Armed with the presentation of useful notation is possible to arrive to the main axle around which this thesis revolves: Inevitable Collision States (ICS). Following its formal definition it is shown in which measure ICS respect the safety criteria presented in Section 2.1. Then a discussion of the level of motion safety that can be achieved with ICS is done. Finally the set of properties that render possible its characterisation is presented.

3.1 Notations

Let $\mathcal{A}$ denote a robotic system operating in a workspace $\mathcal{W}$. The dynamics of $\mathcal{A}$ is described by:

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

where $s \in \mathcal{S}$ is the state of $\mathcal{A}$, $\dot{s}$ its time derivative and $u \in \mathcal{U}$ a control. $\mathcal{S}$ and $\mathcal{U}$ respectively denote the state space and the control space of $\mathcal{A}$. Let $\mathcal{A}(s)$ denote the closed subset of the workspace $\mathcal{W}$ occupied by $\mathcal{A}$ when it is in the state s. Let $\tilde{u} : [0, \infty(\longrightarrow \mathcal{U}$ denote a *control trajectory*, *i.e.,* a time-sequence of controls. The set of all possible control trajectories over $[0, \infty($ is denoted $\tilde{\mathcal{U}}$. Starting from an initial state $s(t_0)$ at time t_0, a *state trajectory* is derived from a control trajectory $\tilde{u}$ by integrating (3.1). A state trajectory is a time-sequence of states, *i.e.,* a curve in $\mathcal{S} \times \mathcal{T}$ where $\mathcal{T}$ denotes the time dimension. Abusing notations, $\tilde{u}(s(t_0))$ and $\tilde{u}(s(t_0), t)$ respectively denote the corresponding state trajectory and the state reached at time t:

$$\tilde{u}(s(t_0), t) = s(t_0) + \int_{t_0}^{t} f(s(\tau), u(\tau)) d\tau \tag{3.2}$$

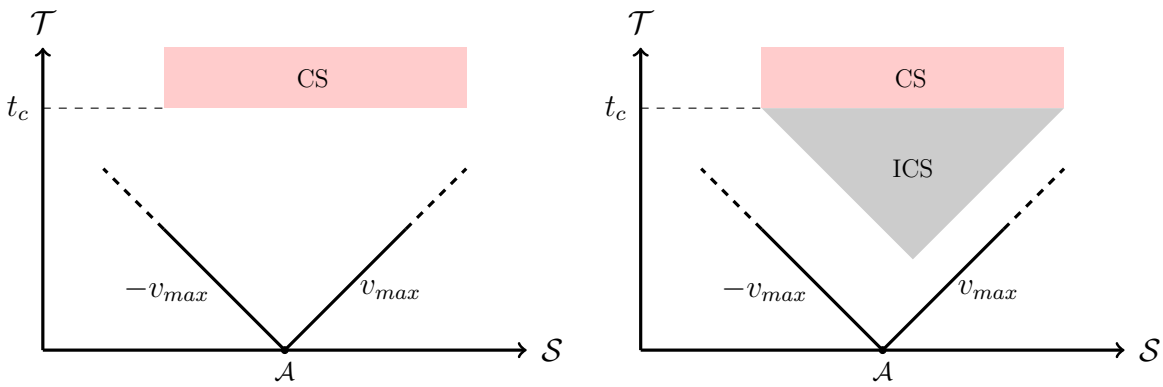
$\mathcal{W}$ contains a set of n_b fixed and moving objects defined as closed subsets of $\mathcal{W}$. Let $\mathcal{B}_i$ denote such an object. Since $\mathcal{B}_i$ maybe moving, the notation $\mathcal{B}_i(t)$ is used to denote the subset of $\mathcal{W}$ occupied by $\mathcal{B}_i$ at a particular time t in the future. Let $\mathcal{B}$ denote the union of the workspace objects (both in space *and* time):

$$\mathcal{B} = \bigcup_{i=1}^{n_b} \bigcup_{t\in[0,\infty(} \mathcal{B}_i(t) \tag{3.3}$$

Both $\mathcal{A}$ and the $\mathcal{B}_i$s are assumed to be rigid objects.

3.2 ICS definition

An ICS is informally defined as a state for which, no matter what the future trajectory followed by $\mathcal{A}$ is, a collision eventually occurs. A simple example may help to visualize the concept. Reconsider the "compactor scenario" presented in Section 2.1. There the collision states CS are the rectangular set of state-time (p, t) wherein $\mathcal{A}$ is in collision with $\mathcal{B}_m$ (the rectangle labeled CS in Fig. 3.1a). They are the result of the moving plate $\mathcal{B}_m$ sweeping across the position line of $\mathcal{A}$ from time t_c (time to collision) for a duration that depends upon the width of $\mathcal{B}_m$ and its velocity v_m. Now, remember that $\mathcal{A}$ is subject to dynamic constraints $v \in [-v_{max}, v_{max}]$ and its dynamics is given by $\dot{s} = v$. It is easy to visualize that there is an area that even though no collision exists the robotic system do not have the time to escape and a collision will occur in the future, no matter what action the system takes. This area is the ICS set (Figure **??**).



(a) Collision States (b) Inevitable Collision States

Figure 3.1: CS vs ICS.

The formal definition of Inevitable Collision States (ICS) is:

Definition 1 (**Inevitable Collision State**).

$$ICS(\mathcal{B}) = \{s \in \mathcal{S} | \forall \tilde{u} \in \tilde{\mathcal{U}}, \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}(t) \neq \emptyset\} \tag{3.4}$$

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

$$\text{ICS}(\mathcal{B}_i) = \{s \in \mathcal{S} | \forall \tilde{u} \in \tilde{\mathcal{U}}, \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\} \tag{3.5}$$

The ICS set yielding a collision with $\mathcal{B}$ for a given trajectory $\tilde{u}$ (or a given set of trajectories $\mathcal{E} \subset \tilde{\mathcal{U}}$) is:

$$\text{ICS}(\mathcal{B}, \tilde{u}) = \{s \in \mathcal{S} | \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}(t) \neq \emptyset\} \tag{3.6}$$
$$\text{ICS}(\mathcal{B}, \mathcal{E}) = \{s \in \mathcal{S} | \forall \tilde{u} \in \mathcal{E}, \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}(t) \neq \emptyset\} \tag{3.7}$$

Similarly, considering a particular object $\mathcal{B}_i$:

$$\text{ICS}(\mathcal{B}_i, \tilde{u}) = \{s \in \mathcal{S} | \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\} \tag{3.8}$$
$$\text{ICS}(\mathcal{B}_i, \mathcal{E}) = \{s \in \mathcal{S} | \forall \tilde{u} \in \mathcal{E}, \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\} \tag{3.9}$$

Finally, the ICS set yielding a collision with $\mathcal{B}_i$ for a given trajectory $\tilde{u}$ and time t is:

$$\text{ICS}(\mathcal{B}_i, \tilde{u}, t) = \{s \in \mathcal{S} | \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\} \tag{3.10}$$

3.3 ICS and Motion Safety Criteria

The definition of ICS allows to establish the measure with which the ICS concept respect the motion safety criteria presented in Section 2.1. The violation of any of them by a robotic system operating in dynamic environments is likely to yield a collision.

The first safety criteria that appeared in the analysis is to ***reason about the future***. Two things must be considered:

1. The robot dynamics and
2. the environment objects' future behaviour.

These two points are clearly considered in Definition.1 of ICS:

1. $\mathcal{A}(\tilde{u}(s,t))$ denotes the closed subset of $\mathcal{W}$ occupied by the robotic system $\mathcal{A}$ at the state reached at time t when it has executed a control trajectory $\tilde{u}$ by integrating the equation defining its dynamics (Eq.3.1).

2. $\mathcal{B}(t)$ and $\mathcal{B}_i(t)$ encodes respectively the future behaviour of all obstacles or a single obstacle in the workspace.

Now, reasoning about the future is not enough. According to the safety criteria it must be done with the ***appropriate lookahead time***. Looking again to Definition.1 we found that the time element is not upper bounded. It simply states that a time exists ($\exists t$) such that a collision will take place between the system and an obstacle in the environment at that time $\mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}(t) \neq \emptyset$. In fact, it can be deduced from this definition that if an appropriate lookahead exists for a particular environment it can be accommodated by this definition (because a collision will no occur after the appropriate t_{la}) and that in the limit the theoretical lookahead for ICS is the infinity.

Finally we have the ***decision time constraint*** that must be respected in dynamic environments. The definition of ICS doesn't give a cut clear answer for this point. Nonetheless, it give us a hint that this may be an issue because to be able to characterize the ICS set is necessary to verify all possible trajectories that the robotic system can follow from a particular state. The quantity of trajectories can be considerable for complex systems.

3.4 Motion Safety Definition

Motion safety intuitively means to have a guarantee that under some conditions, no collision will ever happen with an object in the environment. Of course, the subtlety falls on the type of conditions that are stated. They can be declared in an explicit manner when a formal statement is done or implicitly when there is a lack of such statement. As seen before, many of the navigation methods found in the robotics literature concentrate more in the advantages of their approaches than in a proper evaluation of the impact of their assumptions in the robotic system's motion safety. Although some navigation methods have shown their capacity to yield good results in a large number of circumstances they seldom give any guarantee of their performance. To change this state of affairs, a first step is to produce a set of definitions of motion safety that exploit the safety criteria that has

already been reviewed in Section 2.1.

The top level of safety can be produce if to decide its future motion a robotic system takes all safety criteria into consideration (its reasons about the future considering its own dynamics, the environment objects' future motion, reason over an appropriate lookahead and respect the time decision constraint):

Definition 2 (**Absolute Motion Safety**). *A robotic system is said to have an absolute motion safety navigation method if no collision occur whatsoever.*

When a complete model of the environment is available (complete to the point that it doesn't contain errors) then is possible to attain this level of safety. A navigation method in this level will be able to steer the robotic system through any type of environment, static or dynamic, with absolute certainty that no collision will occur.

This definition will not hold if the model of the future behaviour of obstacles contain errors. A guarantee of motion safety no matter what cannot be provided. Even if the navigation method is capable of detecting that the information provided is incorrect it may be impossible for it at that moment to find a way to avoid a collision, in other words, it may be too late. Now, saying that a model of the future is correct doesn't necessary means it has to be precise. A worst-case approximation of the future behaviour of the obstacles can be enough. As long as the provided information contains the true future obstacle's behaviour then an absolute motion safety may be attainable. In any other case, all it can be said is that the guarantee of motion safety will be *as good as the model of the future is*. This type of motion safety can be defined as:

Definition 3 (**Strong Motion Safety**). *A robotic system is said to have a strong motion safety navigation method if the no occurrence of a collision is guaranteed with respect to the model of the future which is used.*

When relaxing some of the elements of the safety criteria lower levels of safety are obtained. One example is passive motion safety.

Definition 4 (**Passive motion safety**). *A robotic system is said to have a passive motion safety navigation method if a collision happen only when the robotic system is at a complete stop.*

In the passive motion safety the safety criteria which is relaxed is the second one: no appropriate lookahead is considered. Instead the lookahead

is set to a value equal to the time it takes the system to stop from its current state. After that point in time it doesn't matter any more what happen next, the system has already reached its desired "safe" state. This type of safety may be acceptable under certain conditions. For example, if all moving participants in the environment operate under the same passive motion safety principle, then stopping is safe because all moving objects will behave exactly as the system (they will stop whenever a collision may take place). By doing so, the system will not suffer a collision when standing still. However, if not all the participants exhibit the same passive motion behaviour then a risk of a collision occurring in a future time exists and that may be unacceptable for some applications.

If further relaxations are done in the safety criteria, then we arrive at safety levels which are not longer reasonable for dynamic environments or complex system with dynamic constraints. For instance, lifting the safety criteria about reasoning over the future behaviour of the objects in the environment will result in implicitly assuming that the obstacles don't move. As explained in the motion safety analysis, assuming that the environment doesn't move can result catastrophic for the safety of the system. If the safety criteria about the robotic system considering its own dynamics is lifted, then actions that are impossible for the robotic system to perform may be taken as achievable. For example, making the decision of accelerating or decelerating faster than the robotic system constraints allow will only derive in dangerous situations.

To summarize, only when all safety criteria are taken into consideration, a complete definition of motion safety is obtained. In all other cases, only a weaker level of motion safety can exist or at the extreme case no motion safety guarantee can exist at all.

3.5 Motion Safety Level Achievable by ICS

Now, with respect to the ICS concept we have already verified that it respect most of the safety criteria (Section3.3). The only one that has been questioned is the one related to the time decision constraint. Lets assume for the moment that a navigation method is capable of efficiently using the ICS concept so it can respect also the time decision constraint. The question that should be posed now is which level of safety does ICS can achieve in that case? The answer comes from the ICS definition. As ICS needs a model of the future behaviour of the objects in the environment ($\mathcal{B}(t)$) the level of

motion safety that is achievable is the Strong Motion Safety (Def.3). That means that the guarantee of motion safety that ICS can provide will be as good as the model of the future is. Three types of model of the future can be identified:

- The first one is a deterministic model in which the outcome is precisely known. All objects in the environment will draw trajectories which follow strict relationships between inputs and outputs with no margin for error.

- A second type of representation called worst-case model is when the obstacle motion is assumed to be unpredictable. Instead of considering how the obstacle moves, it models how much it could have moved given its physical limits (*e.g.,* maximum speed). The boundary of these constraints is modeled in a geometric fashion which consequently encloses all possible future trajectories.

- Finally, the third representation reasons about the future trajectories of the obstacle as more or less probable. The probabilistic model assigns a probability measure to each of the trajectories to express the degree of belief that they will occur. Uncertainty is explicitly represented and can be properly handled with probability theory.

The focus of this work is in the first model of the future (deterministic). This choice is made because with the worst-case model the obstacles occupy the entire workspace at some point in the future (possibly infinite). Clearly this contradicts reality: obstacles can only have taken one trajectory and thus they can't occupy multiple positions at the same time. The probabilistic model requires an appropiate representation which is out of the scope of this work.

The next section will present the set of properties that render possible the ICS characterisation. They constitute the support in which any efficient method to determine the ICS-ness of a given state must be constructed. These type of methods will make possible to address the only safety criteria which still remains an issue for employing ICS in a navigation method: the time decision constraint.

3.6 ICS properties

Thanks to the definitions presented above (Section 3.2), it is possible to derive a property for the ICS characterisation through combining two properties:

Property 1 (**Control Input Intersection** [FA04]).

$$ICS(\mathcal{B}) = \bigcap_{\tilde{u} \in \tilde{\mathcal{U}}} ICS(\mathcal{B}, \tilde{u}) \tag{3.11}$$

Proof.

$$\begin{aligned} s \in \mathrm{ICS}(\mathcal{B}) &\Leftrightarrow \forall \tilde{u} \in \tilde{\mathcal{U}}, \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}(t) \neq \emptyset \\ &\Leftrightarrow s \in \mathrm{ICS}(\mathcal{B}, \tilde{u}) \\ &\Leftrightarrow s \in \bigcap_{\tilde{u} \in \tilde{\mathcal{U}}} \mathrm{ICS}(\mathcal{B}, \tilde{u}) \end{aligned}$$

□

This first property shows that $\mathrm{ICS}(\mathcal{B})$ can be derived from $\mathrm{ICS}(\mathcal{B}, \tilde{u})$ for every possible future trajectory $\tilde{u}$.

Property 2 (**Obstacles Union** [FA04]).

$$ICS(\mathcal{B}, \tilde{u}) = \bigcup_{i=1}^{n_b} ICS(\mathcal{B}_i, \tilde{u}) \tag{3.12}$$

Proof.

$$\begin{aligned} s \in \mathrm{ICS}(\mathcal{B}, \tilde{u}) &\Leftrightarrow \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}(t) \neq \emptyset \\ &\Leftrightarrow \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \bigcup_{i=1}^{n_b} \mathcal{B}_i(t) \neq \emptyset \\ &\Leftrightarrow \exists \mathcal{B}_i, \exists t, \mathcal{A}(\tilde{u}(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset \\ &\Leftrightarrow \exists \mathcal{B}_i, s \in \mathrm{ICS}(\mathcal{B}_i, \tilde{u}) \\ &\Leftrightarrow s \in \bigcup_{i=1}^{n_b} \mathrm{ICS}(\mathcal{B}_i, \tilde{u}) \end{aligned}$$

□

This second property shows that $\mathrm{ICS}(\mathcal{B}, \tilde{u})$ can be derived from $\mathrm{ICS}(\mathcal{B}_i, \tilde{u})$ for every object $\mathcal{B}_i$. The formal characterisation of the inevitable collision states is then:

Property 3 (**ICS Characterisation** [FA04]).

$$ICS(\mathcal{B}) = \bigcap_{\tilde{u} \in \tilde{\mathcal{U}}} \bigcup_{i=1}^{n_b} ICS(\mathcal{B}_i, \tilde{u}) \tag{3.13}$$

Proof.

$$\begin{aligned}\mathrm{ICS}(\mathcal{B}) &\overset{1}{=} \bigcap_{\tilde{u}\in\tilde{\mathcal{U}}} \mathrm{ICS}(\mathcal{B},\tilde{u}) \\ &\overset{2}{=} \bigcap_{\tilde{u}\in\tilde{\mathcal{U}}} \bigcup_{i=1}^{n_b} \mathrm{ICS}(\mathcal{B}_i,\tilde{u})\end{aligned}$$

□

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

Property 4 (**ICS Approximation** [FA04])**.**

$$ICS(\mathcal{B}) \subseteq ICS(\mathcal{B},\mathcal{E})$$

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

Proof.

$$\begin{aligned}\mathrm{ICS}(\mathcal{B}) &\overset{1}{=} \bigcap_{\tilde{u}\in(\mathcal{E}\cup\mathcal{E}^C)} \mathrm{ICS}(\mathcal{B},\tilde{u}) \\ &= \bigcap_{\tilde{u}\in\mathcal{E}} \mathrm{ICS}(\mathcal{B},\tilde{u}) \cap \bigcap_{\tilde{u}\in\mathcal{E}^C} \mathrm{ICS}(\mathcal{B},\tilde{u}) \\ &\subseteq \bigcap_{\tilde{u}\in\mathcal{E}} \mathrm{ICS}(\mathcal{B},\tilde{u})\end{aligned}$$

□

The following corollary establishes that the larger the subset of control trajectories, the smaller the ICS set, *i.e.,* the better the ICS approximation.

Corollary 1. *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ denote two subsets of $\tilde{\mathcal{U}}$,*

$$\mathcal{E}_1 \subset \mathcal{E}_2 \Rightarrow ICS(\mathcal{B},\mathcal{E}_2) \subseteq ICS(\mathcal{B},\mathcal{E}_1)$$

Proof.

$$\begin{aligned}\mathrm{ICS}(\mathcal{B},\mathcal{E}_2) &= \mathrm{ICS}(\mathcal{B},\mathcal{E}_1\cup\mathcal{E}_2\backslash\mathcal{E}_1) \\ &= \bigcap_{\tilde{u}\in\mathcal{E}_1\cup\mathcal{E}_2\backslash\mathcal{E}_1} \mathrm{ICS}(\mathcal{B},\tilde{u}) \\ &= \bigcap_{\tilde{u}\in\mathcal{E}_1} \mathrm{ICS}(\mathcal{B},\tilde{u}) \cap \bigcap_{\tilde{u}\in\mathcal{E}_2\backslash\mathcal{E}_1} \mathrm{ICS}(\mathcal{B},\tilde{u}) \\ &= \mathrm{ICS}(\mathcal{B},\mathcal{E}_1) \cap \mathrm{ICS}(\mathcal{B},\mathcal{E}_2\backslash\mathcal{E}_1) \\ &\subseteq \mathrm{ICS}(\mathcal{B},\mathcal{E}_1)\end{aligned}$$

□

3.7 Conclusion

The Inevitable Collision States concept address properly the problem of motion safety. They take into consideration most of the safety criteria which is required for any navigation method to operate in dynamic environments. They reason with an appropriate lookahead about the future by considering the dynamics of the system and the future motion of obstacles. However, an efficient method for their characterisation is needed to be able to cope with the decision time constraint. This is the subject of the next chapter.

Résumé

Dans le Chapitre 3, nous avons introduit le concept clé des Etats de Collisions Inévitables (ICS). Cette notion a été introduite dans le but de fournir un cadre formel de raisonnement sur les risques de collision pour un robot en mouvement.

ICS permet de considérer deux de trois critères de sécurité présentés dans le chapitre antérieur. Le dernier critère relatif au temps qui a un robot pour prendre une décision dépend de l'efficacité avec laquelle on peut caractériser les états qui appartiennent à ICS. Les définitions et les propriétés présentées dans ce chapitre nous permettent de voir que la tâche de la dite caractérisation n'est pas simple, cependant, il est possible de définir un algorithme générique qui permettrait la classification. La tâche pour résoudre est de transformer cet algorithme inefficace dans une algorithme qui est capable d'agir en temps réel.

Chapter 4

Determining Safe States

As seen in the previous chapter, Inevitable Collision States are the proper answer to address the motion safety of a robotic system operating in dynamic environments because they take into consideration the robotic system's dynamics, the future behaviour of obstacles and reason with an appropriate lookahead. However a drawback is that the characterization of ICS is not a simple task. This chapter presents an ICS-Checker algorithm, *i.e.,* an algorithm that determines whether a given state is an ICS or not, which is suited for the time decision constraint imposed by dynamic environments. It is *generic* and *efficient*. The efficiency is obtained by applying the following principles: (a) reasoning on 2D slices of the state space of the robotic system, (b) exploiting graphics hardware performances, and (c) precomputing off-line as many things as possible.

4.1 Preliminaries

4.1.1 Evasive Manoeuvres

The ICS approximation property (Property 4) and its corollary raise the question of the choice of the control trajectories that should be considered for the subset $\mathcal{E}$ in order to compute a good approximation of the true ICS set. This question is important because the quality of the approximation largely depends on the subset considered. If the approximation is too coarse, one might end up with most states being labeled as ICS (when in fact they are not).

The primary answer to that question derives from Property 3: selecting control trajectories that tend to minimize the subsets $\mathrm{ICS}(\mathcal{B}_i, \tilde{u})$ helps in minimizing $\mathrm{ICS}(\mathcal{B})$. There is however an intuitive and perhaps more

practical answer to that question: as per Def. 1, it appears that what characterize a state that is *not* an ICS is the existence of at least one control trajectory yielding a collision-free state trajectory. In this respect, the control trajectories that are important should correspond to *evasive manoeuvres*, *i.e.*, trajectories seeking to avoid collisions with the objects of the workspace. An evasive manoeuvre is formally defined as:

Definition 5 (**Evasive Manoeuvre**). *A control trajectory $\tilde{u}_e$ is an evasive manoeuvre iff $\exists s \in \mathcal{S}, \tilde{u}_e(s)$ is collision-free.*

The converse of an evasive manoeuvre is a *collision manoeuvre* which is formally defined as:

Definition 6 (**Collision Manoeuvre**). *A control trajectory $\tilde{u}_c$ is a collision manoeuvre iff $\forall s \in \mathcal{S}, \tilde{u}_c(s)$ yields a collision.*

The following property establishes that adding an evasive manoeuvre to an arbitrary subset $\mathcal{E} \subseteq \tilde{\mathcal{U}}$ is preferable to adding a collision manoeuvre since it always yields a better ICS approximation.

Property 5 (**Evasive *vs* Collision Manoeuvres**).

$$ICS(\mathcal{B}, \mathcal{E} \cup \tilde{u}_e) \subseteq ICS(\mathcal{B}, \mathcal{E} \cup \tilde{u}_c)$$

with $\mathcal{E} \subseteq \tilde{\mathcal{U}}$, $\tilde{u}_e \in \tilde{\mathcal{U}}$ an evasive manoeuvre, and $\tilde{u}_c \in \tilde{\mathcal{U}}$ a collision manoeuvre.

Proof. First, it is shown that $\mathrm{ICS}(\mathcal{B}, \mathcal{E} \cup \tilde{u}_c) = \mathrm{ICS}(\mathcal{B}, \mathcal{E})$:

$$\begin{aligned}
\mathrm{ICS}(\mathcal{B}, \mathcal{E} \cup \tilde{u}_c) &= \bigcap_{\tilde{u} \in \mathcal{E} \cup \tilde{u}_c} \mathrm{ICS}(\mathcal{B}, \tilde{u}) \\
&= \bigcap_{\tilde{u} \in \mathcal{E}} \mathrm{ICS}(\mathcal{B}, \tilde{u}) \cap \mathrm{ICS}(\mathcal{B}, \tilde{u}_c) \\
&= \mathrm{ICS}(\mathcal{B}, \mathcal{E}) \cap \mathrm{ICS}(\mathcal{B}, \tilde{u}_c) \\
&= \mathrm{ICS}(\mathcal{B}, \mathcal{E}) \cap \mathcal{S} \\
&= \mathrm{ICS}(\mathcal{B}, \mathcal{E})
\end{aligned}$$

Then, according to Corollary 1:

$$\mathcal{E} \subset (\mathcal{E} \cup \tilde{u}_e) \Rightarrow \mathrm{ICS}(\mathcal{B}, \mathcal{E} \cup \tilde{u}_e) \subseteq \mathrm{ICS}(\mathcal{B}, \mathcal{E}) = \mathrm{ICS}(\mathcal{B}, \mathcal{E} \cup \tilde{u}_c)$$

□

4.1.2 Braking and Imitating Manoeuvres

Having said that the ICS set could be approximated using evasive manoeuvres (as per Property 5), the question of actually determining the subset $\mathcal{E}$ remains. The answer to this question is not straightforward. It largely depends on the current situation, *i.e.,* the number, disposition and future behaviour of the objects present in $\mathcal{W}$. It is possible however to propose two general types of evasive manoeuvres (they will turn out to be equivalent) that can serve in most situations.

In a static environment, *braking manoeuvres*, *i.e.,* control trajectories driving $\mathcal{A}$ to a stop, are good evasive manoeuvres since, in many cases, they yield collision-free trajectories. Besides, if $\mathcal{A}$ can perform a braking manoeuvre without any collision, its safety is guaranteed forever.

Now, in the presence of moving objects, what is a good evasive manoeuvre? The answer to this question rests upon the following observation: two objects that maintain zero-relative velocity with each other will never collide in the future unless they are already in collision. In other words, the system $\mathcal{A}$ can avoid collision with a moving object $\mathcal{B}_i$ forever if it can achieve and maintain a zero relative velocity *wrt* $\mathcal{B}_i$. This observation leads to the definition of *imitating manoeuvres*, *i.e.,* control trajectories driving $\mathcal{A}$ to imitate the behaviour of a moving object $\mathcal{B}_i$.

Like braking manoeuvres that exists iff $\mathcal{A}$ is without drift, imitating manoeuvres exists iff the dynamic capabilities of $\mathcal{A}$ and $\mathcal{B}_i$ are similar. Let us consider a moving object $\mathcal{B}_i$ whose dynamics is equal to that of $\mathcal{A}$ and whose future behaviour is determined by the control trajectory $\tilde{u}_i$.

Let us assume first that $\mathcal{A}$ is in a state with zero-relative velocity *wrt* $\mathcal{B}_i$. In this case, it can start imitating the future motion of $\mathcal{B}_i$ right away (Fig.4.1-left). The imitating manoeuvre is exactly $\tilde{u}_i$ and the following property can be established:

Property 6. *ICS*$(\mathcal{B}_i, \tilde{u}_i) = \mathcal{B}_i^+(0)$ *where* $\mathcal{B}_i^+(0)$ *denotes the image in the state space* $\mathcal{S}$ *of* $\mathcal{B}_i$ *at time 0.*

Proof. Note first that $\mathrm{ICS}(\mathcal{B}_i, \tilde{u}_i) = \bigcup_{t\in[0,\infty(} \mathrm{ICS}(\mathcal{B}_i(t), \tilde{u}_i)$. It can then be deduced from (3.8) that:

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

or equivalently that:

$$\mathrm{ICS}(\mathcal{B}_i(t), \tilde{u}_i) = \{s \in \mathcal{S} | \tilde{u}_i(s,t) \cap \mathcal{B}_i^+(t) \neq \emptyset\}$$

Assuming that $\mathcal{B}_i^+(t) = \bigcup_{b_j^+ \in \mathcal{B}_i^+} b_j^+(t)$:

$$\mathrm{ICS}(\mathcal{B}_i(t), \tilde{u}_i) = \bigcup_{b_j^+ \in \mathcal{B}_i^+} \{s \in \mathcal{S} | \tilde{u}_i(s,t) = b_j^+(t)\}$$

By definition, the state s such that $\tilde{u}_i(s,t) = b_j^+(t)$ is simply $b_j^+(0)$, therefore:

$$\mathrm{ICS}(\mathcal{B}_i(t), \tilde{u}_i) = \bigcup_{b_j^+ \in \mathcal{B}_i^+} b_j^+(0) = \mathcal{B}_i^+(0)$$

and finally:

$$\mathrm{ICS}(\mathcal{B}_i, \tilde{u}_i) = \bigcup_{t \in [0,\infty(} \mathcal{B}_i^+(0) = \mathcal{B}_i^+(0)$$

□

This property establishes the fact that the ICS set yielding a collision with $\mathcal{B}_i$ for the imitating manoeuvre $\tilde{u}_i$ optimally reduces to $\mathcal{B}_i^+(0)$, *i.e.,* the set of collision states between $\mathcal{A}$ and $\mathcal{B}_i$ at time 0. In other words, unless $\mathcal{A}$ and $\mathcal{B}_i$ are already in collision at time 0, $\mathcal{A}$ can avoid collision with $\mathcal{B}_i$ forever by executing $\tilde{u}_i$.

In general, $\mathcal{A}$ will not be in a state with zero-relative velocity *wrt* $\mathcal{B}_i$. Accordingly, $\mathcal{A}$ cannot start imitating $\mathcal{B}$ right away (for instance, it does not have the proper orientation or the proper velocity). In such a situation, the imitating manoeuvre comprises two parts (Fig.4.1-right):

- The "catch-up" part at the end of which $\mathcal{A}$ achieves a zero-relative velocity with $\mathcal{B}_i$.

- The "follow" part during which $\mathcal{A}$ duplicates $\mathcal{B}_i$'s control trajectory.

As per property 6, if $\mathcal{A}$ can perform the catch-up trajectory without any collision, its safety *wrt* $\mathcal{B}_i$ is guaranteed forever. In this respect, imitating manoeuvres are good evasive manoeuvres.

Finally, it can be noticed that a braking manoeuvre is just a special imitating manoeuvre: by braking down and stopping, $\mathcal{A}$ is simply imitating the behaviour of a fixed object (which is standing still). Unlike imitating manoeuvres that are each defined *wrt* to a given moving object, braking manoeuvres are object-independent.

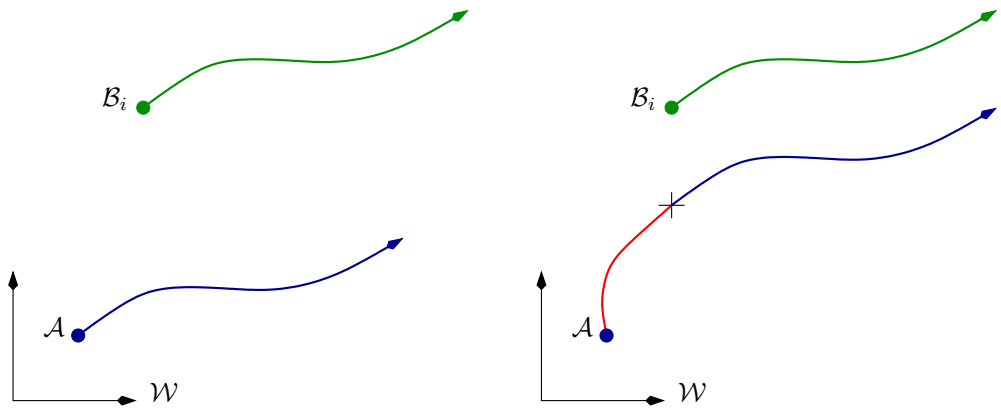


Figure 4.1: $\mathcal{A}$ imitates $\mathcal{B}_i$'s behaviour: $\mathcal{A}$ can imitate right away (left); $\mathcal{A}$ must first "catch-up" (up to the cross) before imitating (right).

4.1.3 General ICS Checking Algorithm

Properties 3 and 4 along with the principles guiding the choice of the evasive manoeuvres provide the basis for a general ICS checking scheme. The steps involved in checking whether a given state s_c is an ICS or not are given in Algorithm 1. Besides the state to be checked, the algorithm takes as input the model of the environment, *i.e.,* the list of the objects (fixed and moving) and their future behaviour (a priori known or predicted).

Algorithm 1: General ICS Checking Algorithm.

Input: s_c, the state to be checked and $\mathcal{B}_i, i = 1 \dots n_b$
Output: Boolean value

1 Select $\mathcal{E}$;
2 Compute $\mathrm{ICS}(\mathcal{B}_i, \tilde{u}_j)$ for every $\mathcal{B}_i$ and every $\tilde{u}_j \in \mathcal{E}$;
3 Compute $\mathrm{ICS}(\mathcal{B}, \tilde{u}_j) = \bigcup_{i=1}^{n_b} \mathrm{ICS}(\mathcal{B}_i, \tilde{u}_j)$ for every $\tilde{u}_j \in \mathcal{E}$;
4 Compute $\mathrm{ICS}(\mathcal{B}, \mathcal{E}) = \bigcap_{\tilde{u}_j \in \mathcal{E}} \mathrm{ICS}(\mathcal{B}, \tilde{u}_j)$;
5 Check whether $s_c \in \mathrm{ICS}(\mathcal{B}, \mathcal{E})$, return TRUE or FALSE accordingly;

The purpose of the first step of the algorithm is to select $\mathcal{E}$, *i.e.,* the set of Evasive Manoeuvres (EM) that are used to compute the conservative approximation of the ICS set. As discussed in §4.1.2, braking and imitating manoeuvres are good candidates for $\mathcal{E}$. Imitating manoeuvres do not always exist though (they require the dynamic capabilities of $\mathcal{A}$ and $\mathcal{B}_i$ to be similar). This is not a problem since arbitrary manoeuvres can be considered for $\mathcal{E}$ (as per Property 5, unless a collision manoeuvre is selected, it improves the

quality of the approximation).

Even with a limited subset $\mathcal{E}$ of EM, a naive implementation of the Algorithm 1 yields an algorithm which is computationally expensive mainly because of the cost of computing each set $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$ individually (step 2 of the algorithm) and then their union and intersection (step 3 and 4). Furthermore, the algorithmic complexity grows exponentially with the dimensionality of $\mathcal{S}$, the state space of $\mathcal{A}$. For a 2D workspace $\mathcal{W}$, it is possible however to design a generic and efficient ICS Checking algorithm. This algorithm is presented in the next section.

4.2 Ics-Check: a 2D ICS Checking Algorithm

As noted earlier, the ICS Checking algorithm outlined in §4.1.3 yields an algorithm which is computationally expensive. For a 2D workspace $\mathcal{W}$, it is possible however to design an ICS Checking algorithm which is both generic, *i.e.,* it can handle a planar robotic system $\mathcal{A}$ with arbitrary dynamics, and efficient. The efficiency of this algorithm, henceforth called Ics-Check, is primarily obtained by reasoning on 2D slices of the state space of $\mathcal{A}$ (see §4.2.1), and by determining a valid lookahead (see §4.2.2). Ics-Check is outlined in §4.2.3 and its complexity is analysed in §4.2.4.

4.2.1 2D Reasoning

The state s of $\mathcal{A}$ is a n-tuple of arbitrary dimensionality. When $\mathcal{A}$ is planar, s can be rewritten $s = (x, y, \hat{\mathbf{z}})$ with (x, y) the workspace coordinates of $\mathcal{A}$'s reference point, and $\hat{\mathbf{z}}$ the rest of the state parameters. The primary design principle behind Ics-Check is to compute the ICS set corresponding to a 2D slice of the state space $\mathcal{S}$ of $\mathcal{A}$ (instead of attempting to perform computation in the fully-dimensioned state space), and then to check if s_c belongs to this set. Assuming the state to be checked is $s_c = (x_c, y_c, \hat{\mathbf{z}}_\mathbf{c})$, the 2D slice considered is the $\hat{\mathbf{z}}_\mathbf{c}$-slice. Such a $\hat{\mathbf{z}}_\mathbf{c}$-slice is diffeomorphic to $\mathcal{W}$ and Ics-Check exploits this property to compute in a straightforward manner the image of the workspace objects in the $\hat{\mathbf{z}}_\mathbf{c}$-slice and then the corresponding ICS set. The details of this computation are given now starting with the heart of Ics-Check, namely how to compute $\text{ICS}(\mathcal{B}_i, \tilde{u}_j)$, *i.e.,* the ICS set for a given object $\mathcal{B}_i$ and a given control trajectory $\tilde{u}_j$ (step 2 of Algorithm 1).

From (3.8), it is possible to define the ICS set of the $\hat{\mathbf{z}}_\mathbf{c}$-slice considered that yields a collision with $\mathcal{B}_i$ at a particular time t for the control trajectory

$\tilde{u}_j$:

$$\begin{aligned}\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t) &= \{s \in \hat{\mathbf{z}}_\mathbf{c}\text{-slice}| \\ &\quad \mathcal{A}(\tilde{u}_j(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\}\end{aligned} \tag{4.1}$$

Fig. 4.2 illustrates how (4.1) can actually be computed by exploiting the diffeomorphism between $\mathcal{W}$ and the $\hat{\mathbf{z}}_\mathbf{c}$-slice. Starting from $s = (x, y, \hat{\mathbf{z}}_\mathbf{c})$, an arbitrary state from the $\hat{\mathbf{z}}_\mathbf{c}$-slice, the state reached at time t when $\mathcal{A}$ follows the control trajectory $\tilde{u}_j$ is $\tilde{u}_j(s,t)$. Let $s_t = (x_t, y_t, \hat{\mathbf{z}}_\mathbf{t})$ denote $\tilde{u}_j(s,t)$. s_t belongs to its own $\hat{\mathbf{z}}_\mathbf{t}$-slice which, in general, is different from the $\hat{\mathbf{z}}_\mathbf{c}$-slice. It is straightforward to show that the set of states in the $\hat{\mathbf{z}}_\mathbf{t}$-slice yielding a collision with $\mathcal{B}_i$ is $\mathcal{B}_i(t) \ominus \mathcal{A}(s_t)$ where $\ominus$ denotes the Minkowski difference. This is the standard way to compute a C-obstacle in the 2D case (see [Lat91, Chap. 3]).

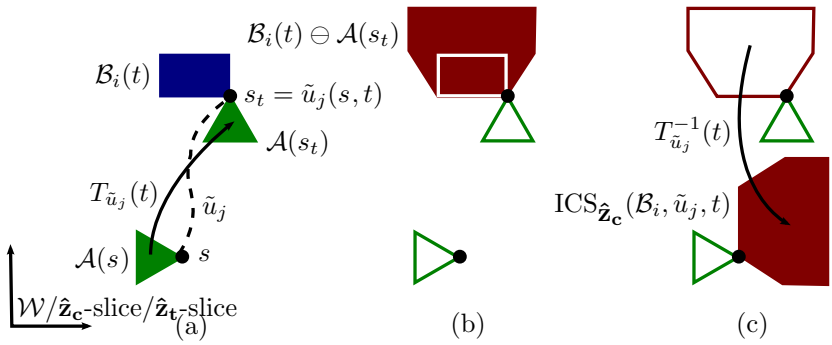


Figure 4.2: Computing $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t)$ (see text).

Now, since $\mathcal{A}$ is a rigid body moving on a plane, there is a unique geometric transformation featuring both a translation and a rotation that describes its motion in $\mathcal{W}$ from s to s_t. Let $T_{\tilde{u}_j}(t)$ denote this transformation (it is a function of both $\tilde{u}_j$ and t): $\mathcal{A}(s_t) = T_{\tilde{u}_j}(t)\mathcal{A}(s)$ and conversely $\mathcal{A}(s) = T^{-1}_{\tilde{u}_j}(t)\mathcal{A}(s_t)$. Accordingly, $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t)$, *i.e.*, the image of $\mathcal{B}_i(t) \ominus \mathcal{A}(s_t)$ in the $\hat{\mathbf{z}}_\mathbf{c}$-slice is readily obtained by applying $T^{-1}_{\tilde{u}_j}(t)$ to the set $\mathcal{B}_i(t) \ominus \mathcal{A}(s_t)$:

$$\begin{aligned}\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t) &= \{s = (x, y, \hat{\mathbf{z}}_\mathbf{c})| \\ &\quad (x,y) \in T^{-1}_{\tilde{u}_j}(t)[\mathcal{B}_i(t) \ominus \mathcal{A}(s_t)]\}\end{aligned} \tag{4.2}$$

Using property 3, it is then possible to characterize the ICS set of the $\hat{\mathbf{z}}_\mathbf{c}$-slice that yields a collision with $\mathcal{B}_i$ for $\tilde{u}_j$:

$$\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j) = \bigcup_{t \in [0,\infty(} \mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t) \tag{4.3}$$

This is the region swept by $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t)$ for all time instants. As depicted in Fig. 4.3, the general shape of $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ is a semi-infinite stripe starting with $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, 0) = \mathcal{B}_i(0) \ominus \mathcal{A}(s)$.

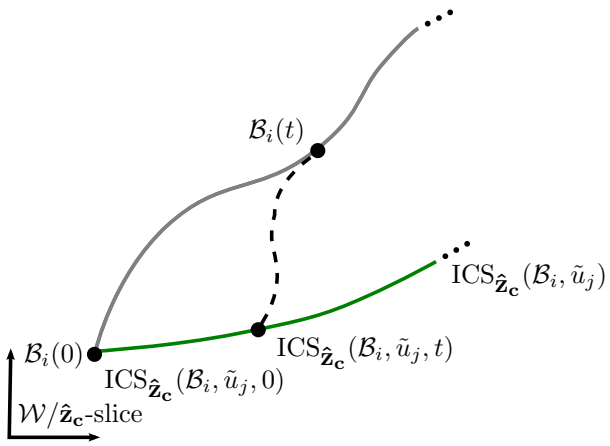


Figure 4.3: General shape of $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ in the case where both $\mathcal{A}$ and $\mathcal{B}_i$ are points.

4.2.2 Valid Lookahead

One distinctive feature of the ICS concept is that it considers trajectories $\tilde{u}_j$ of infinite duration (see Def. 1). Accordingly it has an infinite lookahead and it is this infinite lookahead that guarantees motion safety. This infinite lookahead explicitly appears in the union over an infinite time interval of the right-hand side of (4.3). Now, the question is: how can (4.3) be computed? It turns out that this union can be restricted to a finite time interval provided that (i) $\mathcal{W}$ is bounded and (ii) each moving object $\mathcal{B}_i$ eventually leaves $\mathcal{W}$. If these two (reasonable) assumptions are met then it can be shown that, at some point into the future, $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t)$ becomes and remains either empty or constant. At that point, the union can stop.

Property 7 (Lookahead). *If $\mathcal{W}$ is bounded and each moving object $\mathcal{B}_i$ exits $\mathcal{W}$ at time t^i_{exit} then $\forall \tilde{u}_j, \exists t_{\max}$ such that $\forall t \geq t_{\max}, ICS_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t) = ICS_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t_{\max})$.*

Proof. Four cases are considered depending on the type of $\mathcal{B}_j$ (fixed *vs* moving) and the type of $\tilde{u}_j$ (braking *vs* imitating):

1. *Fixed $\mathcal{B}_i$, braking manoeuvre $\tilde{u}_j$:*

If $\mathcal{B}_i$ is fixed then $\mathcal{B}_i(t)$ is constant, $\forall t$. If $\tilde{u}_j$ is a braking manoeuvre then $\exists t_b$ such that $\forall t \geq t_b, \tilde{u}_j(s,t) = \tilde{u}_j(s,t_b)$. t_b is the time it takes $\mathcal{A}$ to come to a stop. Then, the terms in the right-hand side of (4.1) all become constant at time t_b and so does $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t)$. In this case, $t_{\max} = t_b$.

2. *Moving $\mathcal{B}_i$, braking manoeuvre $\tilde{u}_j$:*

   If $\mathcal{B}_i$ is moving then $\mathcal{B}_i(t)$ becomes empty at time t^i_{exit} and, as per (4.1), so does $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t)$. In this case, $t_{\max} = t^i_{\text{exit}}$.

3. *Fixed $\mathcal{B}_i$, imitating manoeuvre $\tilde{u}_j$:*

   If $\tilde{u}_j$ is an imitating manoeuvre (see §4.1.2) then it eventually drives $\mathcal{A}$ outside of $\mathcal{W}$. Let t^j_{exit} denotes the time when $\mathcal{A}$ exits $\mathcal{W}$, $\mathcal{A}(\tilde{u}_j(s,t))$ becomes empty at time t^j_{exit} and, as per (4.1), so does $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t)$. In this case, $t_{\max} = t^j_{\text{exit}}$.

4. *Moving $\mathcal{B}_i$, imitating manoeuvre $\tilde{u}_j$:*

   Similar to the cases 2 and 3 above. In this case, $t_{\max} = \min(t^i_{\text{exit}}, t^j_{\text{exit}})$.

□

Thanks to Property 7, (4.3) can finally be rewritten:

$$\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j) = \bigcup_{t \in [0, t_{\max}]} \mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j, t) \tag{4.4}$$

4.2.3 Ics-Check Algorithm

Applying the 2D reasoning principle, Ics-Check is similar to the general ICS Checking Algorithm detailed in Algorithm 1 except that, in all steps of the algorithm, $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}$ is computed instead of ICS (see Algorithm 2). It is by keeping all computations in 2D (notwithstanding the actual dimensionality of $\mathcal{S}$) that it is possible to efficiently compute the ICS set corresponding to a given $\hat{\mathbf{z}}_\mathbf{c}$-slice.

4.2.4 Complexity Analysis

Assuming a discrete time model and that $\mathcal{A}$ and the objects $\mathcal{B}_i$ are modeled as convex polygons, the complexity of Ics-Check is $O(n_b n_e t_{\max} m)$ where n_b is the number of objects, n_e the number of evasive manoeuvres, $t_{\max}$ the valid lookahead derived from Property 7, and $m = m_a + m_b$ with m_a the number

Algorithm 2: ICS-CHECK.

Input: s_c, the state to be checked and $\mathcal{B}_i, i = 1 \dots n_b$
Output: Boolean value

1 Select $\mathcal{E}$;
2 Use (4.4) to compute $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ for every $\mathcal{B}_i$ and every $\tilde{u}_j \in \mathcal{E}$;
3 Compute $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_j) = \bigcup_{i=1}^{n_b} \mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ for every $\tilde{u}_j \in \mathcal{E}$;
4 Compute $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E}) = \bigcap_{\tilde{u}_j \in \mathcal{E}} \mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_j)$;
5 Check whether $s_c \in \mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$, return TRUE or FALSE accordingly;

of vertices of $\mathcal{A}$ and m_b the number of vertices of the most complex object $\mathcal{B}_i$. The next section describes a numerical implementation of ICS-CHECK whose complexity is further reduced to $O(n_b n_e t_{\max})$ thanks to the use of standard graphics rendering techniques that can be hardware accelerated.

4.3 Ics-Check: an Efficient Implementation

The implementation of ICS-CHECK presented in this section primarily derives its efficiency from the use of standard graphics rendering techniques that can be hardware accelerated (see §4.3.1) and, to a lesser extent, to the possibility of pre-computing off-line a number of things (see §4.3.2).

4.3.1 Exploiting Graphics Rendering Techniques

Focusing on the computation of the steps 2, 3 and 4 of ICS-CHECK (see Algorithm 2), it can be seen that it is necessary to perform the union and the intersection of regions. For arbitrary regions, such operations are costly to implement however, because the regions considered are planar, it turns out that it is possible to do such unions and intersections with OpenGL[1] primitives and thus to benefit from the processing power of standard Graphics Processing Unit (GPU).

All the steps of ICS-CHECK that involves computing unions and intersections of arbitrary 2D shapes are performed very efficiently by drawing in the OpenGL buffer the regions corresponding to the different 2D sets and by taking advantage of the Red-Green-Blue (RGB) colour coding scheme. The key idea is to assign a RGB colour to each Evasive Manoeuvre (EM) $\tilde{u}_j \in \mathcal{E}$ and to draw the different $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ by performing a bitwise logical

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

AND ($\bigvee$) at the pixel level between the EM colour and the current OpenGL buffer colour (initialized to White, *i.e.,* #FFFFFF). Let col_j denote the colour assigned to $\tilde{u}_j$, the colour assignment is done so as to satisfy the following property:

$$\bigvee_{\tilde{u}_j \in \mathcal{E}} col_j = \#000000 \text{ and } \bigvee_{\tilde{u}_j \in \mathcal{E}' \subset \mathcal{E}} col_j \neq \#000000 \tag{4.5}$$

where #000000 is the Black colour and $\mathcal{E}'$ is an arbitrary subset of $\mathcal{E}$. With such a colour assignment, drawing a region over a region of the same colour (or White) yields the same colour (thus performing a union), whereas drawing a region over a region of a different colour yields another colour (thus performing an intersection). If a particular pixel of the OpenGL buffer is Black, it means that all the EM of $\mathcal{E}$ have contributed to its colour. In other words, it is an ICS.

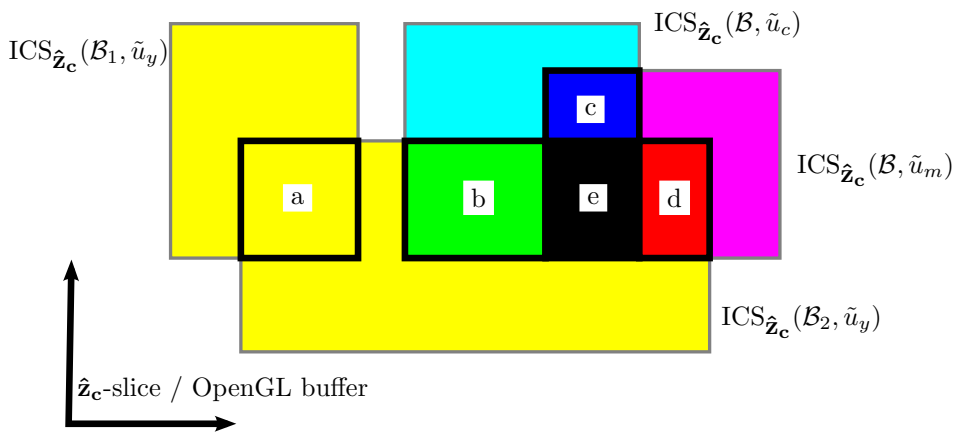


Figure 4.4: Using graphics rendering techniques to compute $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$ (see text).

This mechanism is illustrated in Fig. 4.4 with an example featuring three EM, *i.e.,* $\mathcal{E} = \{\tilde{u}_y, \tilde{u}_c, \tilde{u}_m\}$, and two obstacles, *i.e.,* $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2\}$. The respective colours of $\tilde{u}_y$, $\tilde{u}_c$ and $\tilde{u}_m$ are Yellow (#FFFF00), Cyan (#00FFFF) and Magenta (#FF00FF). Note that such a colour assignment satisfies (4.5). Fig. 4.4 depicts the OpenGl buffer (equivalent to the $\hat{\mathbf{z}}_\mathbf{c}$-slice). Assuming $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_1, \tilde{u}_y)$ and $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_2, \tilde{u}_y)$ are drawn first (they are represented by rectangular regions). By virtue of the logical AND operator, their overlapping region indicated by "a" remains Yellow. $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_y)$, *i.e.,* the union between $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_1, \tilde{u}_y)$ and $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}_2, \tilde{u}_y)$, is therefore achieved. When $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_c)$ is drawn, its region of overlap with $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_y)$ turns green (#00FF00). This

green region indicated by "b" is the intersection between $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_c)$ and $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_y)$. A similar thing happens when $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_m)$ is drawn. Three new regions appear: (i) a Blue (#0000FF) one indicated by "c" which is the intersection between $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_m)$ and $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_c)$, (ii) a Red (#FF0000) one indicated by "d" which is the intersection between $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_m)$ and $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_y)$ and (iii) a Black one indicated by "e" which is the intersection between $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_c)$, $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_y)$ and $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_y)$. This black region is in fact $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$.

4.3.2 Precomputing As Much As Possible

In addition to the above principle, it is possible to further improve the algorithm efficiency by precomputing off-line as many things as possible. Two items are candidates for precomputation, namely the subset $\mathcal{E}$ of Evasive Manoeuvres (step 1 of the algorithm) and the set $\mathrm{ICS}(\mathcal{B}_i, \tilde{u}_j)$, *i.e.,* the set of ICS for a given object $\mathcal{B}_i$ and a given manoeuvre $\tilde{u}_j$ (step 2 of the algorithm). Note that it is not always possible to precompute these two items since they depend on the problem at hand and the particulars of the future behaviour of the moving objects.

4.4 Ics-Check At Work

To illustrate how the algorithm works it has been applied to four different robotic systems. They were selected to exemplify how Ics-Check can handle in a graceful manner the increasing dimensionality of the state space by keeping the reasoning in 2D slices. In the next sections we introduce first the models that describe the motion of the robotic systems employed in the remaining of the document. Then, we present the environment where the systems were placed among static and moving obstacles. Next, we focus in one specific system where we detail the technique employed to compute the evasive manoeuvres and the steps of Ics-Check to verify a given state. Finally, we show similar results for the rest of the systems.

4.4.1 Robotic Systems

The systems under consideration are (1) a point-mass system, (2) a differential-drive system, (3) a car-like vehicle and (4) a spaceship with translational momentum. They all have a circular shape with radius R and their motion is modeled as follows:

Point-Mass System

Let $\mathcal{A}$ be modeled with point mass non-dissipative dynamics. A *state* of $\mathcal{A}$ is defined as $s = (x, y, v_x, v_y)$ where (x, y) are the coordinates of the center of the disk and v_x, v_y are the axial components of the velocity. A control of $\mathcal{A}$ is defined by the pair (u_x, u_y) which denote the force exerted by the actuators along the x- and y-axis respectively. The motion of $\mathcal{A}$ is governed by the following differential equations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v_x} \\ \dot{v_y} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} u_x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_y \tag{4.6}$$

with a bound in the control given by the maximum acceleration: $\frac{u_x^2+u_y^2}{m^2} \leq a_{\max}^2$ where m is the robot mass.

Differential-Drive System

Is composed of two active fixed wheels with a common axle. The system $\mathcal{A}$ is characterized by a 5 dimensional state $s = (x, y, \theta, v_l, v_r)$ where (x, y) are the coordinates of the system reference point (placed equidistantly from the wheels on the axle), θ is the angle between the x-axis of the $\mathcal{W}$ and the robot longitudinal axis, and v_l, v_r denote the left and right wheel velocities respectively. A control of $\mathcal{A}$ is defined by the pair (u_l, u_r), where u_l (resp. u_l) is the acceleration of the left (resp. right) wheel. Let $2b$ be the distance between the system wheels. The motion of the system is governed then by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v_l} \\ \dot{v_r} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} (v_r + v_l)\cos\theta \\ (v_r + v_l)\sin\theta \\ \frac{(v_r - v_l)}{b} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} u_l + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_r \tag{4.7}$$

with a bound in the wheel velocities and control inputs:

$$|v_l|, |v_r| \leq v_{\max} \text{ and } |u_l|, |u_r| \leq u_{\max}$$

Car-like System

The system $\mathcal{A}$ moves like a car-like vehicle. It has a state defined as a 5-tuple $s = (x, y, \theta, v, \xi)$ where (x, y) are the coordinates of the center of its rear wheels axle, θ is the heading angle between the body axis and $\mathcal{W}$'s x-axis,

v is the linear velocity and ξ is the orientation of the front wheels (steering angle). $\mathcal{A}$'s control is denoted by the couple (u_α, u_ξ) where u_α is the linear acceleration of the rear wheels and u_ξ is the steering angle velocity. If L is the distance between the location of (x, y) and the middle point of the driving wheels axle then the system moves accordingly to the following equations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\xi} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ v\tan\xi/L \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} u_\alpha + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_\xi \tag{4.8}$$

with constraints:

$$v \in [0, v_{\max}], |\xi| \leq \xi_{\max}, |u_v| \leq u_{\alpha_{\max}} \text{ and } |u_{xi}| \leq u_{\xi_{\max}}$$

Spaceship System

The spaceship system $\mathcal{A}$ is capable of thrusting forward or rotating in either direction. The rotation has no effect on its translation which is subject to momentum. A state of $\mathcal{A}$ is defined as the 6-tuple $s = (x, y, \theta, v_x, v_y, \omega)$ where (x, y) is the spaceship reference point coordinates, θ is the main orientation, v_x is the speed of the spaceship along the x-axis, v_y the speed along the y-axis, and ω is the angular speed. A control of $\mathcal{A}$ is the couple (u_α, u_τ) where u_α denotes the linear acceleration along the ship main orientation and u_τ denotes the angular acceleration. The motion of $\mathcal{A}$ is described by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ \omega \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} u_\alpha + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_\tau \tag{4.9}$$

where

$$|v_x|, |v_y| \leq v_{\max}, |\omega| \leq \omega_{\max}, u_\alpha \in [0, u_{\alpha_{\max}}] \text{ and } |u_\tau| \leq u_{\tau_{\max}}$$

4.4.2 Workspace Model

The environment features one system $\mathcal{A}$ moving in a simulated 2D workspace $\mathcal{W}$ cluttered up with an initial number of obstacles n_b, static and dynamic. The number of obstacles in the environment will decrease as obstacles exit

the workspace when crossing its boundary (dotted line in Fig. 4.5). Neither a new obstacle nor one that has already exit $\mathcal{W}$ is allowed to traverse the boundary to get inside. All obstacles are disk or polygonal-shaped. Knowledge about their future behaviour is available up to an infinite time $t \in [t_0, \infty]$. A maximum valid lookahead time t_{max} can be derived using Property 7. Their trajectories are described by cubic B-Spline functions with maximum curvature constraint. Thus, an object's reference point coordinate (x, y) at time t is obtained by evaluating its trajectory B-Spline function $\mathbf{x}(t)$. The object's velocity vector has constant magnitude and is defined as the derivative $\mathbf{v}(t) = \frac{d\mathbf{x}}{dt}$ which is tangent to the curve at the position $\mathbf{x}(t)$. The obstacle orientation is therefore set as the angle between $\mathbf{v}(t)$ and $\mathcal{W}$'s x-axis. Figure 4.5 illustrates the type of scenario considered and the obstacle's trajectories.

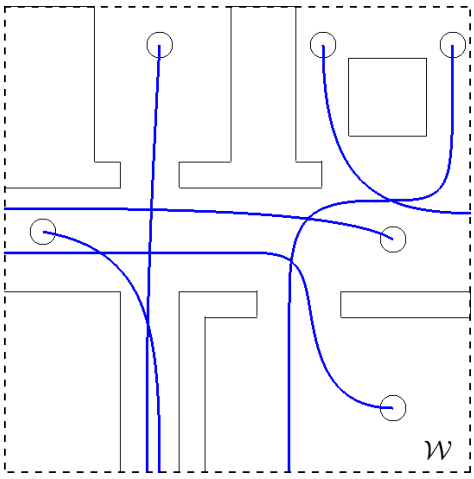


Figure 4.5: Sample scenario with moving obstacle's trajectories in blue lines.

4.4.3 Car-Like System Case Study

Control Law for Evasive Manoeuvres

To compute the set of Evasive Manoeuvres for the Car-like vehicle we use the results of the trajectory tracking problem in the extensive literature in control theory. An exhaustive explanation of the control law is out of the scope of this work (interested readers are referred to [MS08]). Nonetheless the general formulation of the problem is as follows. Given a reference trajectory $\tau : t \longrightarrow (x_r(t), y_r(t), \theta_r(t))$ and a reference robot configuration $(x(t), y(t), \theta(t))$, the objective is to stabilize the error vector $(x_r(t) - x(t), y_r(t) - y(t), \theta_r(t) - \theta(t))$ at zero. The control technique employed reformulates the problem in terms of

a Serret-Frenet frame of reference, *i.e.,* the origin of the rectilinear coordinate system is determined by the curve abscissa $s(t)$ and the orthonormal basis are tangent and normal to the curve. In this "moving" frame, a more suitable non-linear tracking error model can be obtained which allows to locally linearize the system. The linearisation is done through a transformation of the system into its chained form. Once linearised a feedback can be applied that renders the system globally asymptotically stable to the origin, that is, to the equilibrium (when the errors are equal to zero).

We chose to include in our set of EM a fixed set of Braking Manoeuvres and one Imitating Manoeuvre per moving obstacle in the environment. The reference trajectory τ that serve as input for the control law is $\mathcal{B}_i$'s trajectory function translated to the workspace coordinates of the state to be checked s_c.

Scenario #1

In the scenario considered here for the car-like model, the workspace features 6 disk-shaped moving objects and 6 polygon-shaped static ones. Let $s_c = (5, -1, -1.0, 10, 0)$ denote the state to be checked for ICS-ness. ICS-CHECK first computes the ICS set for the corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice, $\hat{\mathbf{z}}_\mathbf{c} = (-1.0, 10, 0)$, and then checks whether s_c is an ICS by finding out the color of the $(5, -1)$ pixel in the $\hat{\mathbf{z}}_\mathbf{c}$-slice. Fig. 4.6 depicts the corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice. It features the 12 isotropically grown objects plus a red circle which center point corresponds to s_c. The line at that point is the orientation $\theta = -1.0$ of s_c. The blue lines attached to each moving object represents their trajectory in the $\hat{\mathbf{z}}_\mathbf{c}$-slice.

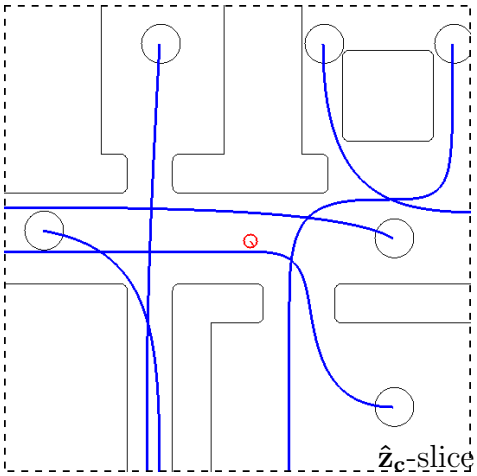
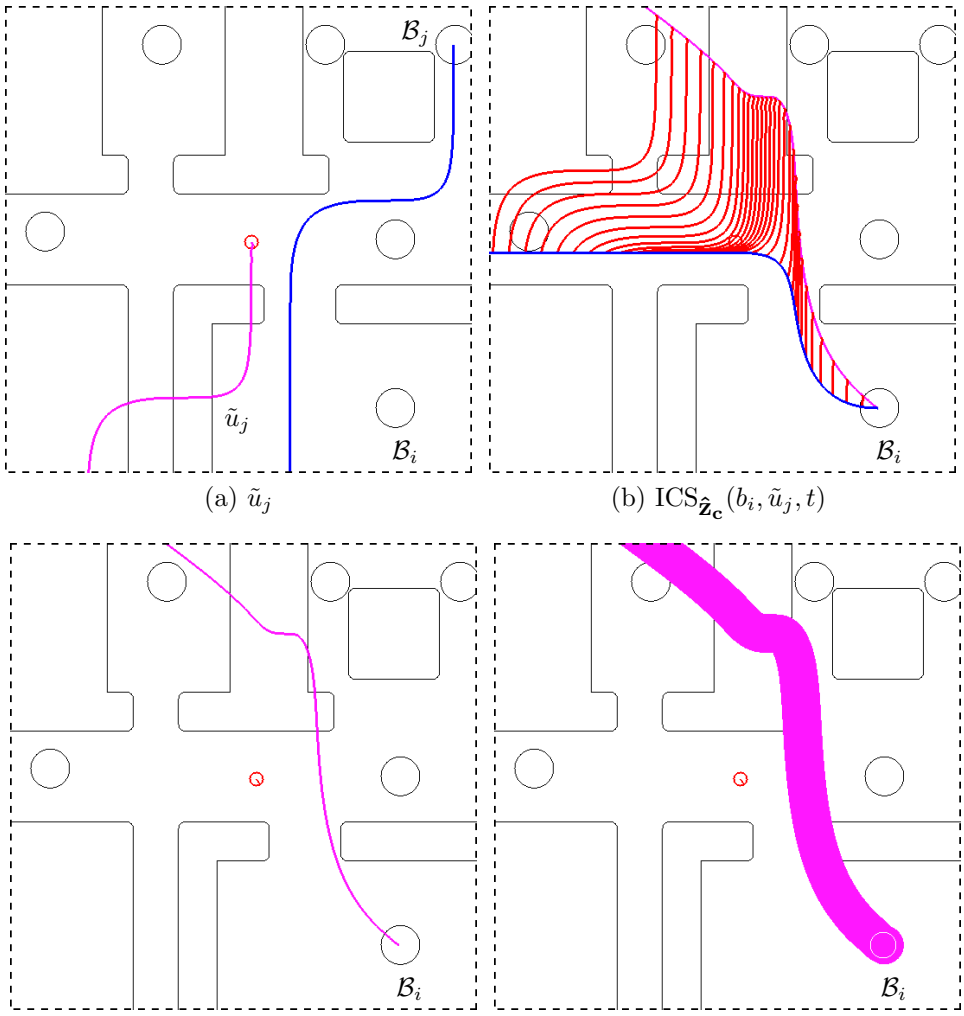


Figure 4.6: Scenario #1 (car-like vehicle).

Fig. 4.7 illustrates how ICS-CHECK computes $\mathrm{ICS}(\mathcal{B}_i, \tilde{u}_j)$, *i.e.,* the set

of ICS corresponding to a given EM $\tilde{u}_j$ and a given object $\mathcal{B}_i$. Fig. 4.7a depicts $\tilde{u}_j$ in pink. It is the imitating manoeuvre corresponding to the object $\mathcal{B}_j$ in the upper-right corner which is moving along its trajectory shown in blue. Fig. 4.7b is the equivalent of Fig. 4.3: it shows how $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(b_i, \tilde{u}_j, t)$ is computed where b_i is the center of the disk object $\mathcal{B}_i$ located on the lower-right corner and moving along its trajectory shown in blue. Fig. 4.7c depicts $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(b_i, \tilde{u}_j)$ for all $t \in [t_0, t_{max}]$ where t_{max} is the lookahead time. Fig. 4.7d depicts the set $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ (step 2 of ICS-CHECK).



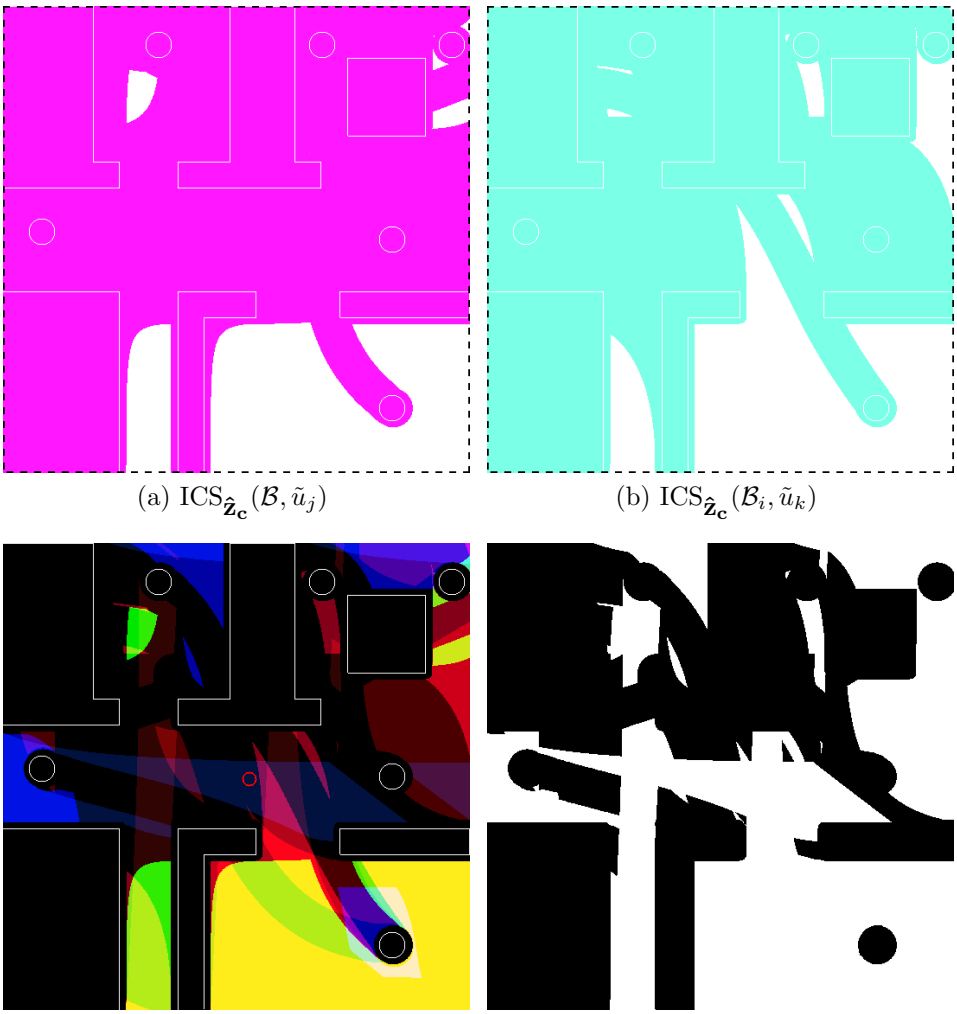
(a) $\tilde{u}_j$ (b) $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(b_i, \tilde{u}_j, t)$

(c) $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(b_i, \tilde{u}_j)$ (d) $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$

Figure 4.7: Computing $\mathrm{ICS}(\mathcal{B}_i, \tilde{u}_j)$ (see text).

This procedure is repeated for every object yielding $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_j)$

(Fig. 4.8a) (step 3 of ICS-CHECK). Then, it is repeated for every EM. Fig. 4.8b depicts the set $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_k)$ corresponding to another EM $\tilde{u}_k$. At the end of the day, the drawing of the different $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_j)$, $\tilde{u}_j \in \mathcal{E}$ on the same buffer yield the final ICS set $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$ (step 4 of ICS-CHECK). The final step of ICS-CHECK simply consists in checking the colour of s_c. If it is black, s_c is an ICS otherwise it is not (step 5 of ICS-CHECK). In the present case, s_c is not an ICS.



(a) $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_j)$ (b) $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_k)$

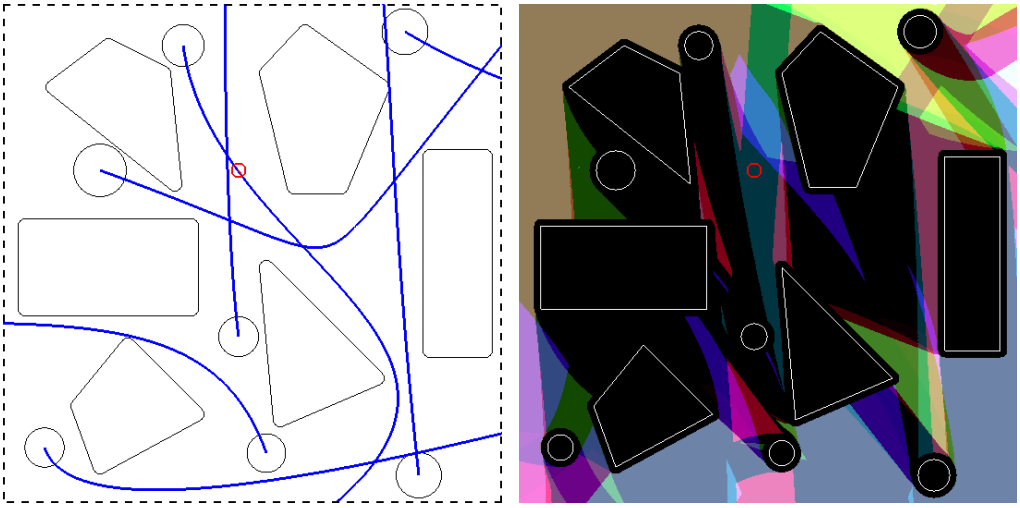
(c) Final set $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$ (in black) (d) $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$

Figure 4.8: Computing $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$ (see text).

4.4.4 Other Examples

Scenario #2 - point-mass system

In this scenario the workspace features 7 disk-shaped dynamic objects and 6 polygon-shaped static ones. The state of the point-mass system to check is $s_c = (-5, 30, 0, 0)$. Figure 4.9a shows the corresponding $\hat{\mathbf{z}}_\mathbf{c}$-slice:$\hat{\mathbf{z}}_\mathbf{c} = (0, 0)$ and Figure 4.9b the final ICS set $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$. In this case the state is not an ICS.



(a) $\hat{\mathbf{z}}_\mathbf{c}$-slice point-mass system (b) $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$

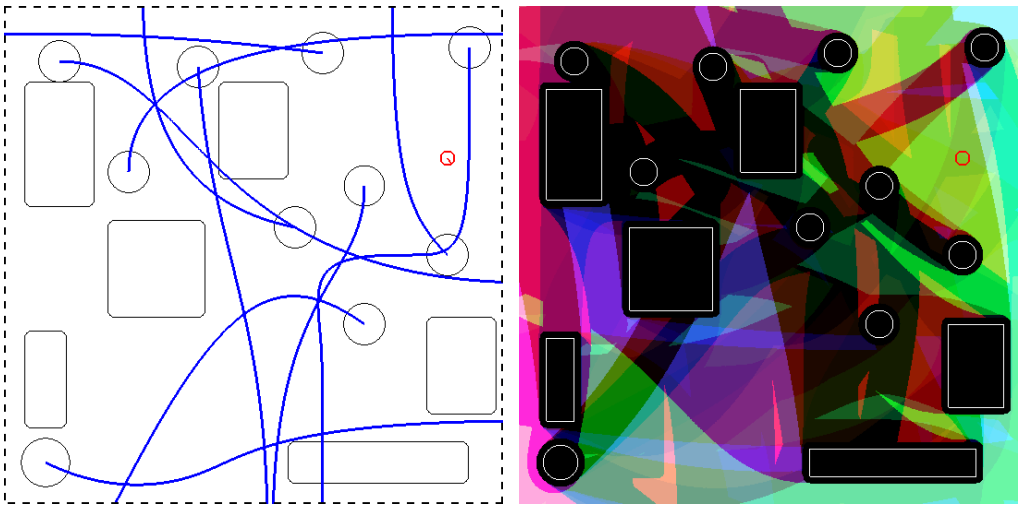
Figure 4.9: Scenario #2

Scenario #3 - differential-drive system

The workspace contains 10 disk-shaped dynamic and 6 square-shaped static objects. The differential-drive system has a state to be checked $s_c = (70, 35, -1.2, 4, 4)$. Note how the dimensionality of the state space has been increased to 5D but the reasoning remains in 2D. For s_c its $\hat{\mathbf{z}}_\mathbf{c}$-slice is equal to $\hat{\mathbf{z}}_\mathbf{c} = (-1.2, 4, 4)$ (Fig. 4.10a). The final ICS set $\text{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$ is shown in Fig. 4.10b. Again the state is not an ICS.

Scenario #4 - spaceship system

Our workspace contains only 3 static objects and 6 dynamic ones. The spaceship system is in a difficult situation, trying to exit from a "corridor" created by the static obstacles, unfortunately the exit is blocked by an



(a) $\hat{\mathbf{z}}_\mathbf{c}$-slice differential-drive system (b) $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$

Figure 4.10: Scenario #3

incoming obstacle. The system state is $s_c = (0, 0, -1, 0.2, -1, 0)$. Note we have a 6D state space. The $\hat{\mathbf{z}}_\mathbf{c}$-slice that corresponds to the situation is $\hat{\mathbf{z}}_\mathbf{c} = (-1, 0.2, -1, 0)$ (Fig. 4.11a). The final ICS set $\mathrm{ICS}_{\hat{\mathbf{z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$ is shown in Fig. 4.11b. This state is an ICS.

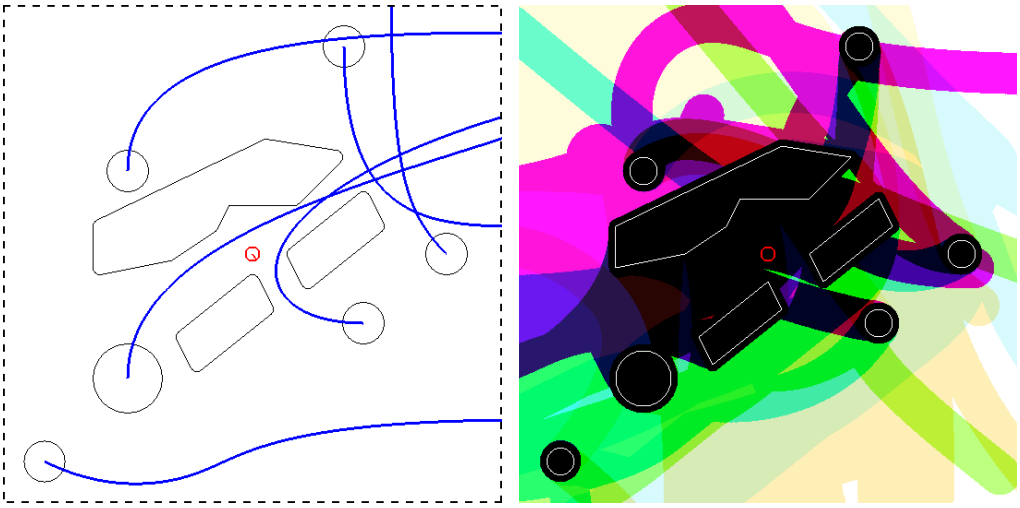
4.4.5 Ics-Check Performances

The running time complexity of Ics-Check is that of Section 4.2.4. It depends on the number of objects n_b, number of evasive manoeuvres n_e and the lookahead time t_{max}. The running times of the current implementation of Ics-Check[2] has been measured using Scenario #1. Table 4.1 reports the results obtained. They are satisfactory especially since there is still room for improvement.

4.5 Conclusion

This chapter has presented an Inevitable Collision States checking algorithm, *i.e.,* an algorithm that determines whether a given state for a robotic system is an ICS or not. The algorithm presented is *generic* and *efficient*. It can be used for planar robotic systems with arbitrary dynamics moving in dynamic

[2] `C++` implementation on an average PC computer: Intel Core2 Duo CPU 2.66GHz, 3 GB RAM, and Nvidia GeForce 8800 GTX GPU.



(a) $\hat{\mathbf{z}}_{\mathbf{c}}$-slice differential-drive system (b) $\mathrm{ICS}_{\hat{\mathbf{Z}}_{\mathbf{c}}}(\mathcal{B}, \mathcal{E})$

Figure 4.11: Scenario #4

Table 4.1: ICS-CHECK Performance evaluation.

$t_{max} = 25$(s),$\Delta t = 160$(ms),$n_e = 11$	n_b	Time (ms)
OBJECTS	13	28.4
	14	30.9
	15	32.4
	16	35.7
	17	39.10
$t_{max} = 25$(s),$\Delta t = 160$(ms),$n_b = 17$	n_e	Time (ms)
EM	7	22.5
	8	25.3
	9	30.4
	10	36.5
	11	39.1
$t_{max} = 25$(s),$n_e = 11$,$n_b = 17$	Δt(ms)	Time (ms)
Δt	10	453.5
	20	237.6
	40	126.9
	80	72.8
	160	39.10

environments. The efficiency is obtained by applying the following principles: (a) reasoning on 2D slices of the state space of the robotic system, (b) exploiting graphics hardware performances, and (c) precomputing off-line as many things as possible.

Once the ICS-ness of a state can be easily verified the next step is to employ this information within a collision avoidance scheme. That will be the subject of the next chapter.

Résumé

Ce chapitre a présenté une algorithme de vérification des ICS, *i.e.,* un algorithme qui détermine si un état donné pour un système robotisé est un ICS ou non. L'algorithme présenté est générique et efficace. Il peut être utilisé pour des systèmes robotisés plats avec une dynamique arbitraire en se déplaçant dans des environnements dynamiques. L'efficacité est obtenue en appliquant les principes suivants : (a) raisonnement sur des tranches 2D de l'espace d'état du système robotisé, (b) exploitation de fonctionnement de la carte graphique et (c) pre-calculant autant de choses que possible.

Une fois que un état peut être facilement vérifié l'étape suivant doit employer ces informations dans un système d'évitement de collision. Ce sera le sujet du chapitre suivant.

Chapter 5

Navigation

In general a navigation method intends to fulfill two key objectives: progress towards a given goal and avoid collision with the objects of the environment. Regardless of the navigation method employed, the success of the collision avoidance objective is conditioned by the fact that the robotic system at hand can not be in an ICS state. In other words, if the system's state is an ICS it means that it does not matter anymore what it does it will inevitable collide with an object of its environment. In contrast, the objective of reaching a given goal is not conditioned to the ICS-ness of the state. As strange as this may appear at first, it simply means that the system which current state is an ICS may still be able to momentarily reach its goal to only then continue to an inevitable collision with an obstacle of its environment. Depending on the application requirements this may be acceptable or not. From a motion safety point of view is clear that reaching a goal to later collide with something else is not acceptable. Consequently, a successful safe navigation method must operate in a way that takes the robotic system from one non-ICS state to the other. To do it, such navigation method would need first to have the capability of determining if a chosen state belongs to the ICS set. The Ics-Check presented in the previous chapter is exactly the kind of tool needed for that characterization. Once a state has been characterized as not being an ICS, is easy to show from the definition of ICS that at least one collision-free trajectory exists. This trajectory can be used as an emergency evasive manoeuvre in case no other collision-free solution is found by the navigation method. This chapter describe an algorithm called Ics-Avoid that precisely operates in that way. It is designed to alternate its operation with Ics-Check and in doing so, it can build a set of controls that guarantee that the robot will be able to find a safe transition between non-ICS states. In consequence it guarantees the motion safety of the system with respect to the model of the future which is employed.

5.1 ICS-Avoid

5.1.1 Overview

The navigation methods reviewed in Chapter 2 operate in different ways to avoid collisions with the objects of the environment. Despite their remarkable differences, they all share a same operating principle. They define a set of rules or a procedure to characterize forbidden regions in the control space of the robotic system at hand. These regions must be avoided by the navigation method when selecting a control among the set of admissible ones that the robotic system can execute. The selection of the control may be conditioned to additional rules to consider other type of objectives like progressing towards a goal, smoothness in path, optimality, etc., but in general, any allowed control must provide at least the capacity to avoid collisions. Additionally the methods must operate in the shortest possible time to respond to the changing nature of the dynamic environments (remember the time decision constraint). Thus, it become impractical with the available time to perform a complete characterization of the forbidden regions if the whole control space is considered. A common alternative employed by the methods to circumvent this difficulty is to resort to some sort of sampling strategy. Is widespread practice to either discretize the control space or to rely in a randomized search of the control space. The flaw in this kind of strategy is that a valid control may not be found at all in the selected subset of the control space (*e.g.,* if the discretization is too coarse). If the navigation method fails to find a control is not clear what it should be done. It is simply assumed that no solution exists. In contrast to these navigation methods, Ics-Avoid provides explicitly a mechanism to fall back in case the sampling strategy fails to find a valid control. This fall-back mechanism is henceforth called the Safe Control Kernel. This kernel guarantee that if a state has been determined by the Ics-Check as not being an ICS then an admissible control (one that takes the system from the current state to another non-ICS state) will be an element of it. With the safe control kernel it doesn't matter if the sampling strategy fails because a valid control will be readily available to be used. Now, for the kernel to work two condition must be met. The first is that the current state is not an ICS, and the second is that a transition between non-ICS states should be possible. This transition constitute the core of Ics-Avoid, *i.e.,* navigate the environment by transitioning from one non-ICS state to the next one until the goal is reached. The transition between non-ICS states is always possible as shown with the following property:

Property 8 (Safe Transition between non-ICS states).

$$\forall s \notin ICS(\mathcal{B}, \mathcal{E}), \exists s_i = \tilde{u}_j(s, t_i) \mid \tilde{u}_j \in \mathcal{E}, s_i \notin ICS(\mathcal{B}) \tag{5.1}$$

Proof.

$$\begin{aligned}
s \notin \mathrm{ICS}(\mathcal{B}, \mathcal{E}) \quad &\Leftrightarrow \quad \exists \tilde{u}_j \in \mathcal{E} \mid \forall t > t_0, \mathcal{A}(\tilde{u}_j(s,t)) \cap \mathcal{B}(t) = \emptyset \\
&\Rightarrow \quad \exists \tilde{u}_j \in \mathcal{E} \mid \forall t \in [t_0, t_i], \mathcal{A}(\tilde{u}_j(s,t)) \cap \mathcal{B}(t) = \emptyset \\
&\qquad \wedge \forall t > t_i, \mathcal{A}(\tilde{u}_j(\tilde{u}_j(s,t_i),t)) \cap \mathcal{B}(t) = \emptyset \\
&\Rightarrow \quad \exists \tilde{u}_j \in \mathcal{E} \mid \forall t > t_i, \mathcal{A}(\tilde{u}_j(s_i,t)) \cap \mathcal{B}(t) = \emptyset \\
&\Rightarrow \quad s_i \notin \mathrm{ICS}(\mathcal{B}, \mathcal{E}) \\
&\Rightarrow \quad s_i \notin \mathrm{ICS}(\mathcal{B}) \\
&\Rightarrow \quad \exists s_i = \tilde{u}_j(s, t_i) \notin \mathrm{ICS}(\mathcal{B})
\end{aligned}$$

□

This property simply states that from a non-ICS state is always possible to make a transition to another non-ICS state. Furthermore it implies that a control trajectory $\tilde{u}_j$ in the subset of control trajectories $\mathcal{E}$ contains the sequence of controls to make this transition possible. This information can be actively exploited to reduce the number of controls which must be searched to find the one that allows a safe transition. This reduced set of controls is what will constitute the Safe Control Kernel.

5.1.2 Safe Control Kernel

From a conceptual point of view a Safe Control Kernel is a small set of commands which contains at least one control that allows a transition between the current state of the robotic system to a non-ICS state. Formally the Safe Control Kernel is defined as follows:

Definition 7 (Safe Control Kernel).

$$K(s) = \{K \subset \mathcal{U} \mid \exists u \in K, s_i \notin ICS(\mathcal{B}), s_i = s + \int_t^{t_i} f(s,u)dt \ \} \tag{5.2}$$

where $f(s, u)$ is the function that describe the dynamics of the robotic system (Eq.3.1).

Logically the safe control kernel is empty if the current state is an ICS, *i.e.,* no control exists which can take the robotic system from an ICS state to a non-ICS state.

Property 9 (Empty Safe Control Kernel).

$$s \in ICS(\mathcal{B}) \Rightarrow K(s) = \emptyset \tag{5.3}$$

Proof.

$$\begin{aligned} s \in \mathrm{ICS}(\mathcal{B}) &\Leftrightarrow \forall \tilde{u}_j \in \tilde{\mathcal{U}}, \exists t, \mathcal{A}(\tilde{u}_j(s,t)) \cap \mathcal{B}(t) \neq \emptyset \\ &\Rightarrow \forall \tilde{u}_j \in \tilde{\mathcal{U}}, s_i \in \mathrm{ICS}(\mathcal{B}), s_i = \tilde{u}_j(s, t_i) \\ &\Rightarrow \neg \exists u \in \mathcal{U} \mid s_i \notin \mathrm{ICS}(\mathcal{B}), s_i = s + \int_t^{t_i} f(s,u)dt \end{aligned}$$

□

Property 8 have already showed that when the contrary is true, *i.e.,* when the state does not belong to the ICS set, the control kernel can not be empty: from a non-ICS state is always possible to arrive to another non-ICS state. Two important conclusions can be obtained from these properties. The first one is that a navigation method can only be successful if the current state of the system is not an ICS. The second conclusion is that the information provided by Ics-Check is vital to determine if a solution exists to the navigation problem. In addition, actively checking if a state is an ICS can probe to be useful in determining the set of controls that can be included in the safe control kernel. The intuition of this assertion is that the evasive manoeuvres used in the Ics-Check contains a set of controls which have already been tested for collision. Now, with this in mind what remains to be explained is how this alternation process between verifying the ICS-ness of a state and the filling mechanism of the safe control kernel can work to produce the desired results.

5.1.3 Augmenting Ics-Check

Algorithm 2 must be modified to allow an external method to retrieve the set of control trajectories that were selected internally to compute the ICS approximation that makes possible to characterize a given state as ICS or not. The modification is quite simple. In particular the output of the algorithm is augmented to include the information of the control trajectories. It must be noted that the selection of the control trajectories that form the subset $\mathcal{E}$ is not modified in anyway by Ics-Avoid. The choice is done within Ics-Check. As was shown in Section 4.1.2 the type of evasive manoeuvres that serve in most situations are *braking manoeuvres* and *imitating manoeuvres*. The braking manoeuvres are only a special case of the imitating ones but conceptually it makes sense to do a distinction between the static and dynamic elements of the environment. The modified version of Algorithm 2 is as follows:

Algorithm 3: Augmented ICS-CHECK.

Input: s_c, the state to be checked and $\mathcal{B}_i, i = 1 \dots n_b$
Output: Boolean value, $\mathcal{E}$

1 Select $\mathcal{E}$;
2 Use (4.4) to compute $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ for every $\mathcal{B}_i$ and every $\tilde{u}_j \in \mathcal{E}$;
3 Compute $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_j) = \bigcup_{i=1}^{n_b} \mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}_i, \tilde{u}_j)$ for every $\tilde{u}_j \in \mathcal{E}$;
4 Compute $\mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E}) = \bigcap_{\tilde{u}_j \in \mathcal{E}} \mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \tilde{u}_j)$;
5 Check whether $s_c \in \mathrm{ICS}_{\hat{\mathbf{Z}}_\mathbf{c}}(\mathcal{B}, \mathcal{E})$, return TRUE or FALSE accordingly and $\mathcal{E}$;

5.1.4 Ics-Avoid Algorithm

The modified version of the general ICS-CHECK algorithm is employed to verify the ICS-ness of the current state of the system. The output of this verification constitute the input of ICS-AVOID. If the state is characterized as not an ICS then ICS-AVOID can be used. The procedure starts with the computation of the set of controls to be included in the safe control kernel. The safe control kernel will include only the first entries of each control trajectory that was returned by the augmented ICS-CHECK. This choice of including only the first elements, *i.e.,* for a time t_i $\tilde{u}(t_i) \forall \tilde{u} \in \mathcal{E}$, is to have a minimum number of elements in the subset. Once the kernel has been built in this way is ready to be included in the control space sampling set $\mathcal{J}$ that is used by ICS-AVOID. The sampling strategy can be anything but the important thing to keep in mind is that this sampling set includes already the elements of the safe control kernel. The next step is simply to select a control of the sampling set and integrate the equations that describe the dynamics of the robotic system to obtain the state that will be reached when applying the control. This state is verified for ICS-ness with Algorithm 2 which will in turn determine if the control is safe or not (if it belongs or not to the ICS set). If the state reached is not ICS the procedure can finish successfully. Otherwise a new control of the sampling set must be picked up and the checking procedure must be done one more time. As the sampling set includes the controls of the kernel eventually the procedure will end in a successful manner. Thus a control that allows a transition between non-ICS states is guaranteed to be found. The ICS-AVOID algorithm is recapitulated in a concise form as follows:

Algorithm 4: ICS-AVOID.

Input: s_i (the current state of $\mathcal{A}$) and $\mathcal{E}$
Output: Control u

1 Compute Safe Control Kernel $K_i = \bigcup_j \tilde{u}_j(t_i)$, $\tilde{u}_j \in \mathcal{E}$;
2 Build control space sampling set: $\mathcal{J} \subset \mathcal{U}$ and include K_i into it;
3 Select control $u \in \mathcal{J}$;
4 Compute $s_{i+1} = s_i + \int_{t_i}^{t_{i+1}} f(s_i, u)dt$;
5 **if** ICS-CHECK(s_{i+1})= *True* **then**
6 Go to 3
7 **else**
8 SUCCESS. Return u
9 **end**

5.1.5 Sampling strategies

ICS-AVOID is not restricted to a particular sampling strategy for the selection of a control (Steps 2 and 3 of the Algorithm 4). However, the choice of this sampling strategy influence directly the result of the navigation process. To illustrate the type of strategies that can be used two examples will be used. The first one is the most basic behaviour. Survive in the environment. The second illustrate how strategies of other navigation methods can be employed directly with ICS-AVOID. Specifically the strategy presented in the Dynamic Window Approach[FBT97] will be employed.

Surviving in the environment

This strategy illustrate the most basic behaviour that can be obtained with ICS-AVOID. Survive in the environment means that the only objective of the robotic system is to avoid collision with the objects of its environment. No interest to reach a goal, follow a path, respond to specific events, etc. To achieve this kind of behaviour it suffice to equal the sampling set to the safe control kernel ($\mathcal{J} = K$) and to sequentially pick the elements of the safe control kernel in step 3. In the worst-case scenario all the elements of the control kernel will need to be checked to find a valid control. Once a control is found it is selected immediately and applied by the system. This basic strategy can be used in conjunction with more complex ones to account directly for constraints like the time decision constraint. The time of execution of this strategy can be easily computed as it depends on the cardinality of the safe control kernel.

Target Heading, clearance and velocity

The Dynamic Window Approach(DWA) presented in [FBT97] is popular in the robotics community given its relatively simplicity and the acceptable results obtained with it. One of its features is the objective function σ which is maximized in order to select the control employed among the not forbidden set of velocities. The objective function takes into consideration three factors. The target heading (measure of progress towards the goal), the clearance (distance to the closest obstacle on the trajectory) and the velocity (it supports faster movements). By combining these three factors the result is that the robotic system circumvent as fast as it can the objects in the environment while still progressing towards the goal. It should be noticed that to obtain the value that maximize this function a discretization of the resulting search space (V_r) is made (the intersection between the admissible velocities and the velocities inside the dynamic window). This kind of successful strategy to select a control can be seamlessly integrated into Ics-Avoid. Step 2 of the algorithm allows the flexibility to define the sampling set $\mathcal{J}$ as it may be desired by the user. Following the DWA example, $\mathcal{J}$ can be equaled to the discretized V_r and the objective function σ computed to each of the elements of $\mathcal{J}$. Once their resulting value is obtained from the function, the elements can be put in a list in descending order. Step 3 then would simply select the elements of the mentioned ordered list is a sequentially manner.

In a similar manner other type of sampling strategies can be used. All that is need to be done is to identify two things:

1. the underlying sampling set used in the navigation method, and
2. the function or mechanism used to select the element to apply.

Once this two features are identify, Ics-Avoid can integrate them into steps 2 and 3. This flexibility makes of Ics-Avoid a versatile method which can be integrated in existing applications.

Now, to evaluates and compare the performance of Ics-Avoid a benchmark was performed with two state-of-the-art navigation methods designed to operate in dynamic environments and presented in Section 2.2.2. The first one is Time-Varying Dynamic Window (TVDW) [SP07]), the second one is Non Linear Velocity Obstacle (NLVO) [LLS05]. To be just, Ics-Avoid was employed in its basic version, *i.e.,* the survive in the environment strategy. The scenario employed and the results obtained are shown in the next section.

5.2 Benchmarking Ics-Avoid with Other Navigation Methods

As explained previously, the benchmarking concerns TVDW, NLVO and ICS-AVOID. The first two are extensions to popular navigation methods used in real-world applications: Dynamic Window (DW) and Velocity Obstacles (VO). DW has been demonstrated at relatively high speeds (up to 1 m/s) in complex environments with Minerva [TBB+99], Rhino [BCF+00] and Robox [PS03], robotic tour-guides that have operated for different time periods in different places in the United States, Germany and Switzerland. VO has been tested with MAid [PSF01], an automated wheelchair that navigated in the concourse of the central station in Ulm (DE) and during the German exhibition Hanover Fair'98. TVDW and NLVO were detailed in Section 2.2.2. Nonetheless, a brief description of their operation mode is done here for completeness.

TVDW (p.33) is an approach where the search for admissible controls is carried out directly in the linear and angular velocity space. As in DWA, the search space is reduced by the robotic system's kinematic and dynamic constraints to a set of reachable velocities (V_r) in a short time interval (Δt) around the current velocity vector. A velocity is admissible (V_a) if it allows the system to stop before hitting an object. The main difference with respect to DWA is that instead of considering the environment as static TVDW calculates at each instant a set of immediate future obstacles trajectories (see Figure.2.6). This trajectories are used to check for collision in the short term, *i.e.,* the lookahead time is set to the time it takes to robotic system to come to a full stop.

NLVO (p.36) is an approach that operates in the Cartesian velocity space of the robotic system. Each object in the environment yields a set of forbidden velocities that would yield a collision between it and the robotic system $\mathcal{A}$. Should the robotic system select a forbidden velocity, it would collide with the moving object at a later time. As opposed to VO that only consider constant linear velocities profiles, NLVO considers known arbitrary velocity profiles for the moving objects. NLVO consist of all velocities of $\mathcal{A}$ at t_0 that would result in collision with $\mathcal{B}$ at any time $t_0 \leq t \leq t_{la}$. Geometrically $NLVO(t)$ is a scaled down $\mathcal{B}$, bounded by the cone formed between $\mathcal{A}$ and $\mathcal{B}(t)$. In consequence, NLVO is a warped cone with apex at $\mathcal{A}$ (see Figure.2.8).

5.2.1 Simulation Setup

A simulation environment capable of reproducing the same conditions and providing the same information for all the navigation methods was chosen to conduce the benchmarking. The robotic system selected, the workspace model setup and the implementation details are discussed in the next sections.

Robotic System: Point Mass Model

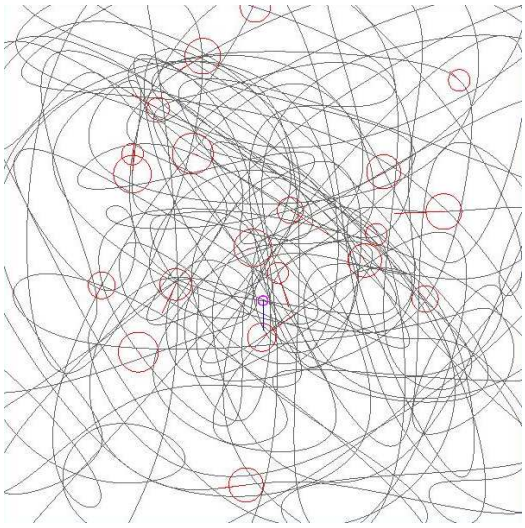
Let $\mathcal{A}$ be modeled as a disk with point mass non-dissipative dynamics. A state of $\mathcal{A}$ is defined as $s = (x, y, v_x, v_y)$ where (x, y) are the coordinates of the center of the disk and v_x, v_y are the axial components of the velocity. A control of $\mathcal{A}$ is defined by the pair (u_x, u_y) which denote the force exerted by the actuators along the x- and y-axis respectively. The radius of $\mathcal{A}$ is $r_{\mathcal{A}}$ and its motion is governed by the following differential equations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v_x} \\ \dot{v_y} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} u_x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_y \tag{5.4}$$

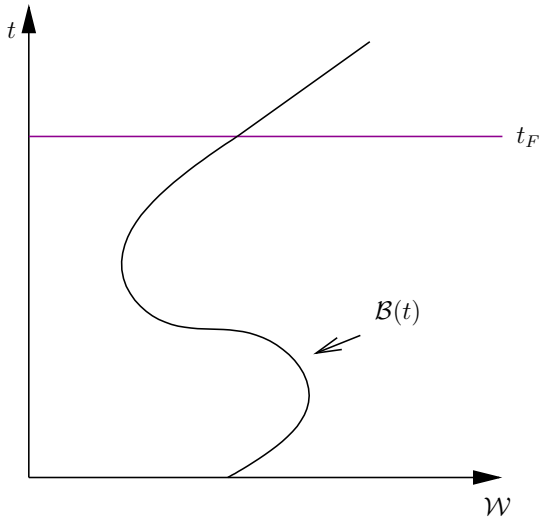
with a bound in the control given by the maximum acceleration: $\frac{u_x^2+u_y^2}{m^2} \leq a_{max}^2$ where m is the robot mass.

Workspace Model

$\mathcal{A}$ moves in a closed 2D workspace $\mathcal{W}$ (100 by 100 meters), cluttered up with disk-shaped moving objects (grown by the radius of $\mathcal{A}$). A total of twenty three objects move in complex cyclic trajectories (trajectories defined by closed B-splines with 10 random control knots). The objects move with random constant speeds (between 1 to 10 m/s) along their trajectories. Figure 5.1a shows the trajectories of the objects to illustrate the complexity of the environment. This setup can theoretically provide future information about the behaviour of the moving objects up to infinity. In practice, knowledge about the future behaviour of the moving objects is provided until a fixed time in the future t_F after which constant linear motion is assumed (Fig. 5.1b). This is done to resemble realistic scenarios where the prediction quality degrades as time pass moves farther in the future.



(a) Workspace example, 23 obstacles (represented by circles) along their trajectories defined by 10 random control knots B-Splines.



(b) World Model of the Future.

Figure 5.1: Benchmark Workspace.

Implementation

The simulation environment and navigation methods were programmed entirely in C++ using OpenGL as rendering engine. The ICS-Checker presented in Chapter 4 was integrated to perform efficient state checking for ICS-AVOID. To achieve an identical reproduction of simulation conditions for each of the collision avoidance schemes in the benchmark, the random number generator was seeded with a set of identical numbers. Each seed identified a run of the simulation. The information about the future behaviour of the objects in the environment was made available to all the schemes with a values of 1, 3 and 5 seconds into the future.

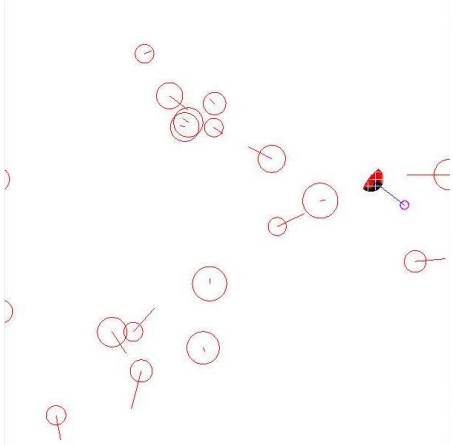
Benchmark Results

The navigation methods were tested on a set of five runs. Each run had a duration of two minutes. The amount of available information about the future behaviour of the obstacles in the environment was varied with values t_F =1, 3 and 5 seconds. For each run and each value of t_F the number of collisions between the system $\mathcal{A}$ and the objects $\mathcal{B}_i$ are summarized in Table 5.1.

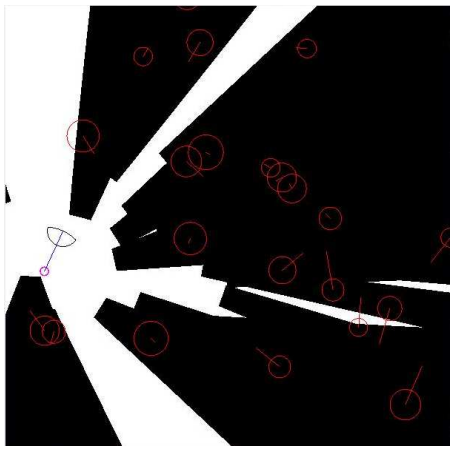
Scheme	Run	Collisions t_F=1(s)	Collisions t_F=3(s)	Collisions t_F=5(s)
TVDW	1	5	6	3
	2	12	4	4
	3	5	7	3
	4	12	2	4
	5	12	2	4
Average:		9.2	4.2	3.6
NLVO	1	10	2	0
	2	8	2	0
	3	12	2	0
	4	3	3	2
	5	7	2	2
Average:		8.0	2.2	0.8
ICS-AVOID	1	7	0	0
	2	0	0	0
	3	1	0	0
	4	1	0	0
	5	1	0	0
Average:		2.0	0.0	0.0

Table 5.1: Benchmarking of navigation methods.

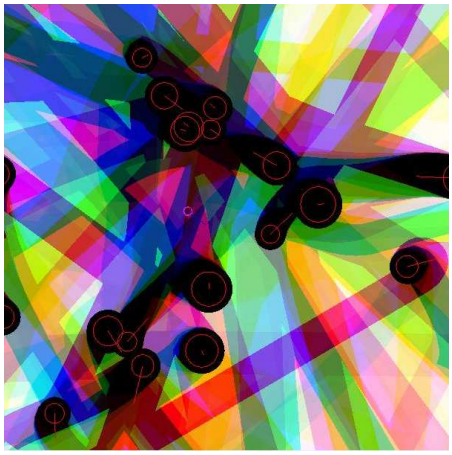
TVDW (Fig. 5.2a) performs poorly in comparison with the other two schemes. One of the main causes of failure is the limited extent in which the scheme use the information available about the future trajectories of the objects: as explained before it limits itself to a small fraction of the time at hand (its lookahead times is limited to the time it takes the system to stop). In contrast, NLVO (Fig. 5.2b) exploits better the given information. In these runs t_{la} was set equal to t_F so all the available information could be taken into account. NLVO averages less of one collision per run in the 5 second setup, nonetheless, it fails to guarantee the safety of the system when provided with less information. ICS-AVOID (Fig. 5.2c) has the best performance in all the time setups. ICS-AVOID is designed to reason in terms of infinite duration but even when dealing with minimal information about the future (1 second) it outperformed the other two schemes. When given more information (3 and 5 seconds) not a single collision occurred. The results show the importance of the lookahead time, when a collision avoidance scheme disregard available information its performance is lower compared to those that use more.



(a) TVDW. Admissible velocities (V_a) are represented in black, velocities in red are forbidden.



(b) NLVO. Black warped cones are forbidden velocities for the robotic system.



(c) ICS-Avoid. Black regions are forbidden states (ICS).

Figure 5.2: Benchmark Navigation Methods.

5.2.2 Conclusion Benchmarking

To demonstrate the efficiency of ICS-AVOID, it has been extensively compared with two state-of-the-art navigation methods, both of which have been explicitly designed to handle dynamic environments. Like ICS-AVOID, TVDW and NLVO make assumptions of the way the environment unfold in the future and therefore they are also susceptible to uncertainty. But if ICS-AVOID was provided with full knowledge about the future, it would guarantee motion safety no matter what. Of course, given the elusive nature

of the future, this assumption is somewhat unrealistic. In practice, knowledge about the future is limited. However, the results obtained show that, when provided with the same amount of information about the future evolution of the environment, ICS-AVOID performs significantly better than the other two schemes. The first reason for this has to do with the respective lookahead of each collision avoidance scheme thus emphasizing the fact that, reasoning about the future is not nearly enough, it must be done with an appropriate lookahead. The second reason has to do with the decision part of each collision avoidance scheme. In all cases, their operating principle is to first characterize forbidden regions in a given control space and then select an admissible control, *i.e.,* one which is not forbidden. Accordingly motion safety also depends on the ability of the collision avoidance scheme at hand to find such admissible control. In the absence of a formal characterization of the forbidden regions, all schemes resort to sampling (with the inherent risk of missing the admissible regions). In contrast, ICS-AVOID through the concept of *Safe Control Kernel* is the only one for which it is guaranteed that, if the current state is not an ICS an admissible control will be part of the sampling set.

5.3 Conclusion

This chapter had presented ICS-AVOID, an ICS-based navigation method designed to handle complex dynamic environments. ICS-AVOID takes into account the behaviour of the moving objects to reason about the future and thanks to the *Safe Control Kernel* provides a mechanism to find an admissible control to move safely from one non-ICS state to the other. The results obtained in a comparative evaluation with two state-of-the-art collision avoidance schemes show that, when provided with the same amount of information about the future evolution of the environment, ICS-AVOID outperforms them. The first reason for this has to do with the extent to which each collision avoidance scheme reasons about the future. The second reason has to do with the ability of ICS-AVOID to find a safe control in the Safe Control Kernel by alternating its operation with ICS-CHECK.

Résumé

Ce chapitre avait présenté ICS-AVOID, une méthode de navigation basée en ICS conçue pour être utilisé dans des environnements dynamiques complexes. ICS-AVOID prend en compte le comportement des objets mobiles pour

raisonner sur le futur. ICS-AVOID fournit un mécanisme (*Safe Control Kernel*) qui permet constater qui existe un contrôle admissible pour se déplace sans risque d'un état de non-ICS à l'autre. Les résultats obtenus dans une évaluation comparative avec deux systèmes d'évitement de collision montrent que, quand fourni avec la même quantité d'informations sur l'évolution future de l'environnement, ICS-AVOID les surpasse. La première raison pour cela a un rapport avec la mesure à laquelle chaque système d'évitement de collision raisonne sur l'avenir. La deuxième raison a un rapport avec la capacité de ICS-AVOIDpour trouver un contrôle sûr dans le *Safe Control Kernel*.

Chapter 6

Experimental Results

This chapter presents results in addition to those already described in previous chapters. The reason is that until now all the results included only simulations and not experiments with a real platform. The main reason for having it done that way is that the conditions in simulated environments are easily controlled and that situations that can not be tested in the real world can be done in the simulated one. This advantage allows to show the limits of the capacity of what is proposed. However, experiments with a real platform provide elements to understand the proposed methods that would otherwise not be discovered if only simulation tests were done. The reality is much more complex than any simulation could reach to imagine or reproduce. This chapter then is the description of the efforts done to produce a practical implementation of Ics-Check. As with any experiment involving navigation in a real platform, the presented methods need a cloud of components that provide the functionality that was assumed previously to be easily obtainable. All the assumptions made earlier clash with the need to translate them into software and hardware entities.

The next sections began with a description of the experimental platform that was used in the tests. The general structure in its hardware and software architectures is developed. Then, the specific software components that fill the void made by the assumptions previously done in the navigation process are detailed. Afterwards, the actual experiment and its result is shown. Finally an analysis of the obtained result and the conclusion reached close the chapter.

6.1 Experimental Platform

The description of the experimental platform is divided into two parts. The first refers to the hardware components of the robotic system: an autonomous wheelchair. The second describes the middleware that serves to coordinate all the software components required for the navigation task of a robotic system.

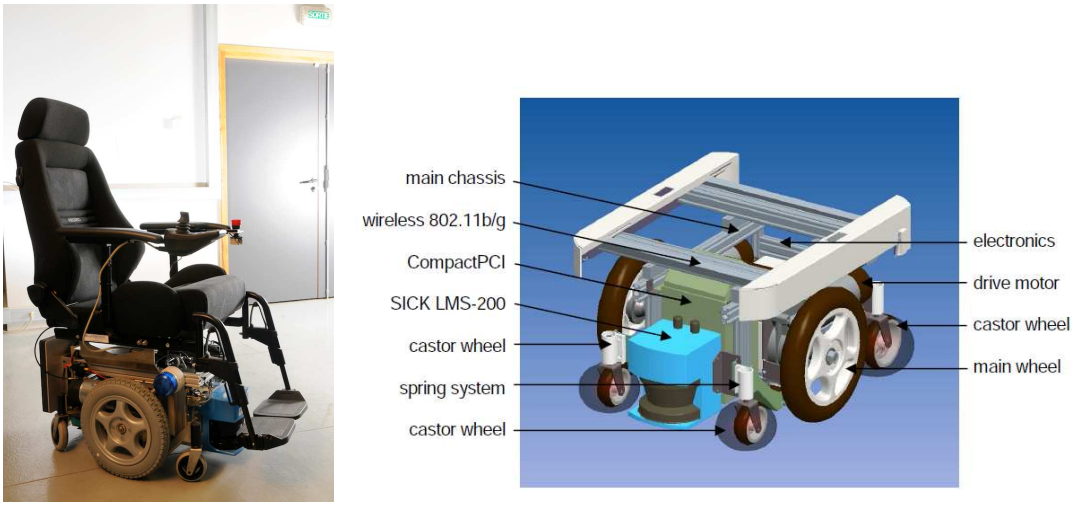
6.1.1 Hardware

The robotic platform employed for the experiments is an autonomous wheelchair manufactured and marketed by BlueBotics (Figure 6.1a). The platform is a modular system issue from the European funded project MOVEMENT (Modular Versatile Mobility Enhancement Technology). The core of the device consists of a mobile base (Figure 6.1b) with all the on-board electronics. To this base different assistive devices can be attached, being the simplest one a seat to transport persons.

The mobile base has a rectangular footprint with dimensions of 0.56 m long by 0.67 m wide. The total height of the robotic platform (including the attached seat) is of 1.30 m. The autonomous wheelchair can carry a maximum payload of 150 kg with a maximum nominal linear velocity of 1.39 m/s and a maximum rated angular velocity of 1.5 rad/s. Its maximum acceleration is rated at 1.35 m/s^2.

Sensors aboard the wheelchair consist of a LIDAR (Light Detection and Ranging) model SICK LMS-200, wheel based quadrature encoders for odometry measurements and emergency bumpers sensors (contact switchs). The odometry is updated at a rate of 100 Hz while the laser scanner works at 37Hz with a field of view of 180 °, resolution of 0.5 ° (361 beams) and a range set to 16 m.

The motion of the wheelchair can be controlled in two ways. Manually via a joystick mounted on the seat armrest or automatically through commands sent by a embedded computer mounted in the mobile base. The embedded computer hardware is based in the CompactPCI industrial system. The on-board electronics provide as devices for communication a 4-port Ethernet switch and a WI-FI 802.11b/g board. Using these network devices the wheelchair can transmit to external computers its sensor information and receive motion commands to execute. In the setup used a laptop with a Corei7 processor at 2.2GHz with 4GB RAM and running as OS an Ubuntu 10.04 64-bits served as the hardware to execute the navigation software described in the following section.



(a) Autonomous Wheelchair (b) Mobile base

Figure 6.1: Robotic Platform.

6.1.2 Software Architecture

The software architecture is based in ROS [QGC+09]. ROS stands for two things, Robot Operating System and Robot Open Source. The first makes reference to the analogy in the type of functionality it provides like hardware abstraction, low-level device control, message-passing between processes and package management. However ROS is not an operating system in the traditional sense of process management and scheduling. In fact is a layer above the hosts operating systems of an heterogeneous computer cluster. Although is intended to be cross-platform its only well supported in Unix-like system such as Ubuntu Linux (other OS like Mac OS X and Windows have "experimental" status). The latter acronym describe the open source nature of ROS. All in it is completely open source (BSD license) and free for others to use. The idea is to enable code reuse across the research community for breaking the time-consuming cycle of writing code from scratch. To that end, reusable components that implement various levels of functionality are available. Examples include mapping, visual odometry, data visualization, object recognition, etc. All this components come from contributions from a growing number of institutions (currently around 30) that collaborate in a federated development model. ROS is not constrained to be used in a particular robot. Many research and commercial robots are already supported. Robots which are not supported like the BlueBotics Autonomous Wheelchair used in the experiments need only to implement

relatively easy communication nodes to expose sensor readings and receive commands. In summary ROS is a well-supported robotic application development environment and a repository of community contributions that provide components with robotics related functionality.
the A ROS based system consists of several processes (potentially distributed in several computers) connected at runtime in a P2P (peer-to-peer) topology. In this P2P network or computation graph there are *nodes* (that perform the computation) and *messages* or links between nodes (to exchange data).

Each *node* is responsible of providing a particular functionality. Examples are data sources (like nodes retrieving sensor information: LaserScan, Odometry, Camera, GPS, etc), data processors (nodes that receive data, process it and produce other kind of information, *e.g.,* sensor fusion from multiple laser scans) and data consumers (nodes that eat data, typically actuators, logging and visualization, *e.g.,* motor control and data graph display). There are a last type of nodes which role consists in the organization, coordination and controlling of the computation graph. They provide the information about which nodes exist in the system and what type of data they consume/produce, they provide a way to share common parameters and configuration options and they maintain information of the current topology of the graph and computer network in distributed systems.

The *messages* or links are responsible for the exchange of data between nodes. A message is simply a data structure comprising typed fields. Primitive types, arrays or nested structures are supported. Messages are routed via two types of transport systems paradigms. The *topics* which use a publish/subscribe model and the *services* which implement a request/reply type of communication. Topics are useful when the same type of data is share among many nodes. For example, laser scans may be used by several nodes (obstacle detection, mapping, tracking, etc) so the laser driver node can publish a topic to the graph which all nodes interested in this type of information can subscribe. A single node, may publish and/or subscribe to multiple topics. Services are designed to respond to a different kind of need. Frequently the data is needed only once or at very low rates. The request/reply paradigm is ideal for this. When a node needs certain information it simply request the service and a node that have the information can reply to call with the information.

The software architecture provided by ROS allows a decoupled operation of the different components needed by a navigation method. Each component will be responsible of providing a certain functionality and thus it can be

wrapped in a node. To exchange data with other nodes all it needs to do is publish/subscribe to topics or request/reply services. The solution to provide navigation capabilities to a robot consist then in the choice of the components that respond to the particular needs of the problem at hand. The approach taken in this thesis is described in the next section.

6.2 Navigation Components

The navigation system was designed for the robotic wheelchair presented in the Section 6.1.1. The environment considered is an indoor environment featuring static and dynamic objects. The navigation system is composed of the following components:

- A robotic platform component that talks directly to the hardware.
- Mapping functionality components that provides the information about the static elements of the environment.
- A localization component that estimates the pose of the robot with respect to the map reference frame.
- Components to detect and estimate the trajectory of moving objects to account for the dynamic entities of the environment.
- The ICS components: Ics-Check and Ics-Avoid to address the motion safety of the robotic system.

6.2.1 Organization of Components

The navigation components are organized in a ROS computation graph where the information is exchanged by publishing and subscribing to topics. Figure 6.2 summarize the topology of the graph. The topology shows how the nodes interact with each other. Conceptually the graph is quite simple. The node that corresponds to the robotic platform is providing all the information from the sensors. This information is processed in different stages. The first is the modeling of the environment. This modeling consists of the construction of a static map, the detection of moving objects and the estimation of its future motion. The second is the estimation of the current state of the robotic system. This requires locating the system within the environment and estimate its velocity with the odometry readings. Once the information of the environment and state of the system is available the decisional process can begin. The

decisional process is done through the ICS perspective taken in this work. ICS-AVOID will take the current state of the system and determine the appropriate control to take the robotic system to the next non-ICS state. In order to do that it will need to interact with ICS-CHECK. Once a safe control is found ICS-AVOID can send it to the robotic platform which will execute it. The process then can start all over again until the goal is reached.

A description of the inner functionality of each node in the graph is done in the following paragraphs.

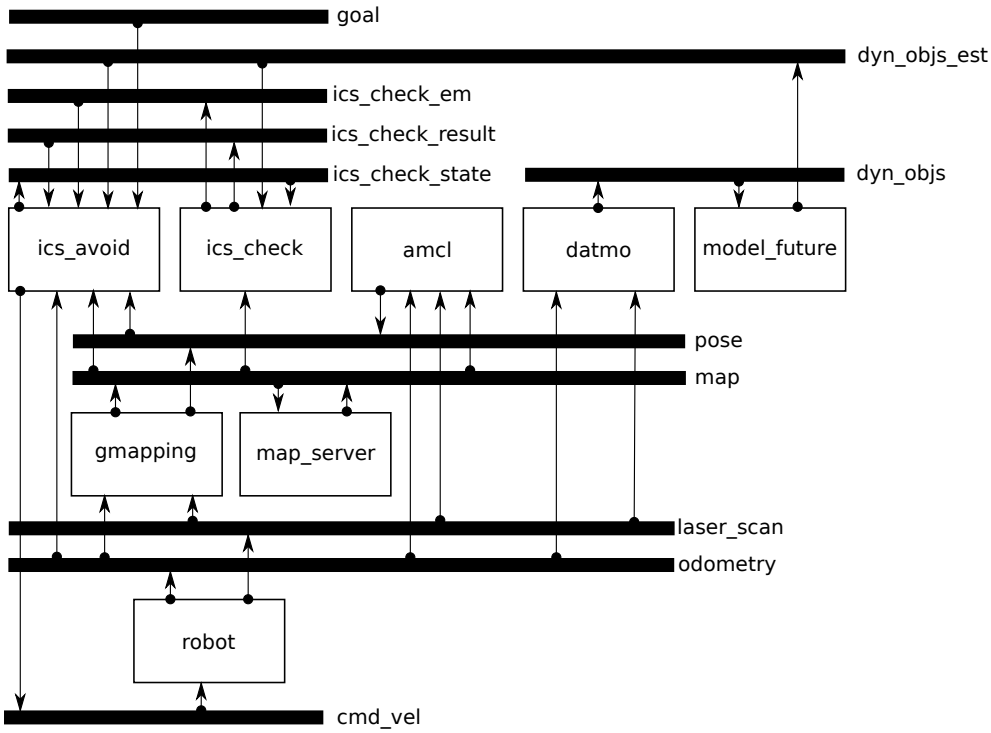


Figure 6.2: ROS Computation Graph Navigation System.

6.2.2 Robot System

The **robot** node extracts the sensor information (laser scan and odometry) from the robotic platform and receives the command (linear and angular velocity) to be sent to the platform. As the BlueBotics wheelchair was not originally supported by ROS it was necessary to implement the node from scratch. The data transmission with the wheelchair is through a TCP/IP socket over the Ethernet link. The communication protocol is a BlueBotics proprietary API called RPC-LOS (Remote Procedure Calls over

a Lightweight Object Streaming). This node is all its needed for the robotic platform to operate with ROS. The relevant topics are published (**laser_scan** and **odometry**) and a subscription is made to (**cmd_vel**) for receiving motion commands.

6.2.3 Mapping

The **gmapping** node provides the SLAM (Simultaneous Localization and Mapping) functionality. The algorithm is an implementation of a highly efficient Rao-Blackwellized particle filer to learn grid maps from laser scan data [GSB06]. The efficiency comes from the techniques employed to reduce the number of particles, the way of computing an accurate proposal distribution that considers not only the current movement of the robot but also the most recent observation and finally the approach taken to selectively perform the re-sampling operations. The node subscribes to the (**laser_scan** and **odometry**) topics and publishes in the (**map** and **pose**) topics which represents respectively a 2D occupancy grid of the environment and a configuration of the robot.

An utility node called **map_server** subscribe itself to the **map** topic to allow the generated maps to be saved to a file. In this way is possible to reuse maps of previously visited areas. The **map_server** offers the **map** data as a service to the interested nodes (in normal operation the map data is only needed once to construct a representation of the static elements of the environment).

6.2.4 Localization

Although the **gmapping** node provides an on-line estimation of the pose of the robot in the environment it was decided to use a more robust localization method. The localization is performed through the **amcl** node that implements the Adaptive Monte-Carlo Localization (AMCL) algorithm [KFM03]. This algorithm requires a pre-built static map of the environment against which to compare observed sensor values. The map is provided as a service (a request/reply transport type) by the **map_server** node. The AMCL algorithm works by maintaining a probability distribution over the set of all possible robot poses, and updates this distribution using data from odometry and the laser scans. The implementation of the algorithm represents the probability distribution using a particle filter. The filter is "adaptive" because it dynamically adjusts the number of particles in the filter: when the robot's pose is highly uncertain, the number of particles is

increased; when the robot's pose is well determined, the number of particles is decreased. The algorithm is therefore able to make a trade-off between processing speed and localization accuracy. The **amcl** node is subscribed then to the **map**, **laser_scan** and **odometry** service/topics and publishes in the **pose** topic.

6.2.5 Detection and Tracking of Moving objects

The detection and tracking of moving objects node (**datmo**) is responsible for providing the information about the dynamic entities around the robotic system. The DATMO algorithm[VA09] use a Bayesian-DDMCMC formulation. The Bayesian framework allows to include knowledge of various aspects including object, laser scan measurements and motion models. The Data-Driven Markov Chain Monte Carlo technique is used to sample the solution space (a joint multiple object-trajectory space) effectively to find the optimal solution. This formulation is general and can successfully handle the ambiguities in the presence of persistent occlusions. The **datmo** has subscription to the **laser_scan** and **odometry** and publishes a list of dynamic objects with information about their positions and their instantaneous linear velocities in the **dyn_objs** topic.

6.2.6 Model of the Future

Having access to the instantaneous velocities of the moving objects in the environment the **model_future** node builds an estimation of their future behaviour. For the time being the implementation of this node considers a very simple deterministic estimation. It is assumed that the objects in the environment will maintain its current linear velocity. This node is subscribed to the **dyn_objs** topic and publishes its estimation in the **dyn_objs_est**.

6.2.7 ICS-Check

The **ics_check** node computes the ICS-ness of a given state using the provided information of the environment as explained in Chapter 4. The state for the purposes of this experimental setup consists of the robot pose and the linear and angular velocities of the robotic wheelchair $s_c = (x, y, \theta, v, \omega)$. The world model is built using the information of the static objects present in the previously built map and the dynamic objects are taken as estimated by the model of the future. This node returns a boolean value indicating if the state belongs to the ICS and the set of evasive manoeuvres employed internally to perform the computations. In summary the node has a subscription to the

map, **dyn_objs_est** and **ics_state_check** topics and publishes its results in the **ics_check_result** and **ics_check_em** topics.

6.2.8 ICS-Avoid

The **ics_avoid** implements the algorithm presented in Chapter 5. It computes a control to take the robotic system from the current state to another non-ICS state. In this implementation a simple goal seeking behaviour is implemented using the sampling strategy "target heading, clearance and velocity" introduced in Section 5.1.5. To that end, it subscribe to the **goal** topic; the **map**, **dyn_objs_est** to obtain information about the environment; to the **pose** and **odometry** to assemble the variables for the state to be checked and; to the **ics_check_result** and **ics_check_em** to obtain the result from ICS-CHECK. It publishes in the **ics_state_check** topic to interact with ICS-CHECK and in **cmd_vel** topic to send a command to the robotic platform.

6.3 Evaluation

Having presented all the elements of the experimental platform is time now to proceed to the description of the experiment employed to validate the ICS-CHECK algorithm. The scenario serves to illustrate ICS-CHECK. The test orders the robotic system to follow a path that will take it to a collision. ICS-CHECK is applied to verify the ICS-ness of the current state.

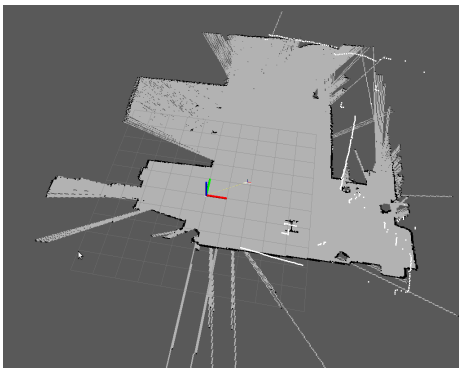
6.3.1 Experimental Conditions

The chosen environment to carry out the experimentations was the entrance hall of the INRIA Grenoble research center (Figure 6.3) host of the laboratory where this thesis was conducted. The dimensions of the hall are approximately 15 m by 12 m. The environment features several static obstacles consisting of the furniture and elements typically found in building receptions like this one. Being all this elements static, a map of the environment was made with the functionality provided by the **gmapping** node.

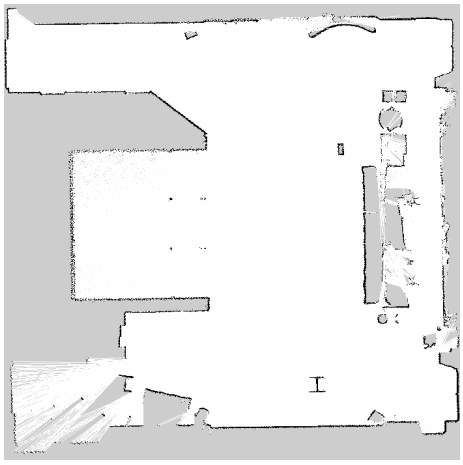
To make the map, the robotic wheelchair was driven around the entrance hall for approximately 5 minutes. At the end of the SLAM process the result was saved to a map file with the **map_server** node. The reason for this is that by saving the map it can be conveniently available for all the experiments that were performed later and at the same time improve the localization

Figure 6.3: INRIA Grenoble Entrance Hall.

of the robotic system inside the environment by using an specialized node. Figures 6.4a and 6.4b shows a snapshot of the SLAM process taking the information from the on-board sensors and the resulting map of the entrance hall respectively.



(a) SLAM process.



(b) Map of INRIA.

Figure 6.4: Mapping.

During the tests the localization was performed through the **amcl** node that through a particle filter maintains the probability distribution over the set of all possible robot poses. The particle with the highest probability is taken as the wheelchair pose in the environment. Figure 6.5 shows a graphical display of amcl particles associated with the different poses of the wheelchair. The reference frame of the map and of the robotic system are also shown in the figure.

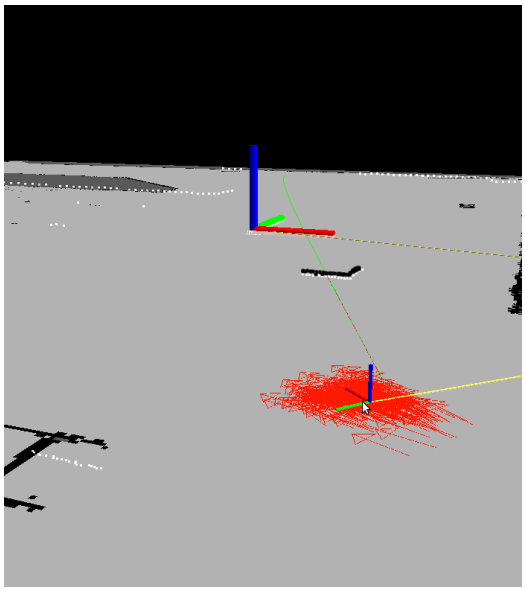


Figure 6.5: AMCL localization.

These were the conditions used in the performed experiment. The next section detail the test that was performed and the obtained results.

6.3.2 Collision With an Object

This scenario was conceived with the objective of demonstrating how ICS-CHECK can determine correctly when a state belongs to the ICS set. The obvious thing to do, is to put the robotic system at hand in a situation where a collision can not be avoided. To achieve this result it was decided that colliding on purpose with an static object was the better way to illustrate this. In order to do the experiment in the safest conditions the object to be collided was a plastic trash bin. The speed of the wheelchair was kept to a minimum to avoid any possible damage to the hardware elements of the wheelchair. The linear velocity of the robotic system at collision time was a moderate $0.18m/s$.

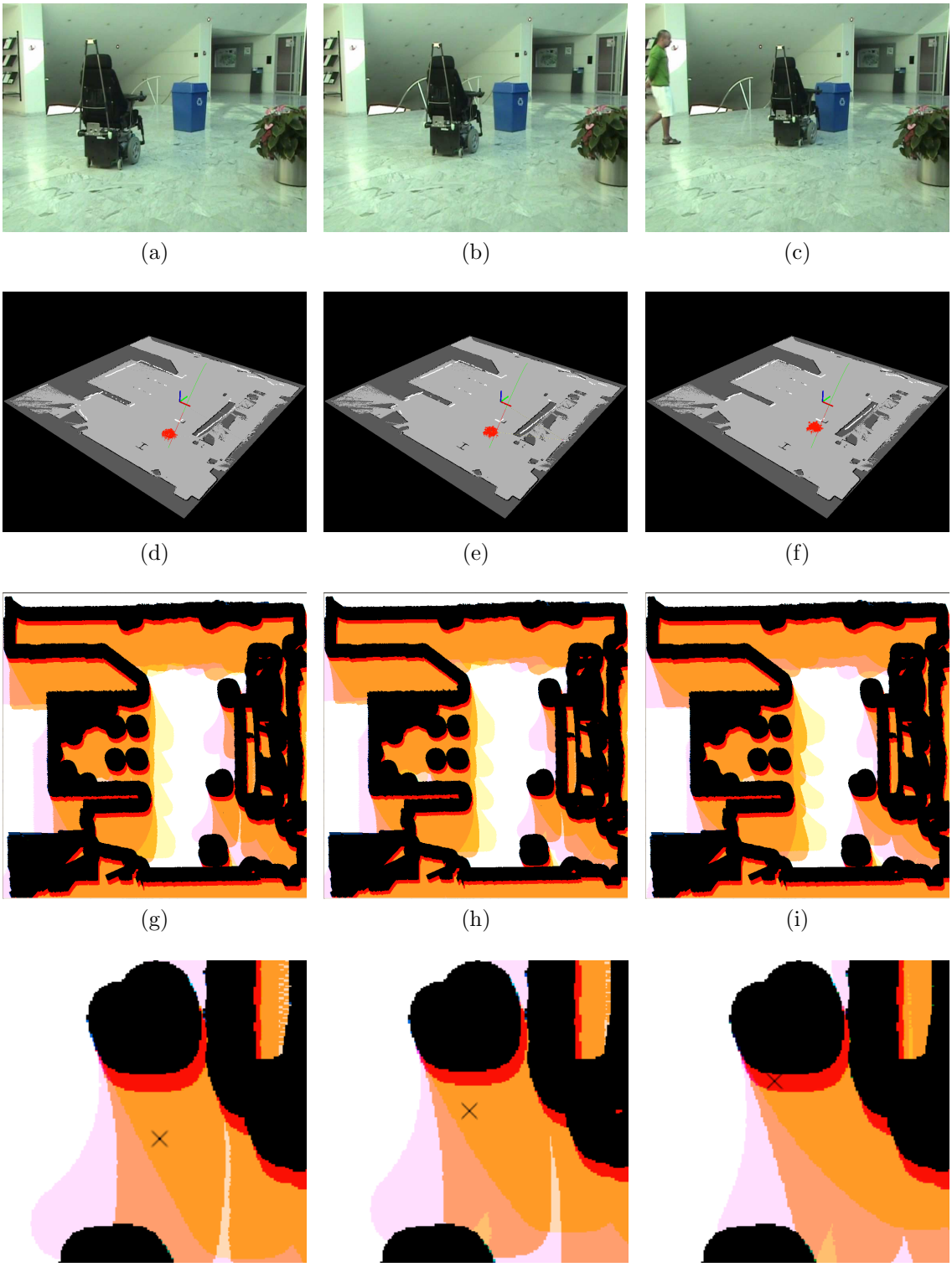


Figure 6.6: Robotic system in non-ICS states.

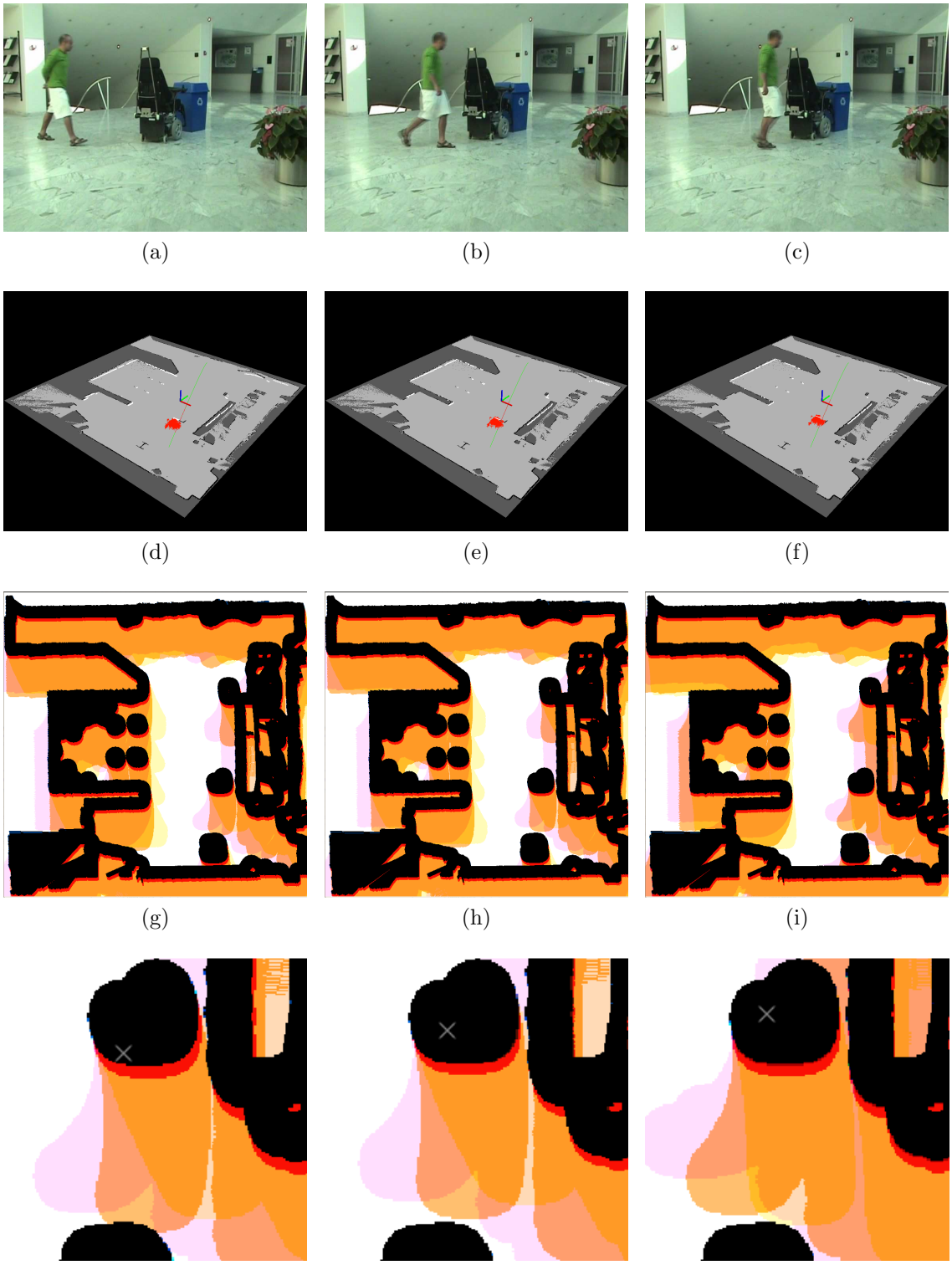


Figure 6.7: Robotic system in ICS states.

Figure 6.6 shows some snapshots during the trajectory followed by the wheelchair before going into an Inevitable Collision State. Each column in the figure is a different time in the trajectory. The first row in the figure shows the real-world scene that correspond to the time of the snapshot. The second row shows the information provided by the **amcl** node about the localization of the system in the environment and the input of the laser scan. The third row shows the 2D slice that correspond to the current state of the robotic system. This is the ICS-CHECK result window for the entire environment. The state of our robot is located near the lower right corner. A zoom in that region is shown in the fourth row of the figure. In this test there was no dynamic objects detected as the person that appears in the pictures of the test is located behind the field of view of the laser scan of the wheelchair. For this reason, the environment is taken as a static one and thus all the evasive manoeuvres considered are braking maneuvers. In total a set of nine braking manoeuvres were computed for this test. It can be seen that as the robotic wheelchair approaches the obstacle the number of maneuvers intersections that lead to collision increases. This is reflected visually by the color that corresponds to the state of the robotic system in the ICS 2D slice. The initial orange color moves towards the red. When the state of the system enters the black region it means it has reached an ICS. This situation is illustrated in Figure 6.7. The figure shows the moment when the wheelchair goes into an inevitable collision state. From that moment on the robotic system can not execute any action that will allow it to avoid a collision with the object. The collision takes place in the time corresponding to the third column of the figure.

6.4 Analysis

The running times of ICS-CHECK were logged during the execution of the manoeuvre. Figure 6.8 shows the boxplot of the recorded data. This graph helps to visualize the central value and variability of the running time. The median value is of 58 ms. However outliers with a maximum of 77 ms are registered. Although the time difference between the median and the outliers may not seem much, it is important to analyze this variability on the performance in order to determine whether or not the method can be used effectively given the time-decision constraint imposed by the dynamic environment. In this case 75% of the values are between 57 ms and 60 ms which signals a strong concentration around the median which in turns means stability in the computation time. The outliers then may come from

OS scheduling issues which may be solved with the use of a real-time OS that provides consistency concerning the amount of time it takes to complete the ICS-CHECK task.

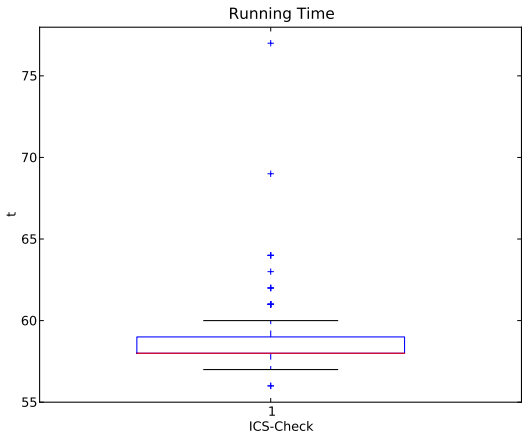


Figure 6.8: ICS-CHECK Running time.

Résumé

Ce chapitre montre les efforts réalisés pour l'obtention de résultats expérimentaux dans une plate-forme réelle. tout d'abord, la plate-forme est présentée: une chaise roulante. Après, l'architecture du logiciel est détaillée. L'architecture est présentée à travers de les modules qui permettent d'obtenir des informations sur l'environnement, de prendre une décision et finalement d'exécuter un ordre.

Les résultats obtenus, bien que limités, montrent la capacité de l'algorithme d'agir en temps réel. La contribution prinicipal de ce chapitre consiste dans las mis en ouvre des algorithmes conçue en chapitres antérieurs.

Chapter 7

Conclusions

This thesis proposes to use Inevitable Collision States (ICS) as an answer to the problem of safe navigation of an autonomous vehicle in a dynamic environment. An ICS 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 in the environment eventually occurs. ICS are particularly well suited to address the navigation problem since they take into account both the dynamic constraints of the robotic system and the future behaviour of the moving objects. These two elements are part of the key issues required in guaranteeing the motion safety of a robotic system that operates in dynamic environments.

The issues or safety criteria are associated with how time is handled by a robotic system when deciding its future course of action. Specifically, there is the need to reason about the future, doing it with an appropriate lookahead time and, respect a decision-time constraint imposed by the nature of the dynamic environments. Reasoning about the future allows to account for the constraints coming from the robotic system's dynamics and the future behaviour of the moving objects. If these constraints are not considered is likely that the decision taken will yield a collision in the future. Now, reasoning about the future is not enough, it must be done with an appropriate lookahead, that is, reasoning until the point in the future where no more information is needed to make a correct decision. Finally, the third safety criteria is that there is a real-time decision constraint that must be respected. A robotic system cannot safely remain passive in a dynamic environment as it risks to be collided by a moving object. The system has only a limited amount of time to come up with a decision that allows it to avoid a possible collision. Failure to do so will jeopardize its safety.

By reasoning with the ICS framework the first two safety criteria are taken into account. Both of them are explicity considered in the definition of ICS. In contrast, the third safety criteria, the time-decision constraint, is not considered in the ICS definition. This constraint has to be solved with an efficient way of working out the intrinsic complexity of the ICS characterization. Such algorithm allows to determine if a state is an ICS or not. Once a given state can be asserted or rejected as ICS is then possible to employ that information for navigation purposes. The way of doing that is through a collision avoidance strategy that keeps the robotic system at hand safe from falling in an ICS. If a robotic system doesn't wants to be involved in a collision it should never ever end up in an ICS. By navigating through non-ICS states the motion safety of the robotic system can be guaranteed with respect to the employed model of the future.

7.1 Contributions

The contributions of this work range from the theoretical study of the motion safety problem to the practical implementation of the algorithms designed to address it. The first contribution of the work is an evaluation of the navigation methods present in the literature with the help of the aformentioned safety criteria. This evaluation showed that some of the flaws of the navigation methods have their origin from not taking into consideration one or several of the safety criteria. Given this state of affairs, the concept of Inevitable Collision States was brought forward as a solution to the motion safety problem. However, the ICS definition and properties give only a theoretical answer to the safety problem. The second contribution was the further theoretical development of the ICS concept to obtain effiency principles that could be applied in the design of an algorithm that allowed the characterization of the ICS set for a given robotic system. The main theoretical results in that sense are the principle of reasoning on 2D slices of the state space of the robotic system (instead of attempting to perform computation in the fully-dimensioned state space) and the determination of a valid lookahead. A third contribution is the presentation of Ics-Check, a generic and efficient algorithm that determines whether a given state is an ICS or not with respect to the model of the future which is used. The algorithm is suited for the time-decision constraint imposed by dynamic environments obtaining its efficiency by applying the theoretical principles discussed earlier and by implementation principles that allows a suplementary increase in the performance, notably, the exploitation of graphics hardware performances that offer the possibility

of parallelizing some computations. The fourth contribution of this work was the introduction of ICS-AVOID, an algorithm that allows to find a control that takes the system between non-ICS states. By preventing the system to fall in an ICS is possible to guarantee its motion safety. The guarantee is derived from the ICS definition. When the robotic system's state is not an ICS it means that at least one collision-free trajectory exists and in consequence that is possible to follow it to avoid collision. Finally, the last contribution of this work is the implementation of the two algorithms within an experimental platform that allowed to validate the approaches in real experiments.

7.2 Discussion

From a global point of view the presented ICS perspective has allowed to address the motion safety problem in a structured way following three safety criteria principles. This structure allows to gradually move from the solid base of motion safety concepts to a practical implementation. However, the approach has some features that can be discussed and even contested. There is of course the assumption of the model of the future. Given the uncertain nature of the future this model will be at best an estimate of what will actually happen. That is why many peole could argue that there is no need to bother with such uncertain information when its validity can be easily contested. Or that constant updates in the information make any decision irrelevan in the long term. The point of view taken in this work is that not reasoning about the future leads more easily to take bad decisions that to do it even with the knowledge that the information may not be completely right. Using this idealized model of the future allows to produce formalisms and techniques that have relevance not only in the determinitic model but also in more flexible models. In addition, as the estimates of the future motion of dynamic objects become better when current prediction methods arrive to maturity, the relevance of approaches such as ICS that reasons about the future behavior of the objects will be greater.

7.3 Future Work

This work can not be complete without the outline of future lines of research that are envisaged.

As in any approach there are points that can be improved to increase the performance or the generality of the method. Among those who may

have a greater impact lies the selection of the set of evasive manouevres. The selected set of evasive manoeuvres determine the quality of the ICS approximation. In general having more manoeuvres improves the quality of the approximation. However more manoeuvres implies more processing time. Until now only a category of maneouvers that have shown good performance in the simulation and experiments done has been used. Braking manoeuvres and more generally imitating manouevres work fine for most situations. However, is possible that increasing the diversity of the manoeuvres can improve the approximation of the ICS set. In addition, such set of diversified evasive manoeuvres could also have a positive impact in ICS-AVOID. Careful analysis should be done to evaluate if adding a category of manoeuvres (*e.g.,* swerving manoeuvres) or increasing the number and diversity does actually improvement the quality of the solution.

An important feature of the Inevitable Collision State concept that open a future line of research is that it guarantee the safety of the robotic system at hand *with respect to the model of the future which is employed.* So far the characterization of the ICS has been based upon deterministic models of the future where the outcome is precisely known. In other words, each moving object was assumed to follow a given nominal trajectory (known *a priori* or predicted). Such deterministic models provide a clear-cut answer to the motion safety issue: a given state is an ICS or not (simple binary answer). However, such models are not well suited to capture the uncertainty that prevails in real world situations, in particular the uncertainty concerning the future behaviour of the moving objects. A more suitable model of the future is a probabilistic one. This type of model assigns a probability measure to the obstacle's future trajectories to express the degree of belief that they will occur. Uncertainty is explicitly represented and can be properly handled with probability theory. A probabilistic model can be built using methods proposed in the literature. For example methods where the model is obtained through an Extended Kalman Filter. Those methods forward simulate the obstacle trajectories using the prediction step of the filter which results in a distribution which is an estimate of the obstacle future position and uncertainty as a function of time. Other methods based in a Bayesian framework learn motion patterns from a set of observations which are later used to perform the prediction of future motion of the obstacles. A future line of research will have as objective precisely to address this type of models and to study to what extent the ICS concept can be extended to handle the uncertainty inherent to the future. To that end a *probabilistic formulation of the ICS concept* should be provided. A probabilistic ICS would permit the characterization of the motion safety likelihood of a given state, a likelihood

that can later be used to design safe navigation strategies in real world situations similar to what ICS-AVOID do now for the deterministic case. This change of paradigm would be the next step towards the applicability of the ICS framework to real robots operating in uncertain dynamics environments.

Résumé

La contribution de cet travail se situe a plusieurs niveaux. Tout d'abord, le concept de ICS est utilisé comme le référant pour analyser de manière rigoureuse les différentes contributions de les méthodes de navigation en environnement dynamique présent dans la littérature. ICS devient pretinent quand on prend en compte la dynamique de l'environnement, les obstacles fixes et mobiles, l'evolution futur de ces derniers, ainsi que la dynamique du robot lui même. Une deuxième contribution a été d'approfondir le concept d'ICS dans le but de concevoir des algorithmes permettant de manière effective de caractériser les états qui appartient a l'ensemble d'ICS pour un robot mobile en environnement dynamique. Enfin, le système a été effectivement mis en oeuvre dans une plate-forme réelle:

Bibliography

[BATM93] P. Bessiere, J.-M. Ahuactzin, E.-G. Talbi, and E. Mazer. The ariadne's clew algorithm: global planning with local methods. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, pages 1373–1380, 1993.

[BCF+00] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 2000.

[BGF08] J.L. Blanco, J. González, and J.A. Fernández. Extending obstacle avoidance methods through multiple parameter-space transformations. *Autonomous Robots*, 24(1):29–48, 2008.

[BH95] M. Barbehenn and S. Hutchinson. Efficient search and hierarchical motion planning by dinamically maintaining single-source shortest path treees. *IEEE Transactions on Robotics and Automation*, 11(2), 1995.

[BJ91] J. Barraquand and Latombe J.C. Robot motion planning: A distributed representation approach. *Int. Journal of Robotics Research*, 10(6), 1991.

[BK89] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, sep/oct 1989.

[BK91] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 278–288, 1991.

[BK99] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 341–346, 1999.

[BK01] O. Brock and L.E. Kavraki. Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1469–1474, 2001.

[BV03] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *RoboCup 2002: Robot Soccer World Cup VI*, pages 288–295. Springer Berlin / Heidelberg, 2003.

[Can88] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[CB00] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *Int. Journal of Robotics Research*, 19(2), 2000.

[CD88] J.F. Canny and B. Donald. Simplified voronoi diagrams. *Discrete and Computational Geometry*, 3(1), 1988.

[Cha87] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[CLH+05] H. Choset, K.M Lynch, S. Hutchinson, G.A. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, 2005.

[DF08] V. Delsart and T. Fraichard. Navigating dynamic environments using trajectory deformation. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, 2008.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 1959.

[Dij68] E. W. Dijkstra. Hart, p.e. and nilsson, n.j. and raphael, b. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 1968.

[Elf89] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(5):46–57, 1989.

[ELP87] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2(4):477–521, 1987.

[FA04] T. Fraichard and H. Asama. Inevitable collision states. a step towards safer robots? *Advanced Robotics*, 18(10), 2004.

[FBT97] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, 1997.

[Fra98] T Fraichard. Trajectory planning in a dynamic workspace: a 'state-time space' approach. *Advanced Robotics*, 13(1):75–94, 1998.

[Fra07] T. Fraichard. A short paper about motion safety. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1140–1145, 2007.

[FS98] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *Int. Journal of Robotics Research*, 17(7), 1998.

[FS06] D. Ferguson and A. Stentz. Anytime rrts. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, pages 5369–5375, 2006.

[GC02] S.S. Ge1 and Y.J. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, 2002.

[GCK+09] S.J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proceedings ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187, 2009.

[GSB06] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2006.

[HAK+08] R. Hoeger, A. Amditis, M. Kunert, A. Hoess, F. Flemish, H.-P. Krueger, A. Bartels, A. Beutner, and K. Pagle. Highly automated vehicles for intelligent transport: Haveit approach. In *Proceedings of the 15th World Congress on ITS*, 2008.

[HBK+05] F. Holzmann, M. BeHino, S. Kolskit, A. Sulzmann, G. Spiegelberg, and R. Siegwart. Robots go automotive - the sparc approach. In *Proceedings IEEE Intelligent Vehicles Symposium*, 2005.

[HKLR02] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Int. Journal of Robotics Research*, 21(3), 2002.

[Hua08] L Huang. Velocity planning for a mobile robot to track a moving target — a potential field approach. *Robotics and Autonomous Systems*, 57(1):55–63, 2008.

[KF07] H. Kurniawati and Th. Fraichard. From path to trajectory deformation. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, 2007.

[KFM03] C. Kwok, D. Fox, and M. Meila. Adaptive real-time particle filters for robot localization. In *Proceedings IEEE International Conference on Robotics and Automation*, 2003.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1), 1986.

[KJCL97] M. Khatib, H. Jaouni, R. Chatila, and J.-P. Laumond. Dynamic path modification for car-like nonholonomic mobile robots. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2920–2925, 1997.

[KK92] J.-O. Kim and P.K. Khosla. Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation*, 8(3):338–349, 1992.

[KL02] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings IEEE International Conference on Robotics and Automation*, 2002.

[KSLO96] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional

configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[Lat91] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[LaV98] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, October 1998.

[LaV06] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[LBL04] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 20(6), 2004.

[LFG+05] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2005.

[LK01] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[LLS05] F. Large, C. Laugier, and Z. Shiller. Navigation among moving obstacles using the NLVO: Principles and applications to intelligent vehicles. *Autonomous Robots*, 19(2):159–171, 2005.

[LP83] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computing*, C-32(2):108–120, 1983.

[LPW79] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[MM06] J. Minguez and L. Montano. Abstracting vehicle shape and kinematic constraints from obstacle avoidance methods. *Autonomous Robots*, 20(1):43–59, 2006.

[MS08] Pascal Morin and Claude Samson. *Handbook of Robotics*, chapter Motion Control of Wheeled Mobile Robots, pages 799–826. Springer Berlin Heidelberg, 2008.

[NS98] Y.K. Nak and R. Simmons. The lane-vurvature method for local obstacle avoidance. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, pages 1615–1621, 1998.

[OM05] E. Owen and L. Montano. Motion planning in dynamic environments using the velocity space. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, pages 2833–2838, 2005.

[PF05] S. Petti and Th. Fraichard. Partial motion planning framework for reactive planning within dynamic environments. In *Proceedings IEEE International Conference on Robotics and Automation*, 2005.

[PS03] R. Philippsen and R. Siegwart. Smooth and efficient obstacle avoidance for a tour-guide robot. In *Proceedings IEEE International Conference on Robotics and Automation*, 2003.

[PSF01] E. Prassler, J. Scholz, and P. Fiorini. A robotic wheelchair for crowded public environments. *IEEE Robotics Automation Magazine*, 8(1):38–45, 2001.

[QGC+09] M. Quigley, B. Gerkey, K. Conley, J. Fausty, T. Footey, J. Leibsz, E. Bergery, R. Wheelery, and A. Ng. Ros: an open-source robot operating system. In *Proceedings IEEE International Conference on Robotics and Automation*, 2009.

[QK93] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807, 1993.

[RAH+09] Hoeger Reiner, Amditis Angelos, Zeng Holger, Hoess Alfred, Flemish Frank, Bartels Arne, and Jakobsson Erika. Selective automated driving as a pivotal element to solve safety and environmental issues in personal mobility. In *Proceedings of the 16th World Congress on ITS*, 2009.

[Rei79] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[RK92] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, 1992.

[Sim96] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 3375–3382, 1996.

[SP07] M. Seder and I. Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1986 –1991, 2007.

[SS83a] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[SS83b] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: II. General techniques for computing topological properties of algebraic manifolds. *Advances in Applied Mathematics*, 12:298–351, 1983.

[Ste93] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation*, 10:89–100, 1993.

[Ste95] A. Stentz. The focussed d* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[TBB+99] Sebastian Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, Frank Dellaert, Dieter Fox, D. Haehnel, Chuck Rosenberg, Nicholas Roy, Jamieson Schulte, and D. Schulz. Minerva: A second generation mobile tour-guide robot. In *Proceedings IEEE International Conference on Robotics and Automation*, 1999.

[UB98] I. Ulrich and J. Borenstein. Vfh+: reliable obstacle avoidance for fast mobile robots. In *Proceedings IEEE International*

Conference on Robotics and Automation, pages 1572–1577, 1998.

[UB00] I. Ulrich and J. Borenstein. Vfh*: local obstacle avoidance with look-ahead verification. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2505–2511, 2000.

[VA09] Trung-Dung Vu and O. Aycard. Laser-based detection and tracking moving objects using data-driven markov chain monte carlo. In *Proceedings IEEE International Conference on Robotics and Automation*, 2009.

[vdBO05] J.P. van den Berg and M.H. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[VKA05] N. Vandapel, J. Kuffner, and O. Amidi. Planning 3-d path networks in unstructured environments. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 4624–4629, 2005.

[WvdBM09] D. Wilkie, J. van den Berg, and D. Manocha. Generalized velocity obstacles. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, pages 5573–5578, 2009.