HAL
open science

Characterization of the Inevitable Collision States for a Car-Like vehicle

Rishikesh Parthasarathi

To cite this version:

Rishikesh Parthasarathi. Characterization of the Inevitable Collision States for a Car-Like vehicle. [University works] 2006. inria-00182005

HAL Id: inria-00182005

https://inria.hal.science/inria-00182005v1

Submitted on 24 Oct 2007

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.

Master 2 Recherche

Imagerie, Vision et Robotique

Characterization of the Inevitable Collision States for a Car-Like vehicle

---

Submitted By : Parthasarathi Rishikesh

Supervisor : Thierry Fraichard

Team: E-Motion

GRAVIR Laboratory

l'INRIA Rhône-Alpes

Abstract

Navigation is an important aspect of mobile robotics. Recently, we see a trend in robotic systems leaving laboratories and clean rooms and moving to real world environments. At this point, it is critical to ensure the safety of the robotic system and the environment in which it moves.

In this work, the topic of "safe navigation" of a robotic system in a highly dynamic environment is considered. It is suggested that the dynamics of the robotic system and of the moving obstacles must be taken into account while performing navigation. It is illustrated that the idea of using a time-horizon for calculating a navigation plan does not guarantee safety. Developing on these points as the motivation, the novel concept of Inevitable Collision States ($\mathcal{ICS}$) is introduced. An inevitable collision state is a state from which any action taken by the robotic system will lead to a collision. A theoretical extension, the notion of *imitating manoeuvres* is proposed and proved to be efficient in calculating the $\mathcal{ICS}$ for a robotic system.

A case study involving the characterization of the $\mathcal{ICS}$ for a car-like vehicle is performed. A polynomial complexity algorithm complexity is proposed for an efficient computation of the $\mathcal{ICS}$. The algorithm is implemented in C++. Experimental results obtained in various environments are presented.

Contents

Acknowledgements **1**

1 Introduction and Overview **2**
1.1 Motivations . . . 2
1.1.1 Motivation 1: Considering Dynamics . . . 4
1.1.2 Motivation 2: Avoiding Time-Horizons . . . 5
1.2 Objectives . . . 5
1.3 Contributions of the report . . . 6
1.4 Report Outline . . . 6

2 Literature Review **8**
2.1 Basic Concepts . . . 8
2.1.1 Configuration Space Formulation . . . 8
2.1.2 State Space Formulation . . . 9
2.1.3 Configuration-Time Space Formulation . . . 10
2.1.4 State-Time Space Formulation . . . 11
2.2 Deliberative Aprroaches . . . 12
2.3 Reactive Approaches . . . 12
2.3.1 Potential Field Approaches . . . 13
2.3.2 Vector Field Histograms . . . 14
2.3.3 Dynamic Window Approach . . . 14
2.3.4 Linear Velocity Obstacles . . . 15
2.4 *k-step* Approaches . . . 16
2.5 Conclusion . . . 17

3 Inevitable Collision States and Obstacles **19**
3.1 Notations and Preliminary Definitions . . . 20
3.2 Inevitable Collision States and Obstacles . . . 21
3.3 Calculating $ICO(\mathcal{B})$ . . . 23

4 Case Study: Car-Like Vehicle **24**
4.1 Characterization of the $\mathcal{ICS}$ . . . 24
4.1.1 Statement of the Problem . . . 24
4.1.2 Trajectory Parameterization . . . 25
4.1.3 ICO calculation for a point obstacle moving in a straight line . . . 26
4.1.4 ICO calculation for a solid obstacle moving in a straight line . . . 28
4.1.5 ICO calculation, for a stationary point and solid obstacle . . . 28

4.2 On the Conservative Approximation . . . 29
4.3 Imitating Manoeuvres . . . 30
4.3.1 Catching-up manoeuvre . . . 32
4.4 Catching-up manoeuvres for the car-like vehicle . . . 32
4.4.1 *ICO* calculation for an obstacle moving in a straight line . . . 32
4.4.2 Optimal Control . . . 32
4.4.3 Optimal Catching-up Manoeuvre . . . 33

5 Algorithm and Software Implementation 36
5.1 Algorithm for $\mathcal{ICS}$-*Checker* . . . 36
5.1.1 Input . . . 36
5.1.2 Output . . . 36
5.1.3 Algorithm . . . 36
5.1.4 Complexity of the Algorithm . . . 37
5.2 Software . . . 38
5.2.1 Programming . . . 38
5.2.2 Experimental Results . . . 39

6 Conclusion and Future work 41

References 43

Acknowledgements

I would like to take this opportunity to thank a number of people who have played an important role during the past one year. Firstly, I would like to thank the French Embassy of Singapore for granting me a scholarship to pursue my Masters degree in France and Dr. Christian Laugier for giving me an opportunity to work in the E-Motion team.

I would like to thank my supervisor Dr. Thierry Fraichard for all the discussions, suggestions and guidance that he has provided me during this internship. I am sure that I have learnt a lot about research from him. I would like to thank my colleague Ronan whose every suggestion in programming improved my software. I also extend my thanks to Christopher and Pierre for discussions concerning various topics, including work.

Finally, I would like to thank Chen Cheng and Ronan for helping me proof-read the report.

Chapter 1

Introduction and Overview

1.1 Motivations

Autonomous robotic systems are a major undertaking in robotics. In the recent years, robotic systems have started leaving laboratories, clean rooms, restricted environments and have started moving towards real world environments. Examples include indoor and outdoor robots performing various tasks from sweeping floors (Probotics Cye-R, Gecko Carebot, iRobot Roomba, *etc.*), tour-guiding people in museums (Rhino, Minerva, Robox, *etc.*), helping people with disabilities, transporting humans in the form of intelligent vehicles in urban areas (Cycab, Parkshuttle), and traversing unstructured terrains (The Grand DARPA Challenge) *etc.*

Autonomous system designing requires tackling a number of challenging problems in perception, localization, modeling, reasoning under uncertainty, decision making, *etc.* Whatever the problem considered, the final decision is made on choosing a control action to produce motion. Hence, motion is of prime importance in autonomous sytems. The more sophisticated the applications become, the more concerned we ought to be about the safety of these robotic systems and the environment in which these interact, be it with people, vehicles or other autonomous systems. The characteristic of a real world environment, irrespective of whether it is structured or unstructured, hostile or friendly, is that it is *dynamic*. That is to say that it contains objects that move. The desired robotic system is expected to move in an environment that may contain moving obstacles, static obstacles in the from of buildings *etc.*

If we would like to create robotic systems that can reach a destination with no or least human intervention (by least human intervention we mean that the input descriptions will only be what we want the vehicle to do rather than how to do it), we must be able to achieve navigation (by navigation, we basically mean the problem of determining the elementary motion that the robotic system should perform during the next time-step) in a dynamic environment. As soon as the size and dynamics of the robotic system make it potentially harmful to itself or it's environment, the system should strive to avoid collision. Now, as advancement in tourist guide vehicles and intelligent transportation vehicles continues, we must ask ourselves one important question. "How safe are these vehicles in a dynamic environment with people, other moving obstacles ?" It is important to understand this operational safety issue.

This is the focus of collision avoidance in motion planning and navigation. In fact

navigation approaches for mobile robotic systems can be broadly classified into three categories, *reactive*, *k-step* and *deliberative* approaches. The deliberative approaches calculate a complete trajectory from the initial point to goal point. These can be computed offline and there is no need to perform any additional obstacle avoidance at execution time. A major disadvantage of these approaches is that they are too slow because of the calculations involved in determining the obstacle-free space. Examples of deliberative approached include road-map approaches, cell decomposition, approximate cell decomposition [1], potential field approaches [2]*etc.*

A robotic system in a dynamic environment faces a hard real-time constraint and is obliged to make a decision within a bounded time. This constraint rules out the usage of deliberative approaches in navigation. Thus our study of real-time navigation in dynamic environments will mostly focus on the other two categories.

Reactive approaches, in contrast with deliberative approaches, generate only control actions that are to be performed at the next time instant. The key advantage of the reactive methods is their low computational load which is particularly more important in a dynamically changing environment. Reactive approaches perform the calculations at execution time unlike the deliberative approach where obstacle avoidance is done offline. This comes at an obvious disadvantage that they cannot yield optimal solutions. Another not so obvious disadvantage is the tendency to be stuck at local minima like in the cases of Vector Field Histograms (VFH) [3] and its derivative VFH+ [4], potential field methods, Dynamic Window Approach [5] *etc.*

The primary concern of navigation is to ensure safety of the robotic system. In an environment with moving obstacles this concern is critical as there is a necessity to take into concern the dynamics and the future behavior of the obstacles and the robotic system. A number of research papers [6], [7], [8] have addressed this issue. Another topic of interest under the framework of navigation is to incorporate the idea of a *time horizon* as in [7]. Before discussing the problems of incorporating a time-horizon, it is necessary to explain how it is implemented or rather what characterizes a time-horizon. Let us say we a model of the environment from the current time instant ($t = 0$) till $t = \infty$. If we use a time-horizon of say, $t = 100$, the model of the environment at $t = 101$ and then on is just truncated. It is assumed that the model is not available. The reason for incorporating a time-horizon normally is mainly to reduce computational load. While [7], uses a time-horizon, [8] and [6] assume that a complete trajectory to the goal can be computed when considering the dynamics of the robotic system and the moving obstacles.

The *k-step* approach is in between the deliberative and reactive approaches. These approaches cater to the hard real time constraint and calculate a partial plan of k-steps. They also take into consideration the environment changes into their planning. Examples include the Partial Motion Planning (PMP) scheme [9] and the VFH* [10] variation of the VFH method. PMP has an any-time flavor *i.e.*, it returns the best trajectory when requested. In general, the look-ahead of *k-step* methods is shorter than deliberative approaches but better than reactive approaches and hence the chance of getting trapped in a local mininma is lower. They also claim that a complete trajectory to the goal cannot be calculated under hard real-time constraints. One important issue concerning the *k-step* methods is that it is necessary to guarantee the safety of the system at the end of the k steps. By safety we mean that the system should not end in a state where it cannot avoid a collision.

Before going into the safety issue in k-step methods, let us show that there is in fact a need to consider the dynamics of the system and of the moving obstacles. We will also discuss why it is correct to avoid using time-horizons in order to ensure safety of the robotic system. It takes a couple of simple examples such as the ones depicted in Fig. 1.1 and Fig. 1.2 to illustrate this.

1.1.1 Motivation 1: Considering Dynamics

To explain the necessity of considering the dynamics of the system and of the moving obstacles, let us asumme the following case. $\mathcal{A}$ is considered to be a point mass capable of translating only to its right and is also capable of accelerating and decelerating. $\mathcal{A}$ is also assumed to be forward-moving only, as opposed to being able to reverse the direction of motion. The obstacle $\mathcal{B}$ is moving with a constant velocity $v_{\mathcal{B}}$ in the opposite direction.

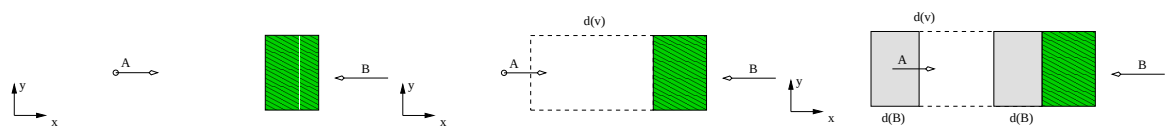

Figure 1.1: $\mathcal{A}$ translates along $\hat{x}$ without any rotation. The obstacle $\mathcal{B}$ (striped region)translates with constant velocity -v_B along the $\hat{x}$ axis. The plane (x,y) denotes $\mathcal{W}$ where the objects move. a) The case where both $\mathcal{A}$'s dynamics and $\mathcal{B}$'s dynamics are not considered. b) $\mathcal{A}$'s dynamics is alone considered. c) The case where the dynamics of both $\mathcal{A}$ and $\mathcal{B}$ are considered.

Let us now discuss the different cases where dynamics of the objects in the workspace are either considered or neglected.

Not considering any system dynamics:

If the system dynamics of $\mathcal{A}$ and $\mathcal{B}$ are not considered, then the region occupied by the obstacle is the only forbidden area. In Fig.1.1a, this is represented by the striped area. In this case, the precaution is to not be inside this striped area.

Considering only the dynamics of $\mathcal{A}$

If the dynamics of $\mathcal{A}$ alone is considered, we will take into account the distance $d(v)$ in Fig, 1.1b,that it takes $\mathcal{A}$ to break according to the current velocity. This produces the extended region of $\mathcal{B}$ along the direction of $\mathcal{A}$'s velocity. It is obvious that the dynamics or the future motion of the obstacle are not taken into consideration and the precaution is not to be inside this new area from where $\mathcal{A}$ cannot stop before hitting $\mathcal{B}$.

Considering the dynamics of both $\mathcal{A}$ and $\mathcal{B}$

In this case, both $\mathcal{A}$'s and $\mathcal{B}$'s dynamics are taken into consideration. We will have the distance $d(v)$ that it takes $\mathcal{A}$ to break according to the current velocity and also the distance $\mathcal{B}$ will travel $d(\mathcal{B})$. Hence the collision region is a distance $d(v) + d(\mathcal{B})$

(Fig, 1.1c,)from the current position of $\mathcal{B}$. This is bad news since $\mathcal{A}$ is already in this region. Hence a collision with $\mathcal{B}$ is inevitable.

Hence, if a naive collision avoidance system that doesn't take into consideration both the dynamics of $\mathcal{A}$ and of $\mathcal{B}$ is used, it would assume that the current state is safe and would eventually lead to a collision. This simple but neat example emphazises the lack of safety in most of the current algorithms in literature like [5],[10],[2],[3] *etc.*

1.1.2 Motivation 2: Avoiding Time-Horizons

When it comes to the issue of time-horizon, it is argued that whenever a robotic system decides it future motion by restricting the reasoning to a finite time-horizon, collisions may potentially happen beyond this time-horizon and safety is not guaranteed. This argument is supported with the following example. Let us consider the example in Fig.1.2

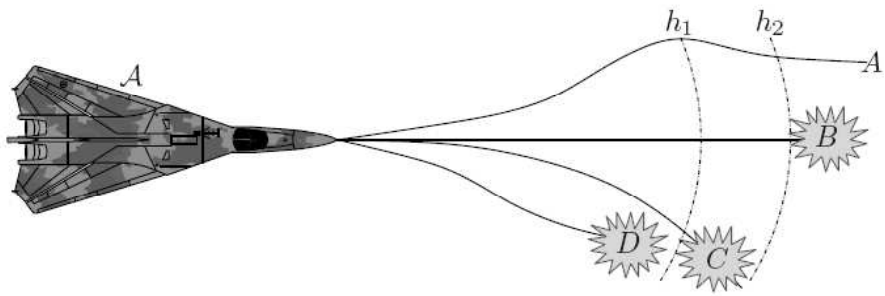

Figure 1.2: $\mathcal{A}$ is in a situation to choose one among the four trajectories. In reality, trajectory A is the only safe trajectory, but incorporation of a time-horizon at h_2 makes B a safe trajectory and at h_1 makes C a safe trajectory, which are obviously fatal. Example taken from [11].

Now in our example, it is assumed that $\mathcal{A}$ can choose between four trajectories only to reach it's goal, A, B, C or D. Evaluating each of these trajectories with a time-horizon at h_1, leads $\mathcal{A}$ to eliminate trajectory D and choose between A, B and C. If B or C is unfortunately chosen, $\mathcal{A}$ is definitely in trouble. It should be noted that increase of the time-horizon will not solve the problem. Only a time-horizon infinite or at least greater than the time to reach the goal state guarantees $\mathcal{A}$'s safety (assuming that $\mathcal{A}$ can rest safely at the goal).

1.2 Objectives

The main objective of this project is to consider the dynamics of the robotic system, the dynamics of the moving objects and to avoid the incorporation of a time-horizon in navigation. The navigation method that will be used is the *k-step* navigation. Although technically sound and computationally fast, the key question that arises whenever we use *k-step* methods is *How sure can we be that the system will never end up in a state (at the end of k-steps) from where a collision is inevitable?.*

The solution proposed in this document is the concept of Inevitable Collision States ($\mathcal{ICS}$) introduced in [12]. An $\mathcal{ICS}$ for a robotic system is a state for which, no matter what the future trajectory is, a collision eventually occurs. So the safety of the system is guaranteed by never being in an $\mathcal{ICS}$. A theoretical solution for the safety of a robotic system is offered in Chapter 3.

1.3 Contributions of the report

The basic study of this concept was done in [13]. Formal definitions of the inevitable collision states and obstacles, properties fundamental for their characterisation and case study with simple static objects, *etc* were introduced in [12]. This report extends this concept to moving objects.

On the theoretical side, the concept of *imitating manoeuvres* that produce a compact approximation of the $\mathcal{ICS}$ is proposed. These manoeuvres are suitable for systems that are capable of reproducing the trajectory of the obstacles. They are comprised of a *catching-up* part, in which the system orients itself along the direction of the obstacle, and an *imitating* part, in which the system follows the control inputs of the obstacle. It is proved that the $\mathcal{ICS}$ region depends on the time taken for the catching-up part. An optimization procedure that uses bang-controls is performed to take the robotic system through the catching-up part as fast as possible, thus reducing the $\mathcal{ICS}$ region as much as possible. This concept is further explored to encompass braking trajectories, a popular concept in collision avoidance literature, as a special case of imitating manoeuvres.

On the practical side, the $\mathcal{ICS}$ concept is applied to a car-like vehicle case study. An algorithm for an $\mathcal{ICS}$**-Checker** is proposed. In short, the algorithm is a predicate `ICS`($\mathbf{s}$) that returns true if and only if the state $\mathbf{s}$ is an $\mathcal{ICS}$. The algorithm has a polynomial complexity. The algorithm is implemented in software with the braking trajectories and imitating manoeuvres. Typical environments, both cluttered and highly dynamic, are defined. Examples include a highway scenario and a city scenario with pedestrians and buildings etc. A maze environment for a cycle, which could be considered as the piano-mover problem's (in configuration space) equivalent in state space, is also presented. The idea is to calculate if a given trajectory is collision free or whether there exists a safe trajectory.

1.4 Report Outline

This document is organized as follows.

- **Chapter 2** introduces the basic concepts and formalization for solving navigation problems. A literature review details the current state of the art. Although various algorithms exist, only algorithms that are relevant to the one proposed in this document are reviewed. This chapter concludes with a statement of problem for which a solution is proposed in this document.

- **Chapter 3** introduces the concept of Inevitable Collision States ($\mathcal{ICS}$) and formal definitions of the inevitable collision states and obstacles are presented. Properties fundamental for their characterisation are established.

- **Chapter 4** describes the characterization the $\mathcal{ICS}$ for a car-like vehicle. Extensions to the concepts in Chapter 3 (concept of imitating manoeuvres) are proposed. The proposed concept is extensively detailed and it's capability to improve the quality of conservativeness in approximating the $\mathcal{ICS}$ is theoretically proved.

- **Chapter 5** discusses the overall algorithm for generating the $\mathcal{ICS}$ region and also for collision-checking in the form of predicate `ICS`(**s**). A software implementation of the proposed algorithm is then described. Typical environments and classical problems are introduced and implemented using the software. A laborious analysis of the complexity of the algorithm is performed.

- **Chapter 6** concludes the document, and proposes ideas for further work that can be performed.

Chapter 2

Literature Review

The robotics literature presents many approaches to collision-free motion for robots. They are generally categorized according to their underlying methodology. Historically, navigation was generally an off-line task, as only stationary environments were considered. Methods used were also purely geometry-based. The problem was formalized in high dimensional spaces rather than the workspace $\mathcal{W}$. The advantage is that the robotic system can be replaced by a point in higher dimensions and hence navigation for the system is reduced to a problem involving a point object. An extensive review is provided in [1].

Research in this area is mature and the notion of configuration space ($\mathcal{C}$) proposed in [14] has been the most successful way of solving these problems. Later extensions to this formulation resulted in the notion of State space ($\mathcal{S}$), Configuration-Time space ($\mathcal{CT}$), State-Time space ($\mathcal{TS}$) *etc*, according to whether the dynamics of the robotic system and moving obstacles are taken into consideration.

2.1 Basic Concepts

2.1.1 Configuration Space Formulation

The underlying idea of the $\mathcal{C}$-space is to represent $\mathcal{A}$ as a point in an appropriate space-$\mathcal{A}$'s $\mathcal{C}$-space, and to map the obstacles in this space. This mapping transforms the problem from plannin a motion for a dimensioned object, in this case $\mathcal{A}$, to planning a motion for a point object in the new space. In other words, it make the constraints on the motions of $\mathcal{A}$ more explicit.

Notion of $\mathcal{C}$-space

Let $\mathcal{A}$ be represented as a closed polygonal region, a subset of $\mathcal{W}$, the workspace, which in turn is the physical space on which $\mathcal{A}$ moves. $\mathcal{W}$ is either R^2 or R^3 as the case may be. The obstacles $\mathcal{B}_i$'s are considered fixed closed polygonal regions of $\mathcal{W}.\mathcal{F}_\mathcal{A}$ is the moving frame with $\mathcal{A}$ and $\mathcal{F}_\mathcal{W}$ is fixed. By definition, since $\mathcal{A}$ is rigid, every point a in $\mathcal{A}$ has a fixed position in $\mathcal{F}_\mathcal{A}$. But a's position in $\mathcal{W}$ depends on the position of $\mathcal{F}_\mathcal{A}$ in $\mathcal{F}_\mathcal{W}$. The $\mathcal{B}_i$'s are fixed and hence every point on each $\mathcal{B}_i$ has a fixed position w.r.t $\mathcal{F}_\mathcal{W}$.

A configuration $\mathbf{q}$ of $\mathcal{A}$ is a set of parameters that uniquely determine the position of every point on $\mathcal{A}$. In other words, $\mathbf{q}$ specifies the position and orientation of $\mathcal{F}_{\mathcal{A}}$ in $\mathcal{F}_{\mathcal{W}}$. The set of all configurations $\mathbf{q}$'s forms the configuration space $\mathcal{C}$, and a configuration is represented as a point in $\mathcal{C}$. A configuration $\mathbf{q}$ is collision-free if $\mathcal{A}$ placed at $\mathbf{q}$ does not collide with obstacles in the workspace or with itself. The free configurations form a subset F of $\mathcal{C}$.

The configuration space of a robotic system is the appropriate framework to address path planning problems where the focus is on the geometric aspects of motion planning (no collision between the system and the fixed obstacles of the workspace).

Obstacles in $\mathcal{C}$-space

The $\mathcal{B}_i$'s in the previous section were the workspace image of the obstacles. To plan a path in the configuration space, the obstacles have to be mapped to the $\mathcal{C}$-space. Every obstacle $\mathcal{B}_i$ in $\mathcal{W}$ maps in $\mathcal{C}$ to a region:

$$\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C}/\mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \emptyset\} \tag{2.1}$$

This region $\mathcal{CB}_i$ is called the $\mathcal{C}$-obstacle of $\mathcal{B}_i$. The union of all the $\mathcal{C}$-obstacles:

$$\bigcup_{i=1}^{\mathbf{q}} \mathcal{CB}_i \tag{2.2}$$

is called the $\mathcal{C}$-obstacle region and the set

$$\mathcal{C}_{free} = \mathcal{C}/\bigcup_{i=1}^{\mathbf{q}} \mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C}/\mathcal{A}(\mathbf{q}) \cap (\bigcup_{i=1}^{\mathbf{q}} \mathcal{B}_i) = \emptyset\} \tag{2.3}$$

is called the free space. Any configuration $\mathbf{q}$ in $\mathcal{C}_{free}$ is called a free configuration.

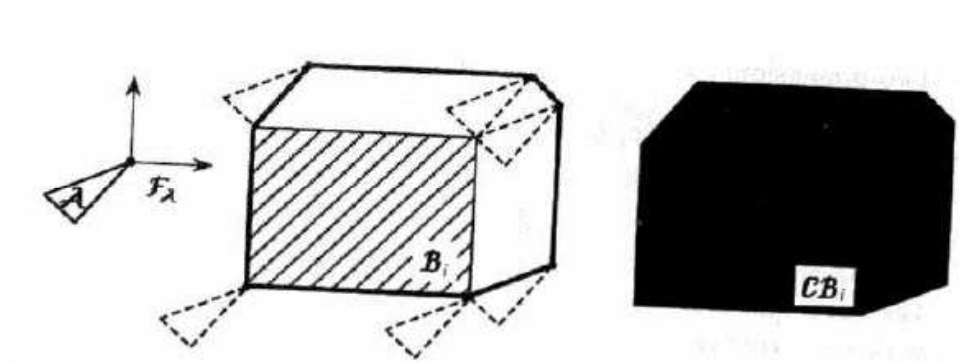

Figure 2.1: Obstacle representation in configuration space

2.1.2 State Space Formulation

Notion of $\mathcal{S}$-Space

The $\mathcal{C}$-space is apt for geometric path planning, *i.e.*, it takes into consideration the kinematic constraints of $\mathcal{A}$ but as soon as we talk about real systems like vehicles, the

dynamic constraints like maximum velocity and acceleration. play a critical role. The $\mathcal{S}$-space takes into consideration these constraints. Or in other words the $\mathcal{S}$-space has both the $\mathcal{C}$-space parameters and their derivatives for parameters. In the most general case it could be $\mathbf{s} = (x, y, \theta, v)$.

Obstacles in $\mathcal{S}$-space

The obstacles whose image in $\mathcal{W}$ are $\mathcal{B}_i$'s map to a region $\mathcal{SB}_i$ in $\mathcal{S}$.

$$\mathcal{SB}_i = \{\mathbf{s} \in \mathcal{S}/\mathcal{A}(\mathbf{s}) \cap \mathcal{B}_i \neq \emptyset\} \tag{2.4}$$

This region is also called the $\mathcal{S}$-obstacle of $\mathcal{B}_i$. The union of all the $\mathcal{S}$-obstacles:

$$\bigcup_{i=1}^{N} \mathcal{SB}_i \tag{2.5}$$

is called the $\mathcal{S}$-obstacle region and the set

$$\mathcal{S}_{free} = \mathcal{S}/\bigcup_{i=1}^{N} \mathcal{SB}_i = \{\mathbf{s} \in \mathcal{S}/\mathcal{A}(\mathbf{s}) \cap (\bigcup_{i=1}^{N} \mathcal{B}_i) = \emptyset\} \tag{2.6}$$

is called the free space. Any state $\mathbf{s}$ in $\mathcal{S}_{free}$ is called a free state. Not surprisingly, the $\mathcal{SB}_i$ is same as the $\mathcal{CB}_i$ for all obstacles. This is due to the fact that irrespective of the velocity or other derivative parameters of $\mathcal{C}$ that are incorporated in $\mathcal{S}$, as long as the position is the same there is a collision. For example, it does not matter if the collision between $\mathcal{A}$ and $\mathcal{B}_i$ takes place when $\mathcal{A}$ has a velocity $v_{\mathcal{A}} = 0$ or 50 m/s.

2.1.3 Configuration-Time Space Formulation

Notion of $\mathcal{CT}$-space

The $\mathcal{C}$ and $\mathcal{S}$-space formulation of the motion planning does not extend to deal with time constraints, for example, if some of the $\mathcal{B}_i$'s are moving, the $\mathcal{CB}_i$ corresponding to the obstacle is time-dependant. Hence, there is a need to incorporate the *time* dimension. The easiest extension is the $\mathcal{CT}$-space [1]. Here, $\mathcal{B}_i(t)$ represents the region of $\mathcal{W}$ occupied by $\mathcal{B}_i$ at time t and we assume that the regions $\mathcal{B}_i(t)$ are known $\forall$ i. This assumption requires that the future behavior of the obstacles is not affected by the movement of $\mathcal{A}$.

Obstacles in $\mathcal{CT}$-space

Now, since the location of the $\mathcal{C}$-obstacles may vary with t, it impossible to represent them in $\mathcal{C}$, in such a way that we can still reduce the motion of $\mathcal{A}$ to the motion of a point among fixed constraints. Hence we add the time dimension to $\mathcal{C}$. This results in the new space $\mathcal{CT}$=$\mathcal{C}$ x $[0, +\infty)$. $\mathcal{A}$ maps in $\mathcal{CT}$ to a configuration-time ($\mathbf{q}$,t) meaning that $\mathcal{A}$'s configuration at time t is $\mathbf{q}$. Every obstacle $\mathcal{B}_i$ maps in $\mathcal{CT}$ to a stationary region $\mathcal{CTB}_i$ called a $\mathcal{CT}$-obstacle defined by:

$$\mathcal{CTB}_i = \{(\mathbf{q}, t)/\mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i(t) \neq \emptyset\} \tag{2.7}$$

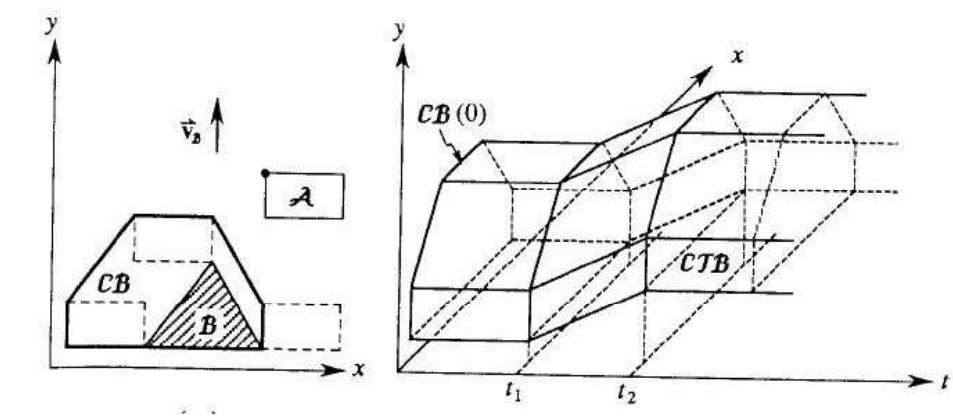

Figure 2.2: $\mathcal{A}$ translates freely without any rotation in the plane. The obstacle $\mathcal{B}$ translates at fixed orientation with constant velocity v_B between the time instants t1 and t2. v_B is parallel to the y-axis and points upwards. At any other time $t \notin [t1, t2]$, $\mathcal{B}$ is motionless. The $\mathcal{CT}$-obstacle is obtained by sweeping $\mathcal{CB}$ along a line that is perpendicular to the xy-plane when $t \notin [t1, t2]$ and oriented along the vector $(0, v_B, 1)$ when $t \in [t1, t2]$.

Fig.2.2 shows an example of obstacle representation in the $\mathcal{CT}$-space. We define free space in $\mathcal{CT}$ as:

$$\mathcal{CT}_{free} = \mathcal{CT} / \bigcup_{i=1}^{N} \mathcal{CTB}_i \tag{2.8}$$

2.1.4 State-Time Space Formulation

The $\mathcal{TS}$-space is a tool to formulate problems of trajectory planning in dynamic workspaces. This concept was proposed in [15]. $\mathcal{TS}$-space permits the study of the different aspects of dynamic trajectory planning, *i.e.* moving obstacles and dynamic constraints in an unified way. It stems from two concepts we have already seen: the $\mathcal{CT}$-space and the $\mathcal{S}$-space, the space of the configuration parameters and their derivatives. In this framework, the constraints imposed by both the moving obstacles and the dynamic constraints can be represented by static forbidden regions.

A state-time of $\mathcal{A}$ is represented by adding the time dimension to a state, hence it is represented as $\mathcal{A}(\mathbf{s},\mathrm{t})$. As explained in the case of $\mathcal{S}$, where the $\mathcal{S}$-obstacles was same as the $\mathcal{C}$-obstacles, here again we have the $\mathcal{TSB}$ same as the $\mathcal{CT}$-obstacle. On the other hand, a state-time $(\mathbf{s},\mathrm{t})$ is admissible if and only if it does not violate the dynamic constraints on the maximum velocities and acceleration.

In fact in order to perform navigation, we are not obliged to follow the above mentioned formulations. For example, most of the recent algorithms in the literature do not always use these formulations, but other notions of space, for example the velocity space *etc* (*e.g.*, Velocity obstacles, [7], Dynamic Window Approach, [5], Global Dynamic Window Approach, [16] *etc*,) but a knowledge of these formalizations is important to analyse and understand the different approaches.

2.2 Deliberative Aprroaches

$\mathcal{C}$-space formulations were widely used during the 80's. Navigation methods like the Visibility graph method, Retraction method, Freeway method, Silhouette method, [1] calculate the obstacle free area of $\mathcal{C}$ which is then used for motion planning. The interested reader is referred to [1] for an extensive review of these algorithms.

Advantages

On the whole, these algorithms share an advantage in the sense that, the safety issue is taken care of during the free space calculating stage. Hence it is done only once and can be performed off-line.

Disadvantages

- A major disadvantage is that the calculation of free space is computationally very expensive. In higher dimensions, it is almost impossible to capture the free space connectivity.
- A second disadvantage is that, if the assumed model of the environment changes to a small extent, the free space connectivity is altered. Hence it has to be re-calculated. This is a major issue especially in highly dynamic environments.

Hence, global methods are not of much practical use for a dynamic environment and for the same reason we will not go into the details of these methods.

2.3 Reactive Approaches

These methods calculate only a local (small region around the robotic system) free space connectivity rather than the entire free space connectivity.

Advantages

- The locality is a characterisitc feature of reactive methods that helps reduce the computational effort.
- Dynamic environments and static environments are given same treatment.
- It is easy to incorporate reactive methods with a motion planner such that we can rapidly verify if a state is a collision state, unlike the deliberative methods.

Disadvantages

- The reactive nature of the algorithms make them susceptible to local minima problems and sometimes may not be able to reach the goal, if used with a motion planner.
- The path calculated by a reactive motion planner is not an optimal solution. Hence this is an issue if the problem requirement demands an optimal solution.

Despite these disadvantages, reactive methods are the only methods that can work in dynamic environments and can be incorporated with motion planners. Few algorithms that perform reactive navigation are reviewed below.

2.3.1 Potential Field Approaches

Potential Field methods introduced in [2], can be classified under both the categories of deliberative and reactive methods depending on the potential function involved. For example, a simple distance from the goal potential would only result in a reactive method *etc*. More complex potentials (with special attention to degenerate cases) can make it a global method. Using the notion of $\mathcal{C}$-space, $\mathcal{A}$ is viewed as a point. A path in this space is constructed by exploring the $\mathcal{C}$-space with heuristics. The heuristics used in the potential field method is based on the physical potentials. Obstacles exert a repulsive force on $\mathcal{A}$ and the goal exerts an attractive force. All potentials at each point in $\mathcal{C}$ are added vectorially to produce a resultant force acting on that point. This potential value implicitly defines a path by acting as a direction value. The path then forms a gradient descent to the goal.

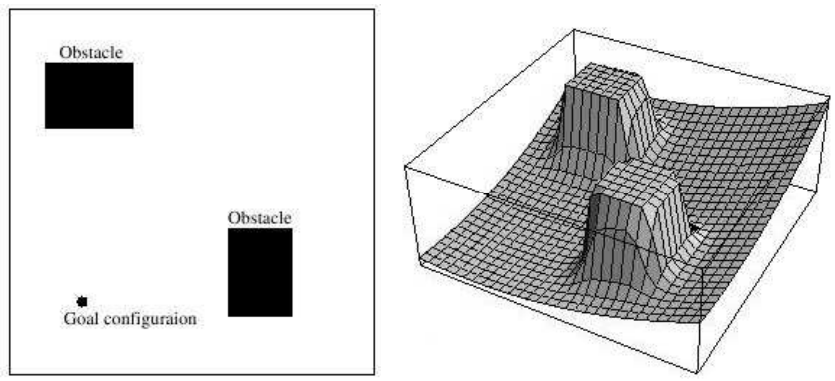

Figure 2.3: a)Workspace $\mathcal{W}$, also the $\mathcal{C}$-space in this case, containing two obstacles. b) After calculating the attractive and repulsive forces at each point of $\mathcal{C}$.

Advantages

- The path is not calculated in advance; rather the path is implicitly represented by the potential function. Hence, the computational complexity is low and motion commands can be generated very easily.

Disadvantages

- In some cases, there is a possibility that the net force on the robot sums up to be zero at places other than the goal position. Although these cases can be dealt with separately by providing a slight motion (considering it as a degenerate case of the algorithm) in some direction, there are other possibilities of the robot being stuck in the local minima.

- No consideration is given to the dynamics of $\mathcal{A}$ and of the moving obstacles. Hence the same reasoning in Section 1.1.1 is applicable here.

2.3.2 Vector Field Histograms

This methods compute a direction for $\mathcal{A}$ to head in (action to be performed at the next time step), in $\mathcal{W}$ or $\mathcal{C}$. The inputs to the algorithm are the sensor values at time instant t. The controller builds a histogram (Vector Field Histogram (VFH) [3], VFH+ [4], VFH* [10] *etc*,) using the sensor values and looks for gaps in the histograms which could help in identifying possible directions of motion. This method is very efficient for identifying a directional command for collision-free movement in real-time.

Certain defects of the first proposed VFH method were identified and rectified in the future versions. For instance, VFH+ considers the size of the robot and expands the obstacles in the histogram (quite similar to idea of $\mathcal{C}$-spaces, but does not explicitly build the configuration space), it also takes into account the robot trajectory using parameters like the minimum steering radius etc. Finally, it uses a cost function to determine the best action among the available (maximum of three) actions.

Disadvantages

- This method too does not take into account the dynamics of the robot, and the obstacles.

- Another problem with the VFH methods in general is that they build the histogram in the angle space. They also choose an action in the wide collision-free angle space. This could be a problem, for example, when the robot is near an obstacle even a small collision-free distance will be seen as a large collision-free angle. On the other hand, if the robot is far from the same opening, it will regard the same collision-free distance as a small collision-free angle.

2.3.3 Dynamic Window Approach

This approach proposed by [5] works on the velocity space of the robotic system $\mathcal{A}$. The advantage of working in the velocity space is that the kinematic and dynamic constraints can easily be taken into consideration by restricting the search space only to the reachable velocity space. The search space is the set of tuples (v, ω) the translational and rotational velocities that are achievable by $\mathcal{A}$. Among all the tuples, those are selected that, if selected and executed would allow $\mathcal{A}$ to come to a stop before hitting the obstacle. These velocities are called admissible velocities.

The search is further restricted by a *dynamic window* giving this method the name. This reflects the dynamic limitations of $\mathcal{A}$, *i.e.* given $\mathcal{A}$'s current velocity and acceleration capabilities, the dynamic window has those velocities that can be achieved within a given time interval. Fig.2.4 illustrates the subdivision of the search space in the DWA.

Disadvantages

- The DWA does not take into consideration the dynamics of the obstacles although

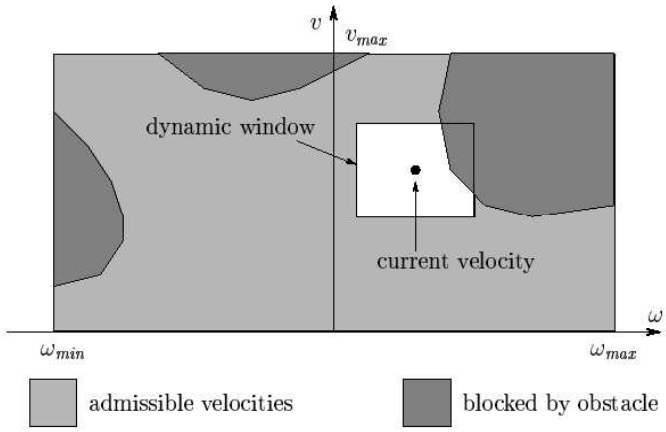

Figure 2.4: Search space in the DWA. The space of all possible velocity commands is divided into admissible and blocked regions. The small rectangle corresponds to the dynamic window consisting of velocities reachable by $\mathcal{A}$ within the specified time interval.

it takes into account the dynamics of the robotic system. The disadvatages of not considering the dynamics of the obstacles was already illustrated in Fig.1.1. The same problem exists here.

The reactive nature of this algorithm was overcome by the Global Dynamic Window Approach in [16], but even this method does not take into consideration any dynamics.

2.3.4 Linear Velocity Obstacles

This interesting method proposed in [7], assumes that all obstacles in $\mathcal{W}$ move along arbitrary trajectories and their instantaneous state (position or velocity) are known or at least measurable (model assumption). Like the DWA, this method too operates in the velocity space. $\mathcal{A}$ and all $\mathcal{B}$s are assumed to be circular, so that $\mathcal{C}$ is easy to calculate. A colliding relative velocity cone is defined with respect to every moving obstacle (as the $\mathcal{CB}$ s are always circular, it is definitely a cone). Velocities in this cone lead to a collision with the obstacle at arbitrary times. This is calculated for all $\mathcal{B}$s in $\mathcal{W}$. This whole set of velocities forms the set of non-admissible velocities.

The dynamic constraints are used to reduce the search space to only the set of reachable velocities depending on the acceleration limits and current velocity of $\mathcal{A}$. The complement set of the non-admissible velocities in the set of reachable velocities is the set of velocities that avoid all obstacles.

Fig.2.5 shows geometrically how the non-admissible velocities region is calculated.

Advantages

- This is one of the few algorithms that takes into consideration the dynamics of both $\mathcal{A}$ and $\mathcal{B}$s. Hence it does not suffer from the short comings discussed in Section 1.1.1.

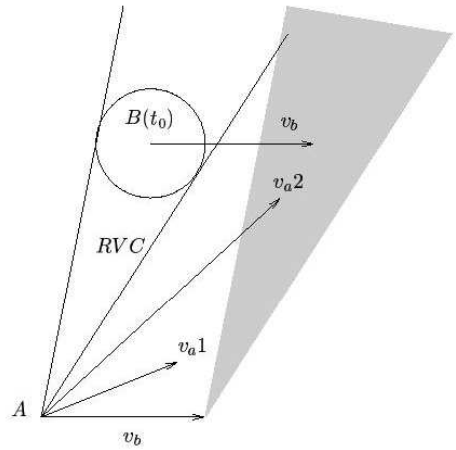

Figure 2.5: Velocity space in the VO. The space of all velocity commands is divided into admissible and non-admissible regions. Here RVC is the relative velocity cone that will lead to a collision between $\mathcal{A}$ and $\mathcal{B}$ constituing the non-admissible region. $\mathcal{B}$ has a velocity $v_{\mathcal{B}}$ shown as v_b. To calculate the absolute velocity of $\mathcal{A}$ that induces a collision, a Minkowski sum is performed between RVC and v_b. Hence, here velocity v_{a1} avoids a collision with $\mathcal{B}$ while velocity v_{a2} causes a collision.

- The method is computationally very efficient as the search space is not high-dimensional (velocity space).
- It can take into account any number of obstacles, both dynamic and stationary.
- The motion command is generated as a velocity selected from the set of reachable admissible velocities which in turn is generated using simple boolean operations on sets.

Disadvantages

Although this is one of the most popular and efficient methods, the following problem is identified.

- For implementation issues, the VO method uses a time-horizon t_h calculated using the distance proximity between obstacles and $\mathcal{A}$. The idea is based on clipping the set of non-admissible velocities, with t_h *i.e.*. all velcoties that lead to a collision after t_h are moved to set of admissible velocities. This is a typical situation as shown in Fig.1.2. Collision free trajectory is gauranteed until time t_h but we have no idea if we can find another velocity after time t_h to gaurantee collision. Hence for the reasons mentioned in Section 1.1.2, this method too does not ensure safety.

2.4 *k-step* Approaches

These approaches bridge the gap between the deliberative and reactive approaches. Their look ahead is more than the latter but less than the former. Instead of planning only the

next time instant, they plan a few steps (say k) ahead. Such an extension to the VFH+ method discussed in Section 2.3.2 resulted in VFH*, which tries to address the issue of the susceptibility to local minima. VFH* takes into account the local nature of VFH *i.e.*, the lack of a look ahead, and circumnavigates the problem by building a search tree at a given robot position. It takes into consideration the primary candidate directions (current action) and the projected candidate directions (future actions) by constructing a VFH+ histogram at each projected state to bring in a flavor of look-ahead.

Even though some of these work address real time motion planning a few only consider highly changing environment, performing fast replanning using probabilistic techniques, though for simple systems *e.g.*, [6] and [17]. It is argued in [9] that when complex robotic systems and environments are considered, the real time constraints have to be explicitly into account in the form of a time-bound for making a decision, *etc.* In such a case a complete trajectory to the goal cannot be computed in general and only partial plans can be found. They have thus introduced the Partial Motion Planning (PMP) method with an any-time flavor that returns the best trajectory planned (till that instant) whenever it is polled for a solution. This is the framework that this project assumes for navigation.

On the other hand, when dealing with partial plans, it becomes of the utmost importance to consider the behaviour of the system at the end of the trajectory. What if a car ends its trajectory in front of a wall at high speed? It becomes clear that strong guarantees should be given to this trajectory in order to handle the safety issues raised by such a partial planning.

Hence, there is a need for a *collision-checker* module that takes in a state $\mathbf{s_i}$ of the robotic system as input and produces a boolean output as to whether $\mathbf{s_i}$ is safe. Now, how do we define *safe* for a robotic system in a dynamic environment?

Def. 1 *A state* $\mathbf{s_i}$ *is defined as safe iff it is not already in collision with an obstacle and there is at least one control input* $\phi_i \in \Phi$*, such that, the terminal state* $\mathbf{s_j}$*, the result of applying* ϕ_i *at* $\mathbf{s_i}$ *is also safe.*

This work proposes a collision-checker that takes into consideration the dynamics of the system and of the moving obstacles, does not incorporate a time-horizon and returns a boolean value for a state verifying if it is *safe*.

2.5 Conclusion

The methods we have reviewed in this chapter have their own merits and de-merits according to their respective assumptions. In general, the earliest reactive methods did not take into account the dynamics of the system, or the dynamics of the obstacles. The methods using the velocity space take into account the dynamics and the kinematics, thanks to the simplicity of the velocity space. But these methods use a time horizon which leads to problems mentioned in Section 1.1.2. How can we be sure that after choosing a velocity and executing it, we can always find another velocity in the reachable admissable region at the next time instant? What if we end the trajectory at an instant where the set of reachable admissible velocities is a null set? This ambiguity caused by the time-horizon is one that this work aims to overcome.

Hence, in conclusion, there is a need to develop a navigation algorithm that takes into consideration the dynamics of the robot and of the moving obstacles and does not incorporate a time horizon. The navigation approach taken in this project is the PMP framework, and the collision checker proposed is the Inevitable Collision State-checker or, $\mathcal{ICS}$*-checker*. The theoretical framework for calculating the ICS is presented in the next Chapter.

Chapter 3

Inevitable Collision States and Obstacles

An *inevitable collision state*, proposed in [12] for a robotic system can be defined as a state for which, no matter what the future trajectory followed by the system is, a collision with an obstacle eventually occurs. An inevitable collision state takes into account the dynamics of both the system and the obstacles, fixed or moving. This concept is very general and can be useful both for navigation and motion planning purposes: for its own safety, a robotic system should never find itself in an inevitable collision state.

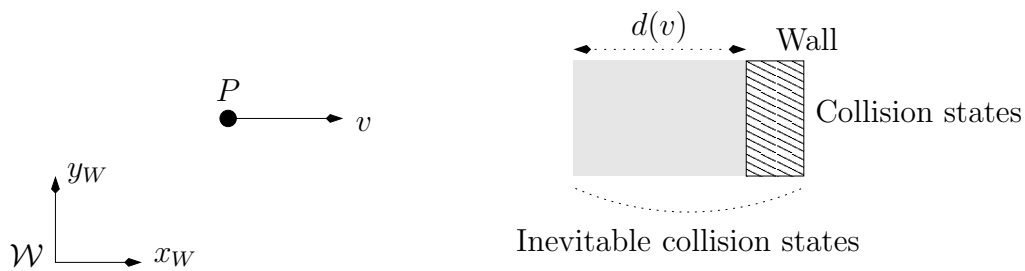

Figure 3.1: Collision states vs inevitable collision states.

Consider Fig.3.1, let P be a point mass that can only move to the right with a variable speed. A state of P is characterised by its position (x, y) and its speed v. If the workspace $\mathcal{W}$ features a wall, the states whose position corresponds to the wall are obviously collision states. On the other hand, assuming that it takes P a certain distance $d(v)$ to slow down and stop, the states corresponding to the wall *and* the states located at a distance less than $d(v)$ left of the wall are such that, when P is in such a state, no matter what it does in the future, a collision will occur. These states are inevitable collision states for P. Clearly, for P's own safety, when it is moving at speed v, it should never be in one of these inevitable collision states. The size of the inevitable collision states region, *i.e.* the grey region located to the left of the wall, depends on the distance $d(v)$ which in turns depends on the current speed of P. Assuming that $d(v)$ varies linearly with v, the complete set of inevitable collision states is a prism embedded in the state space $\mathcal{S}$ of P (Fig.3.2).

In general, an *inevitable collision state* for a given robotic system can be defined as a state for which, no matter what the future trajectory followed by the system is, a collision

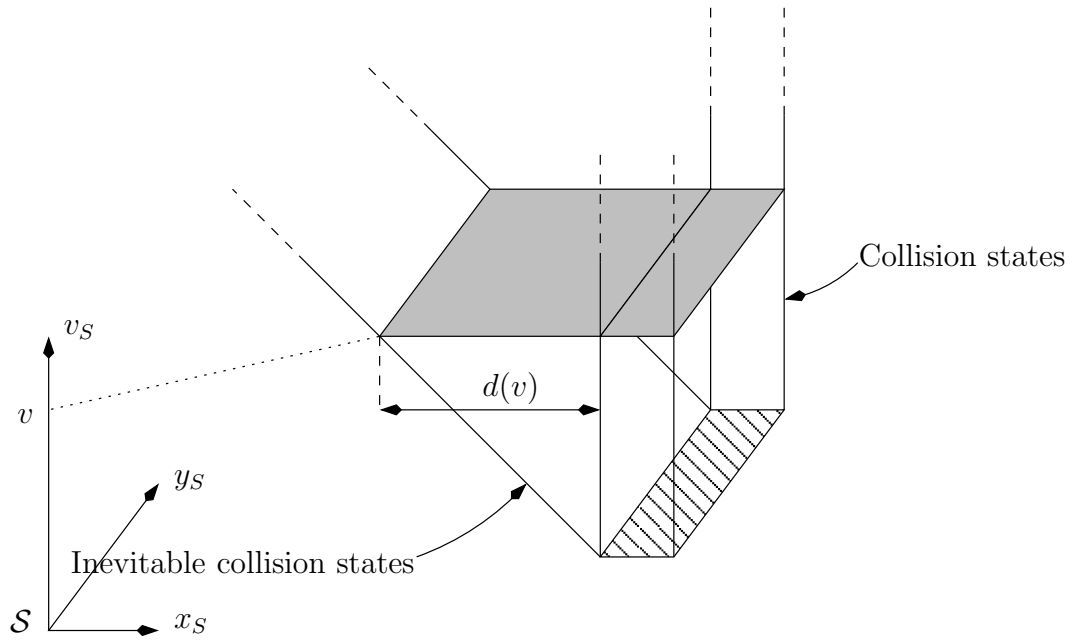

Figure 3.2: Full representation in the xyv state space $\mathcal{S}$ of P of the inevitable collision states corresponding to the situation depicted in Fig.3.1.

eventually occurs with an obstacle of the environment. Similarly, it is possible to define an *inevitable collision obstacle* as the set of inevitable collision states yielding a collision with a particular obstacle.

3.1 Notations and Preliminary Definitions

Before defining the inevitable collision states and obstacles, useful definitions and notations are introduced. Let $\mathcal{A}$ denote a robotic system. It is assumed that its dynamics can be described by a differential equation such as: $\dot{s} = f(\mathbf{s}, u)$ where $\mathbf{s} \in \mathcal{S}$ 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 $\phi \in \Phi$ denote a *control input*, *i.e.* a time-sequence of controls. ϕ represents a trajectory for $\mathcal{A}$. Starting from an initial state $\mathbf{s_0}$ (at time 0) and under the action of a control input ϕ, the state of $\mathcal{A}$ at time t is denoted by $\phi(\mathbf{s_0}, t)$

Given a control input ϕ and a state $\mathbf{s_0}$ (at time 0), a state $\mathbf{s}$ is *reachable from* $\mathbf{s_0}$ *by* ϕ iff $\exists t, \phi(\mathbf{s_0}, t) = \mathbf{s}$. Let $\mathcal{R}(\mathbf{s_0}, \phi)$ denote the set of states reachable from $\mathbf{s_0}$ by ϕ. Likewise, $\mathcal{R}(\mathbf{s_0})$ denotes the set of states $\mathbf{s}$ reachable from $\mathbf{s_0}$, *i.e.* such that $\exists \phi, \mathbf{s} \in \mathcal{R}(\mathbf{s_0}, \phi)$:

$$\begin{aligned} \mathcal{R}(\mathbf{s_0}, \phi) &= \{\mathbf{s} \in \mathcal{S} | \exists t, \phi(\mathbf{s_0}, t) = \mathbf{s}\} \\ \mathcal{R}(\mathbf{s_0}) &= \{\mathbf{s} \in \mathcal{S} | \exists \phi, \mathbf{s} \in \mathcal{R}(\mathbf{s_0}, \phi)\} \end{aligned}$$

Introducing $\phi^{-1}(\mathbf{s_0}, \text{t})$ to denote the state $\mathbf{s}$ such that $\phi(\mathbf{s}, t) = \mathbf{s_0}$, it is possible to define $\mathcal{R}^{-1}(\mathbf{s_0})$ (resp. $\mathcal{R}^{-1}(\mathbf{s_0}, \phi)$), as the set of states from which it is possible to reach $\mathbf{s_0}$ (resp. to reach $\mathbf{s_0}$ by ϕ):

$$\mathcal{R}^{-1}(\mathbf{s_0}, \phi) = \{\mathbf{s} \in \mathcal{S} | \exists t, \phi(\mathbf{s}, t) = \mathbf{s_0} \Leftrightarrow \phi^{-1}(\mathbf{s_0}, t) = \mathbf{s}\}$$

$$\mathcal{R}^{-1}(\mathbf{s_0}) \quad = \quad \{\mathbf{s} \in \mathcal{S} | \exists \phi, \mathbf{s} \in \mathcal{R}^{-1}(\mathbf{s_0}, \phi)\}$$

Let $\mathcal{W}B$ denote an obstacle. When $\mathcal{W}B$ is moving, $\mathcal{W}B(t)$ represents the subset of $\mathcal{W}$ occupied by $\mathcal{W}B$ at time t. When $\mathcal{W}B$ is fixed, the time index is omitted: $\forall t, \mathcal{W}B(t) = \mathcal{W}B(0) = \mathcal{W}B$. Every obstacle has an image in the state space: the set of states yielding a collision between the robotic system and an obstacle $\mathcal{W}B(t)$ determines the *state obstacle* of $\mathcal{W}B(t)$ which is denoted $\mathcal{B}(t)$. $\mathcal{B}(t) = \{s \in \mathcal{S} | \mathcal{A}(\mathbf{s}) \cap \mathcal{W}B(t) \neq \emptyset\}$, where $\mathcal{A}(\mathbf{s})$ denotes the closed subset of $\mathcal{W}$ occupied by $\mathcal{A}$ in state $\mathbf{s}$. Once again, when $\mathcal{W}B$ is fixed, the time index is omitted. A state $\mathbf{s}$ is a *collision state at time t* iff $\exists \mathcal{B}, \mathbf{s} \in \mathcal{B}(t)$. In this case, $\mathbf{s}$ is a *collision state at time t with* $\mathcal{B}$.

The document places itself in the state space framework unless otherwise mentioned. For the sake of simplicity, state obstacles are called obstacles only and the time index is indicated only when necessary.

3.2 Inevitable Collision States and Obstacles

Based on the definitions and notations introduced in the previous section, the inevitable collision states and the inevitable collision obstacles are formally defined.

Def. 2 (Inevitable Collision State) *Given a control input ϕ, a state $\mathbf{s}$ is an **inevitable collision state for** ϕ iff $\exists t$ such that $\phi(\mathbf{s}, t)$ is a collision state at time t. Now, a state $\mathbf{s}$ is an **inevitable collision state** iff $\forall \phi, \exists t$ such that $\phi(\mathbf{s}, t)$ is a collision state at time t. Likewise, $\mathbf{s}$ is an **inevitable collision state with $\mathcal{B}$ for** ϕ iff $\exists t$ such that $\phi(\mathbf{s}, t)$ is a collision state at time t with $\mathcal{B}$. Finally, $\mathbf{s}$ is an **inevitable collision state with $\mathcal{B}$** iff $\forall \phi, \exists t$ such that $\phi(\mathbf{s}, t)$ is a collision state at time t with $\mathcal{B}$.*

Def. 3 (Inevitable Collision Obstacle) *Given an obstacle $\mathcal{B}$ and a control input ϕ, $ICO(\mathcal{B}, \phi)$, the **inevitable collision obstacle of $\mathcal{B}$ for** ϕ is defined as:*

$$\begin{aligned} ICO(\mathcal{B}, \phi) &= \{\mathbf{s} \in \mathcal{S} | \mathbf{s} \textit{ is an inevitable collision state with } \mathcal{B} \textit{ for } \phi\} \\ &= \{\mathbf{s} \in \mathcal{S} | \exists t, \phi(\mathbf{s}, t) \textit{ is a collision state at time } t \textit{ with } \mathcal{B}\} \\ &= \{\mathbf{s} \in \mathcal{S} | \exists t, \phi(\mathbf{s}, t) \in \mathcal{B}(t)\} \end{aligned}$$

*Now, $ICO(\mathcal{B})$, the **inevitable collision obstacle of $\mathcal{B}$**, is defined as:*

$$\begin{aligned} ICO(\mathcal{B}) &= \{\mathbf{s} \in \mathcal{S} | \mathbf{s} \textit{ is an inevitable collision state with } \mathcal{B}\} \\ &= \{\mathbf{s} \in \mathcal{S} | \forall \phi, \exists t, \phi(\mathbf{s}, t) \textit{ is a collision state at time } t \textit{ with } \mathcal{B}\} \\ &= \{\mathbf{s} \in \mathcal{S} | \forall \phi, \exists t, \phi(\mathbf{s}, t) \in \mathcal{B}(t)\} \end{aligned}$$

Based upon the two definitions above, the following property can be established. It shows that $ICO(\mathcal{B})$ can be derived from the $ICO(\mathcal{B}, \phi)$ for every possible control input ϕ.

Property 1 (Control Inputs Intersection)

$$ICO(\mathcal{B}) = \bigcap_{\Phi} ICO(\mathcal{B}, \phi)$$

Proof:

$$\begin{aligned}
\mathbf{s} \in ICO(\mathcal{B}) &\Leftrightarrow \forall \phi, \exists t, \phi(\mathbf{s}, t) \text{ is a collision state at time } t \text{ with } \mathcal{B} \\
&\Leftrightarrow \forall \phi, \mathbf{s} \in ICO(\mathcal{B}, \phi) \\
&\Leftrightarrow \mathbf{s} \in \bigcap_{\Phi} ICO(\mathcal{B}, \phi)
\end{aligned}$$

■

Assuming now that $\mathcal{B}$ is the union of a set of obstacles, $\mathcal{B} = \bigcup_i \mathcal{B}_i$, the following property can be established. It shows that $ICO(\mathcal{B}, \phi)$ can be derived from $ICO(\mathcal{B}_i, \phi)$ for every subset $\mathcal{B}_i$.

Property 2 (Obstacles Union)

$$ICO(\bigcup_i \mathcal{B}_i, \phi) = \bigcup_i ICO(\mathcal{B}_i, \phi)$$

Proof:

$$\begin{aligned}
\mathbf{s} \in ICO(\textstyle\bigcup_i \mathcal{B}_i, \phi) &\Leftrightarrow \exists t, \phi(\mathbf{s}, t) \text{ is a collision state at time } t \text{ with } \textstyle\bigcup_i \mathcal{B}_i \\
&\Leftrightarrow \exists \mathcal{B}_i, \exists t, \phi(\mathbf{s}, t) \text{ is a collision state at time } t \text{ with } \mathcal{B}_i \\
&\Leftrightarrow \exists \mathcal{B}_i, \mathbf{s} \in ICO(\mathcal{B}_i, \phi) \\
&\Leftrightarrow \mathbf{s} \in \textstyle\bigcup_i ICO(\mathcal{B}_i, \phi)
\end{aligned}$$

■

Combining the two properties above, the following property is derived. It is the property that permits the formal characterisation of the inevitable collision obstacles for a given robotic system.

Property 3 (ICO Characterisation) *Let* $\mathcal{B} = \bigcup_i \mathcal{B}_i$,

$$ICO(\mathcal{B}) = \bigcap_{\Phi} \bigcup_i ICO(\mathcal{B}_i, \phi)$$

Proof:

$$ICO(\mathcal{B}) \overset{1}{=} \bigcap_{\Phi} ICO(\mathcal{B}, \phi) \overset{2}{=} \bigcap_{\Phi} \bigcup_i ICO(\mathcal{B}_i, \phi)$$

■

Consider property 1 (and property 3), it establishes that $ICO(\mathcal{B})$ can be derived from the $ICO(\mathcal{B}, \phi)$ for every possible control input ϕ. In general, there is an infinite number of control inputs which leaves little hope of being actually able to compute $ICO(\mathcal{B})$. Fortunately, it is possible to establish a property which is of a vital practical value since it shows how to compute a conservative approximation of $ICO(\mathcal{B})$ by using a subset only of the whole set of possible control inputs.

Property 4 (ICO Approximation) *Let $\mathcal{I}$ denote a subset of the set of possible control inputs Φ,*

$$ICO(\mathcal{B}) \subset \bigcap_{\mathcal{I}} ICO(\mathcal{B}, \phi)$$

Proof:

$$\begin{aligned} ICO(\mathcal{B}) &\stackrel{1}{=} \bigcap_{\mathcal{I} \cup \overline{\mathcal{I}}} ICO(\mathcal{B}, \phi) \\ &= \bigcap_{\mathcal{I}} ICO(\mathcal{B}, \phi) \cap \bigcap_{\overline{\mathcal{I}}} ICO(\mathcal{B}, \phi) \\ &\subseteq \bigcap_{\mathcal{I}} ICO(\mathcal{B}, \phi) \end{aligned}$$

■

3.3 Calculating $ICO(\mathcal{B})$

Given $\mathcal{A}$'s state space equation, in order to calculate the ICO for the obstacles in $\mathcal{W}$, we move on to the $\mathcal{S}$-space of $\mathcal{A}$. Initially we decide on the control inputs that will be considered to approximate the ICO. These can be (but are not limited to) braking trajectories, where $\mathcal{A}$ starts at the current state $\mathbf{s_i}$ and breaks until the velocity is zero. The braking trajectories also depend on $\mathcal{A}$'s dynamic and kinematic constraints.

Next we calculate the $ICO(\mathcal{B}_i, \phi_j)$ for each obstacle in $\mathcal{W}$ and take the union of these sets for each ϕ_j in Φ. Then we calculate the intersection of all these sets to identify the ICO for each obstacle.

The interest of these properties to characterise inevitable collision obstacles for obstacles in $\mathcal{W}$ appears in the next chapter which discusses a car-like vehicle case study.

Chapter 4

Case Study: Car-Like Vehicle

A car-like vehicle is a typical example of system subject to kinematic constraints since it cannot translate and rotate freely in the workspace $\mathcal{W}$. The constraints affecting the motion of a car-like robotic system translate into an equation and an inequation involving the velocity parameters of the robot. They are said to be *non-holonomic*. They do not restrict the set of configurations reachable from a given configuration, but they do restrict the space of velocities achievable at any configuration of the robotic system which follows the constraint below,

$$\dot{y}\cos\theta = \dot{x}\sin\theta \tag{4.1}$$

where x and y are the coordinates of the midpoint R between the two rear wheels and $\theta \in [0, 2\pi)$ is the angle between the x-axis of $\mathcal{F}_{\mathcal{W}}$ attached to the workspace and main axis of the car. This is shown in the bicycle model in Fig.4.1.

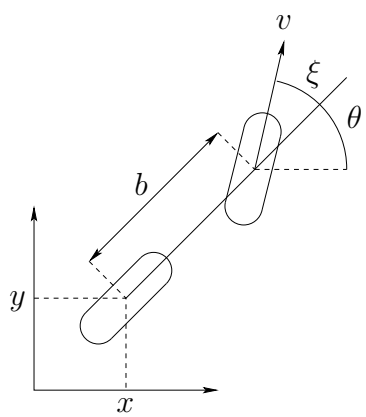

Figure 4.1: The car-like vehicle $\mathcal{A}$ (bicycle model).

4.1 Characterization of the $\mathcal{ICS}$

4.1.1 Statement of the Problem

Let us consider a robotic system $\mathcal{A}$ that moves like a car-like vehicle and whose dynamics follow a bicycle model. A state of $\mathcal{A}$ is defined by the 4-tuple $\mathbf{s} = (x, y, \theta, v)$ where (x, y)

are the coordinates of the rear wheel, θ is the main orientation of $\mathcal{A}$, and v is the linear velocity of the front wheel (Fig. 4.1). A control of $\mathcal{A}$ is defined by the couple (u^ξ, u^v) where u^ξ is the steering angle and u^v the linear acceleration. The motion of $\mathcal{A}$ is governed by the following differential equations:

$$\begin{cases} \dot{x} &= v\cos\theta\cos u^\xi \\ \dot{y} &= v\sin\theta\cos u^\xi \\ \dot{\theta} &= v\sin u^\xi / b \\ \dot{v} &= u^v \end{cases}$$

with $|u^\xi| \leq \xi_{\max}$ and $|u^v| \leq u^v_{\max}$. b is the wheelbase of $\mathcal{A}$. $\mathcal{A}$ moves on a planar workspace $\mathcal{W}$ cluttered up with a set of polygonal obstacles $\mathcal{W}B_i$. Although the state space $\mathcal{S}$ of $\mathcal{A}$ is four-dimensional, it is not attempted to compute the inevitable collision obstacles in the full four dimensional state space. Instead, the structure of $\mathcal{S}$ is exploited and the inevitable collision obstacles are computed in two-dimensional slices of $\mathcal{S}$ only. The slices considered are slices with constant θ and v. Such slices are interesting because it is straightforward to compute, for such a slice, the state obstacles $\mathcal{B}_i$.

4.1.2 Trajectory Parameterization

The trajectory of a car-like robot with non-zero steering angle and constant velocity can be parameterized in terms of its (v_0,θ_0) slice, where v_0 and θ_0 are the initial values of velocity and orientation. Considering this slice we can integrate the above differential equations to get,

$$\theta(t) = \theta_0 + \frac{v_0}{b}\sin u^\xi t \tag{4.2}$$

$$x(t) = x(0) + \frac{b}{\tan u^\xi}(\sin(\theta_0 + \frac{v_0}{b}\sin u^\xi t) - \sin\theta_0) \tag{4.3}$$

$$y(t) = y(0) - \frac{b}{\tan u^\xi}(\cos(\theta_0 + \frac{v_0}{b}\sin u^\xi t) - \cos\theta_0) \tag{4.4}$$

Detecting collision states

Thanks to these equations, it is easy to find the position of $\mathcal{A}$ at time $t = 0$ for collision with the obstacle at any arbitrary time t. We just need to backpropagate in time from the values of $\mathcal{B}_x$(t) and $\mathcal{B}_y$(t) for (x(t),y(t)) and solve for (x(0),y(0)). Thus, $\mathbf{s_0}$=(x(0),y(0)) is an $\mathcal{ICS}$ for $\mathcal{B}$ for this control input.

Now, let $ICO_\mathcal{I}(\mathcal{B})$ denote $\bigcap_\mathcal{I} ICO(\mathcal{B}, \phi)$. $ICO_\mathcal{I}(\mathcal{B})$ is the conservative approximation of $ICO(\mathcal{B})$. In order to calculate the $\mathcal{ICS}$, only a finite subset $\mathcal{I}$ of the whole set of possible control inputs Φ is considered (Property 4). The subset $\mathcal{I}$ selected contains the control inputs ϕ of arbitrary duration with constant steering angle u^ξ and constant linear acceleration u^v. First, $\mathcal{I}$ is split into two subsets $\mathcal{I}_S$ and $\mathcal{I}_T$ corresponding respectively to control inputs for which $\mathcal{A}$ is moving straight, *i.e.* $u^\xi = 0$, and control inputs for which $\mathcal{A}$ is turning, *i.e.* $u^\xi \neq 0$. Then, the set of control inputs $\mathcal{I}_T^\xi$ is introduced. It is the set of control inputs for which $\mathcal{A}$ is turning with the steering angle u^ξ.

4.1.3 ICO calculation for a point obstacle moving in a straight line

Let $\mathcal{B}$ be a moving point obstacle. Recall that $\mathcal{B}$(t) gives the position of $\mathcal{B}$ at time t. In order to characterize $ICO(\mathcal{B})$, we consider $\mathcal{B}$ as the union $\cup_t \mathcal{B}(t)$. We proceed to calculate $ICO(\mathcal{B}, \phi) = \cup_t ICO(\mathcal{B}(t), \phi)$. Now, according to Definition 2 in Section 2, $ICO(\mathcal{B}(t), \phi)$ is the set of states s such that if $\mathcal{A}$ starts from s (at time 0) and is subject to the control input ϕ it reaches $\mathcal{B}$(t) (at time t). Such a state s belongs to $R^{-1}(\mathcal{B}(t), \phi)$ and is actually the unique solution of the equation $\phi(s, t) = \mathcal{B}(t) \Leftrightarrow s = \phi^{-1}(\mathcal{B}(T), t)$. Hence, in conclusion $ICO(\mathcal{B}, \phi) = \cup_t \phi^{-1}(\mathcal{B}(t), t)$

Computing $ICO_{I_T^\xi}(\mathcal{B})$ ($\mathcal{A}$ is turning)

Now, to calculate the $\mathcal{ICS}$ for a u^ξ analytically, we need to find all states in the (θ, v) slice from where $\exists t \mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset$. This is nothing other than solving the equations 4.3 and 4.4 for various values of t. Solving for $x(0)$ and $y(0)$ we obtain the result illustrated in Fig.4.2.

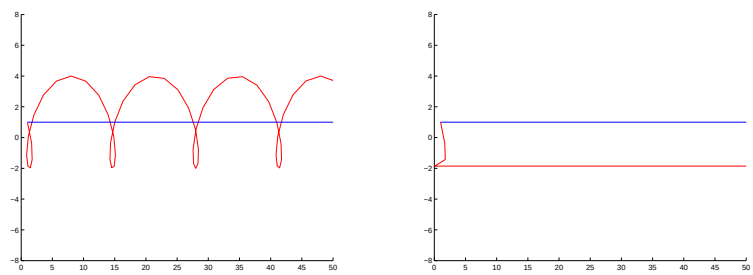

Figure 4.2: Left: $ICO_{\mathcal{I}_T^\xi}(\mathcal{B})$ when $\mathcal{A}$ is turning with constant u^ξ and zero u^v. The line across the curve is the trajectory of the obstacle. $\mathcal{A}$'s velocity $v_{\mathcal{A}}$ is assumed to be two times that of the obstacle's velocity $v_{\mathcal{B}}$. Right: The same situation except, here $\mathcal{A}$ is turning with constant u^ξ and decelerating with a braking distance $d(v)$

The curve on the left is a cycloid. On the right is $ICO_{\mathcal{I}_T^\xi}(\mathcal{B})$. When $\mathcal{A}$ is turning and decelerating it collides with $\mathcal{B}$ *iff* it is on a collision course and its distance to $\mathcal{B}_i$ is less than the braking distance $d(v)$. The straight line part is due to the fact that, when $v_{\mathcal{A}}$ is zero, it stops on the path of the obstacle. This control input is also known as the braking trajectory.

Generalizing for all steering commands

We repeat the same process for the extreme steering angles (maximum positive and maximum negative angles). Now, the analysis is complete except that it lacks the ICO(B) characterization for straight motion of $\mathcal{A}$ ($u^\xi = 0$), which is the topic of the next section.

Computing $ICO_{\mathcal{I}_s}(\mathcal{B})$ ($\mathcal{A}$ is moving straight)

Again, we have to parameterize the trajectory of $\mathcal{A}$ by time t. Generalization to other slices is direct. The main issues we have to take into consideration are cases in which $v_A(t) \leq 0$, $0 < v_A(t) < v_{A_{Max}}$,$v_A(t) \geq v_{A_{Max}}$ according to whether $\mathcal{A}$ is decelerating or accelerating. $v_{A_{Max}}$ here is $\mathcal{A}$'s maximum velocity and $v_A(t) = v_i + u^v_{max}t$, gives the velocity at time t. Like in the case of steering angles, here too, we consider trajectories with maximum acceleration and deceleration only.
In the decelerating case, as we consider forward moving car-like robot, we don't allow $v_A(t) < 0$. Hence, as long as $v_A(t) > 0$,

$$y(t) = y(0) - v_0 t + \frac{1}{2}u^v_{max}t^2 \tag{4.5}$$

where, v_0 is the (v,θ) slice we are considering and u^v_{max} is absolute value of the maximum deceleration. But if $v_{\mathcal{A}}$(t) = 0

$$y(t) = y(0) + \frac{{v_0}^2}{2u^v_{max}} \tag{4.6}$$

In the accelerating case, we shouldn't allow $v_A(t) > v_{A_{Max}}$. Hence, as long as $v_A(t) < v_{A_{Max}}$

$$y(t) = y(0) - vt - \frac{1}{2}u^v_{max}t^2 \tag{4.7}$$

But if $v_A(t) = v_{A_{Max}}$, then

$$y(t) = y(0) - v_{A_{Max}}t \tag{4.8}$$

Using these equations, we get an ICO(B) characterization as shown in Fig.4.3 for the straightline motion of $\mathcal{A}$.

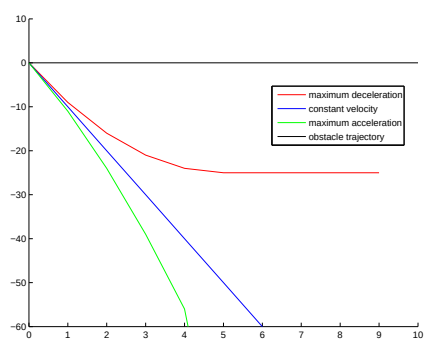

Figure 4.3: $ICO_{\mathcal{I}_S}(\mathcal{B})$ when $\mathcal{A}$ is moving straight (u^ξ=0) with maximum acceleration, u^v_{max}, constant speed, u^v=0, and maximum deceleration, -u^v_{max}. The straight line y=0, is the obstacle's trajectory.

Computing $ICO_{\mathcal{I}}(\mathcal{B})$

The two previous sections have characterized the inevitable collision obstacles for different subsets of $\mathcal{I}$, the whole set of control inputs considered. The final characterization of the inevitable collision obstacles is determined using:

$$\begin{aligned}
ICO_I(\mathcal{B}) &= \bigcap_{\chi \in \{I_s, I_T^\zeta\}} ICO_\chi(\mathcal{B}) \\
&= \bigcap_{\phi \in I_s} ICO(\bigcup_t B(t), \phi) \cap \bigcap_{\psi \in I_T^\zeta} ICO(\bigcup_t B(t), \psi) \\
&= \bigcap_{\phi \in I_s} \bigcup_t ICO(\mathcal{B}(t), \phi) \cap \bigcap_{\psi \in I_T^\zeta} \bigcup_t ICO(\mathcal{B}(t), \psi) \\
&= \bigcap_{\phi \in I_s} \bigcup_t \phi^{-1}(\mathcal{B}(t), t) \cap \bigcap_{\psi \in I_T^\zeta} \bigcup_t \psi^{-1}(\mathcal{B}(t), t)
\end{aligned}$$

Now, for the point obstacle case, it is obvious that the only $\mathcal{ICS}$ is $\mathcal{B}(0)$ itself.

4.1.4 ICO calculation for a solid obstacle moving in a straight line

Let us now assume that $\mathcal{B}$ is a moving solid obstacle. $\mathcal{B}$ is the union of a set of moving point obstacles and we can write: $\mathcal{B} = \cup_i \cup_t \mathcal{B}_i(t)$. $ICO(\mathcal{B})$ is derived as before.

$$\begin{aligned}
ICO_I(\mathcal{B}) &= \bigcap_{\chi \in \{I_s, I_T^\zeta\}} ICO_\chi(\mathcal{B}) \\
&= \bigcap_{\phi \in I_s} ICO(\bigcup_i \bigcup_t B_i(t), \phi) \cap \bigcap_{\psi \in I_T^\zeta} ICO(\bigcup_i \bigcup_t B_i(t), \psi) \\
&= \bigcap_{\phi \in I_s} \bigcup_i \bigcup_t ICO(\mathcal{B}_i(t), \phi) \cap \bigcap_{\psi \in I_T^\zeta} \bigcup_i \bigcup_t ICO(\mathcal{B}_i(t), \psi) \\
&= \bigcap_{\phi \in I_s} \bigcup_i \bigcup_t \phi^{-1}(\mathcal{B}_i(t), t) \cap \bigcap_{\psi \in I_T^\zeta} \bigcup_i \bigcup_t \psi^{-1}(\mathcal{B}_i(t), t)
\end{aligned}$$

As seen above, it is the union of $ICO_I(\mathcal{B}_i)$ for every point $\mathcal{B}_i$ of $\mathcal{B}$. It is therefore the convolution between $\mathcal{B}$ and the $ICO_I(\mathcal{B})$ obtained in the previous case. More precisely, it is the Minkowski sum between them. This is illustrated in Fig.4.4 below.

4.1.5 ICO calculation, for a stationary point and solid obstacle

This topic has been extensively studied in [12]. Hence it is not dealt with here. But, infact, the stationary obstacle case is similar to the dynamic obstacle case with zero velocity and $\forall t, \mathcal{B}(t) = \mathcal{B}(0)$. Hence, the calculation is exactly as shown for the dynamic case.

Now, when calculating $ICO(\mathcal{B})$ for two or more obstacles, it is important to note that the obstacle union is nested within the control input intersection in Property 2.

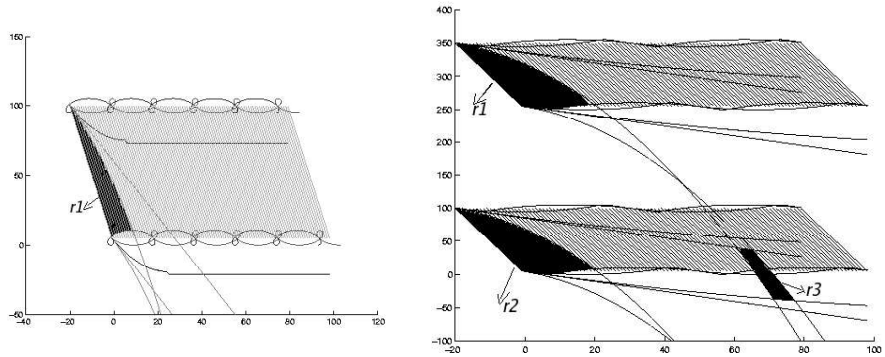

Figure 4.4: Left: $ICO_I(\mathcal{B})$ for $\mathcal{A}$, taking into account both $\mathcal{I}_S$ and $\mathcal{I}_T$. So, whenever $\mathcal{A}$ finds itself in the highlighted region at $t = 0$, it will eventually collide with $\mathcal{B}$. Right: $ICO_I(\mathcal{B})$ for $\mathcal{A}$ in the case of two obstacles. Notice regions r_1, r_2 and r_3. From r_1, all control inputs lead to collision with $\mathcal{B}_1$, similarly from r_2 with $\mathcal{B}_2$ but from r_3, some control inputs lead to a collision with $\mathcal{B}_1$ and others with $\mathcal{B}_2$.

Theoretically, this property is complete for computing the $ICO(\mathcal{B})$, but in some cases, as the one shown in Fig.4.4b, we find regions of $\mathcal{ICS}$, (for example r_1 r_2 and r_3 in this case). Among these regions, r_3 is created due to the intersection of infinite strips of $ICO(\mathcal{B})$ for different control inputs leading to collision with different obstacles.

4.2 On the Conservative Approximation

In fact, every point in the set of $\mathcal{ICS}$ has it's own time-to-collision and most often, $\mathcal{ICS}$ that lie far away from the obstacles are mainly due to collision after a long time. States in these regions (*e.g.* r_3), although correctly characterized as $\mathcal{ICS}$ *for the considered set of control inputs*, $\mathcal{I}$, can be made to vanish by considering a larger set $\mathcal{I}'$, (shown later). On the other hand, regions r_1 and r_2 can only be reduced (by considering many control inputs) but not made to vanish unless we assume trajectories uncharacteristic to $\mathcal{A}$.

This issue could be easily fixed by incorporating a time-horizon, but the objective of this project is precisely to avoid the notion of time-horizon as it renders the system unsafe. How else could we get rid of these regions which are too conservative an approximation? First, we begin by defining conservativeness which will help us formalize this issue.

Def. 4 *A set of control inputs $\mathcal{I}$ is considered to be more conservative than a set $\mathcal{I}'$* iff $\mathcal{VO}_I > \mathcal{VO}_{I'}$, *where $\mathcal{VO}$ is the volume of the $\mathcal{ICS}$ region calculated using these control inputs.*

It is to be noted that the $\mathcal{VO}$ not only depends on the cardinality of set $\mathcal{I}$ but also the quality of the control input in evading the obstacles. The fact that $\mathcal{VO}$ is minimum ($\mathcal{VO}_{min}$) in the case where we consider the set of all possible control inputs, (practically infinite) is evident. We call this volume to be conservative of order zero, or exact $\mathcal{ICS}$. Hence the lower the order, the smaller the volume (closer to the exact $\mathcal{ICS}$). In the same lines, we also define the *quality* of the approximation.

Def. 5 *The quality of the approximation of a set of control inputs $\mathcal{I}$ is given as the ratio $\mathcal{VO}_{min}/\mathcal{VO}_{\mathcal{I}}$. A quality value of* 0 *means the approximation is too conservative, while a quality of* 1 *is the best one can achieve.*

Now, getting back to the question, *is it possible to completely get rid of $\mathcal{ICS}$ that are far away from the obstacles, only by considering a fixed number of control inputs?*. The answer is no, as will be seen below.

Property 5 *If the number of control inputs considered for the robotic system is fixed, the quality reduces with increase in the number of obstacles.*

Proof: Assume $\mathcal{I} = \{\mathcal{I}_1,\mathcal{I}_2\}$ be the set of all control inputs available to the robotic system and $\forall\phi \in \mathcal{I}_1, \exists t, \phi(\mathbf{s_0}, t) \in \mathcal{B}_1(t)$ and $\forall\psi \in \mathcal{I}_2, \exists t, \psi(\mathbf{s_0}, t) \in \mathcal{B}_2(t)$, or in other words, state $\mathbf{s_0}$ is an $\mathcal{ICS}$. ■

Now, let $\mathcal{G} = \bigcap_{I_1} ICO(\mathcal{B}_1(t), \phi) \cap \bigcap_{I_2} ICO(\mathcal{B}_2(t), \psi)$. $\mathcal{G}$ defines the set of states from where one set of control inputs lead to a collision with one obstacle and another set of control inputs lead to a collision with another obstacle. Now, to avoid this region, we can add another trajectory ϕ_{new} to $\mathcal{I}$ in such a way that $\mathcal{G} \cap ICO(\mathcal{B}_1(t), \phi_{new})$ and $\mathcal{G} \cap ICO(\mathcal{B}_2(t), \phi_{new}) = \emptyset$.In other words, this seems to provide a solution that the addition of the new trajectory avoids the region $\mathcal{G}$. But, now let us fix the number of control inputs, and start adding obstacles to $\mathcal{W}$. Theoretically it is easy to carefully place an obstacle, to make $\mathcal{G} \cap ICO(\mathcal{B}_3(t), \phi_{new}) \neq \emptyset$

Thus as we keep on adding obstacles to $\mathcal{W}$, the order of conservativeness increases *i.e.* the $\mathcal{ICS}$ volume increases. The main reason for this behavior is the presence of infinite strips of ICO's. Hence, there is a need to have a control input that produces only a finite $ICO(\mathcal{B})$ region. To this we introduce the idea of *imitating manoeuvres*.

4.3 Imitating Manoeuvres

Imitating manoeuvres are control inputs in which the robot tries to imitate the obstacle's trajectory. They consist of two parts:

- The "catching-up" part, during which the robot from an arbitrary state tries to achieve a zero relative velocity with the obstacle.
- The "following" part, during which the robot duplicates the obstacle's control inputs.

The interest in these imitating manoeuvres is that, the $ICO_($B) for these control inputs tend to cloud around a fixed radius of the obstacle (under the condition that the robotic system is physically capable of duplicating the obstacle's trajectory).

The Fig.4.5 illustrates this idea. $\mathcal{A}$ is the robotic system and $\mathcal{B}$ is the obstacle. They have the same velocity and orientation (zero relative velocity). Hence in this case, there is no need for a catching-up manoeuvre. If from this time instant, $\mathcal{A}$ applies the same control inputs as $\mathcal{B}$ (the following manoeuvre), they will have a collision *iff* $\mathcal{A}(0)$ and $\mathcal{B}(0)$ intersect. In other words, $ICO(\mathcal{B})$ region is finite and limited only to $\mathcal{B}(0)$. On the other hand, $\mathcal{A}$' and $\mathcal{B}$ have a non-zero relative velocity. To have a finite $ICO(\mathcal{B})$ region,

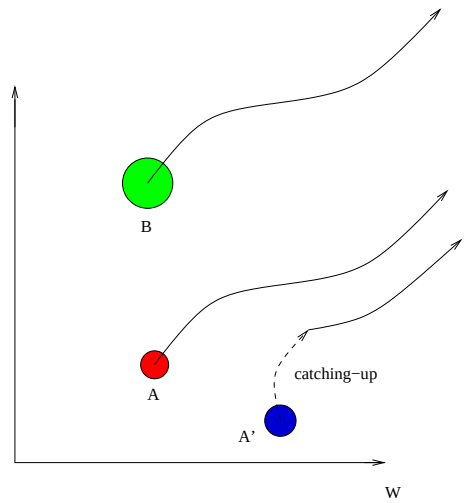

Figure 4.5: $\mathcal{A}$ and $\mathcal{B}$ have the same velocity and heading angle. Heading angle of $\mathcal{A}$' is different. The catching-up manoeuvre consists of orienting $\mathcal{A}$' with $\mathcal{B}$ in the shortest possible time. $\mathcal{W}$ denotes the workspace.

a catching-up manoeuvre is necessary (shown as the dotted line). Once the catching-up manoeuvre is successfully accomplished, the following manoeuvre is executed.

This is analytically proved below.

Property 6 *A catching-up manoeuvre that results in a zero relative velocity between the robotic system $\mathcal{A}$ and the obstacle $\mathcal{B}$ gives rise to a finite $ICO(\mathcal{B})$ region.*

Let t_c denote the catching-up time, ϕ_c denote the control input corresponding to the catching-up manoeuvre and ϕ_f denote the control input corresponding to the following manoeuvre (ϕ_f is same as the obstacle's control inputs).

Proof:

$$\begin{aligned} ICO(B) &= \bigcup_t ICO(\mathcal{B}(t)) \\ &= \bigcup_{0\le t\le t_c} ICO(\mathcal{B}(t),\phi_c) \cup \bigcup_{t\ge t_c} ICO(\mathcal{B}(t),\phi_f) \\ &= \bigcup_{0\le t\le t_c} {\phi_c}^{-1}(\mathcal{B}(t),t) \cup {\phi_c}^{-1}(\bigcup_{t\ge t_c} {\phi_f}^{-1}(\mathcal{B}(t),t)) \\ &= \bigcup_{0\le t\le t_c} {\phi_c}^{-1}(\mathcal{B}(t),t) \end{aligned} \tag{4.9}$$

■

It is clear that the $ICO(\mathcal{B})$ is the union of the $ICO(\mathcal{B})$ due to the catching-up and the following manoeuvre. When the robotic system has the same control inputs as that of the obstacle, the following manoeuvre's $ICO(\mathcal{B})$ reduces to ${\phi_c}^{-1}(\mathcal{B}(t_c),t_c)$ (because for two objects having a zero relative velocity a collision can take place only if they start at the same point on the workspace. Thus the $ICO(\mathcal{B})$ is reduced to a finite region represented by the first term based on the catching-up manoeuvre alone.

4.3.1 Catching-up manoeuvre

The key to have a good quality $\mathcal{ICS}$ region is to avoid infinite strips of $ICO(\mathcal{B})$. This is accomplished by the imitating manoeuvre. In fact, the following inferences can be made from Property 6:

- If the robot is capable of reproducing the obstacles behavior, a combination of a catching-up and following manoeuvres can improve the quality of $\mathcal{ICS}$ approximation.
- The catching-up part is different for different slices of the (v,θ) plane and hence it is of prime importance to characterize it in terms of v and θ values.
- And, the shorter the time taken by the catching-up manoeuvre is, the smaller the ICO region.

With these points, we can go ahead to define the best possible manoeuvre for the robot to execute the catching-up manoeuvre in the shortest possible time.

4.4 Catching-up manoeuvres for the car-like vehicle

4.4.1 ICO calculation for an obstacle moving in a straight line

Consider a (v_0,θ_0) slice in the (v,θ) space. The equation that relates the change in θ value w.r.t time is given as,

$$\theta(t) = \theta_0 + \frac{v_0 t \sin(u^\xi)}{b} \tag{4.10}$$

Consider an obstacle $\mathcal{B}$ moving in a straightline with an orientation θ_{obs} and velocity v_{obs}. The distance to be travelled by $\mathcal{A}$ to reach the orientation is fixed at r(θ_{obs}-θ_0), the length of the arc L_{arc}, where $r = \frac{b}{\tan(u^\xi)}$. According to the previous section, the idea of the catching-up manoeuvre is to make the relative velocity between $\mathcal{A}$ and $\mathcal{B}$ zero as soon as possible. This makes it neccessary to accelerate or decelerate according to whether v_0 is lesser or greater than v_{obs} respectively. But what exactly is the optimal way to cover the distance and satisfy the velocity constraint at the same time ?

4.4.2 Optimal Control

According to the Pontryagin Maximum Principle [18], optimal controls are typically hard limited and piecewise continuous. That is to say that, they are hard against the constraint boundaries and switch abruptly between limits at certain critical points in the time axis. These optimal controls are also called Bang-Bang controls for the same reason.

A simple example of bang-bang control is to cover a fixed distance starting from rest state and stopping at the final state in the shortest possible time. The solution for this problem is to accelerate as hard as possible to reach maximum speed (Maximum Acceleration Phase), travel as fast as possible for as long as we can (Maximum Velocity Phase) just in time to brake as hard as possible to stop at the destination (Maximum Deceleration Phase), all assuming that the distance is large enough to allow all three

phases to be executed. The bangs divide the problem into sub-tasks and the optimal solution is locally optimal in the sense that at each instant we should do as well as we can for the current subtask to achieve a globally optimal solution.

4.4.3 Optimal Catching-up Manoeuvre

In our case, the catching-up manoeuvre can hence be characterized in three phases, phase I to reach $v_{A_{Max}}$ from $v_{\mathcal{A}}$ (not from 0 like in the example), phase II at the maximum velocity and phase III to reach v_{obs} (not 0 again).

1. Phase I: Time needed to accelerate to $v_{\mathcal{A}_{Max}}$

$$t_1 = \frac{v_{\mathcal{A}_{Max}} - v_0}{u^v_{Max}} \tag{4.11}$$

2. Phase III: Time needed to decelerate to $v_{\mathcal{B}}$

$$t_3 = \frac{v_{\mathcal{B}} - v_{\mathcal{A}_{Max}}}{-u^v_{Max}} \tag{4.12}$$

3. Distance traveled in Phase I

$$d_1 = v_0 t_1 + \frac{1}{2} u^v_{Max} t_1^2 \tag{4.13}$$

4. Distance traveled in Phase III

$$d_3 = v_{\mathcal{A}_{Max}} t_3 - \frac{1}{2} u^v_{Max} t_3^2 \tag{4.14}$$

5. Time needed to reach the $v_{\mathcal{B}}$ from v_0

$$t_{\mathcal{AB}} = \frac{|v_0 - v_{\mathcal{B}}|}{u^v_{Max}} \tag{4.15}$$

6. Distance traveled in time $t_{0\mathcal{B}}$

$$d_{0\mathcal{B}} = v_0 t_{0\mathcal{B}} \pm \frac{1}{2} u^v_{Max} t_{0\mathcal{B}}^2 \tag{4.16}$$

   $\pm$ according to whether v_0 is greater or lesser than $v_{\mathcal{B}}$

Now, with these constraint equations, we have the following equations to verify the feasibility of executing the three phases.

1. $d_1 + d_3 < L_{arc}$
   In this case, $\mathcal{A}$ should execute Phase II for $t_2 = \frac{L_{arc} - (d_1 + d_3)}{v_{\mathcal{A}_{Max}}}$ seconds. This is illustrated in Fig.4.6

2. $d_1 + d_3 = L_{arc}$
   In this case, $\mathcal{A}$ cannot execute Phase II, and hence $t_2 = 0$, but $\mathcal{A}$ will reach the required velocity and orientation.

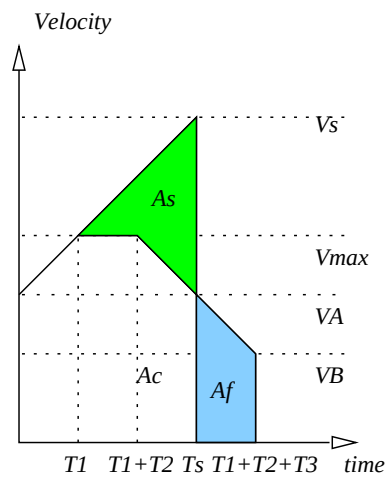

Figure 4.6: Velocity-Time graph of $\mathcal{A}$. ($v_{\mathcal{B}}$ is the velocity of the obstacle (to be achieved at the end of the catching up manoeuvre),v_0 is the current velocity of the $\mathcal{A}$, (v_s, T_s) the velocity of $\mathcal{A}$ and the best time to cover the distance (maximum acceleration all the way). The time taken by the catching up manoeuvre is $T_f = t_1 + t_2 + t_3$, where t_1 is the time taken to reach maximum velocity, t_2 is the time traveled at maximum velocity and t_3 is the time taken to decelerate to $v_{\mathcal{B}}$ from the maximum velocity. Although the best time T_s is shorter than T_f, it is to be noted that the speed V_s ($> v_{\mathcal{A}_{Max}}$) is not attainable by $\mathcal{A}$. To travel an equivalent distance of L_{arc}, the area A_f should be equal to A_s. Area A_c denotes the common area. Obviously, $A_s + A_c = L_{arc}$

3. $d_1 + d_3 > L_{arc}$
   In this case, $\mathcal{A}$ will actually travel a distance of $d_1 + d_3 - L_{arc}$ in the direction of the motion of the obstacle if we want to execute both phases I and III to completion. But the optimal solution is to accelerate for a time t_1' ($< t_1$) and then decelerate to $v_{\mathcal{B}}$ in time to exactly cover the distance. This is a special case and we will need to work out the following possibilities.

   - For the sub-case where $d_{0\mathcal{B}} < L_{arc}$, the optimal solution is as explained above. Two cases exist, one when $v_0 < v_{\mathcal{B}}$ and the other when $v_0 > v_{\mathcal{B}}$. It is necessary to find t_i that gives an area equal to the distacne to be traveled. Fig.4.7a. illustrates this case
   - On the other hand, the case where $d_{0\mathcal{B}} > L_{arc}$ occurs when the distance traveled on applying the control action (maximum acceleration or deceleration) to reach the $v_{\mathcal{B}}$ is already greater than the length of the arc. In this case it is best (the only option) to provide control action to reach $v_{\mathcal{B}}$ and a distance of $d_{0\mathcal{B}} - L_{arc}$ is traveled along the trajectory of the obstacle. This is explained in Fig.4.7b.

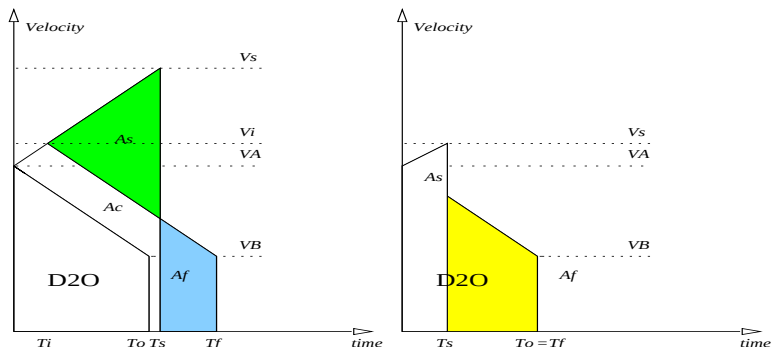

Figure 4.7: Velocity-Time graph of $\mathcal{A}$. a) $(v_{\mathcal{B}}, T_o)$ are the velocity of the obstacle (to be achieved at the end of the catching up part) and the time taken to attain this velocity (directly) resp. (v_s, T_s) are the velocity of $\mathcal{A}$ and the best time to cover the distance (maximum acceleration all the way) resp. and (v_i, T_i) the maximum velocity which also satisfies property 1, and time to reach this velocity. The time taken by this catching up manoeuvre is T_f. To travel an equivalent distance of L_{arc}, the area A_f should be equal to A_s. Area A_c denotes the common area. Obviously, $A_s + A_c = L_{arc}$. b) Here $d_{0\mathcal{B}} > As + Ac = L_{arc}$, hence $T_f = T_o$

Finally, we have to calculate the ICO for the imitating manoeuvre for every obstacle $\mathcal{B}_i$ in the scene. An example of the imitating manoeuvre for an obstacle moving in a straight line is shown below in Fig.4.8. Notice the finite $ICO(\mathcal{B})$ region.

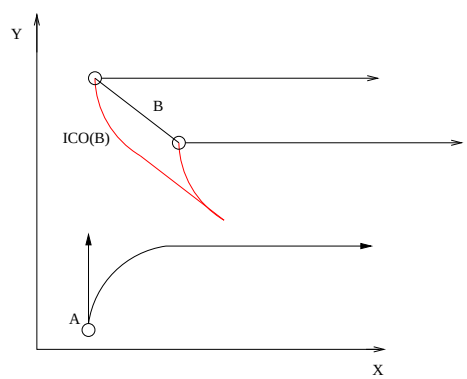

Figure 4.8: Obstacle $\mathcal{B}$ is moving in a straight line towards the right. $\mathcal{A}$ has an heading angle of θ=90. The catching-up manoeuvre consists of reaching a state where θ is 0 and v is $v_{\mathcal{B}}$ in the shortest time possible. The following manoeuvre consists of applying zero linear acceleration. The finite region of $ICO(\mathcal{B})$ is the set of states from where the catching-up trajectory would lead to a collision.

In conclusion, this Chapter has studied the characterization of $\mathcal{ICS}$ for a car-like vehicle and proposed the concept of imitating manoeuvres that are critical for the quality of the approximation of the $\mathcal{ICS}$ set. The next Chapter describes an algorithm for the software implementation of this case-study.

Chapter 5

Algorithm and Software Implementation

This Chapter presents an algorithm to realize the $\mathcal{ICS}$ characterization for a car-like vehicle discussed in Chapter 4. The proposed algorithm is then implemented in software. A complexity analysis shows that collision checking can be performed in polynomial time.

5.1 Algorithm for $\mathcal{ICS}$-*Checker*

The algorithm takes the form of a predicate $\texttt{ICS}(\mathbf{s_i})$ that is true *iff* $\mathbf{s_i}$ is an $\mathcal{ICS}$.

5.1.1 Input

- $\mathbf{s_i}$, the state that is to be checked for collision. The values passed are x_i,y_i,θ_i and v_i. Here (x_i,y_i) is the workspace position of $\mathcal{A}$ and (θ_i,v_i) are the heading angle and velocity of $\mathcal{A}$ respectively. (They also correspond to an unique slice in the 4-dimensional state space).
- A set $\mathcal{B}$ of N_s static obstacles and N_d dynamic obstacles, the positions in the case of static obstacles and the predicted future behavior (velocity and heading angle) in the case of dynamic obstacles.

5.1.2 Output

A boolean value saying whether $\mathbf{s_i}$ is an $\mathcal{ICS}$.

5.1.3 Algorithm

By definition, whenever $\mathcal{A}$ is inside the region $ICO(\mathcal{B})$ *at time* 0, no matter what action it takes, it eventually collides with $\mathcal{B}$. Hence, when the $\mathcal{ICS}$ *checker* (collision checker) is provided with a state $\mathbf{s_i}$ (which will be reached by $\mathcal{A}$ at time t'), the time axis is translated to commence from t' as 0 and the obstacle's positions at $t = t'$ are assumed to be their positions at 0 and the $ICO(\mathcal{B})$ is calculated. And, if $\mathbf{s_i}$ is inside this calculated region $ICO(\mathcal{B})$ *at time* 0 (t'), we return a boolean value *true* to signify that $\mathbf{s_i}$ is infact an $\mathcal{ICS}$.

After adding imitating manoeuvres, the ICO characterization in Section 4.1.4 is altered as,

$$ICO_{\mathcal{I}}(\mathcal{B}) = \bigcap_{\chi \in \{\mathcal{I}_f, \mathcal{I}_{\mathcal{T}}{}^{\zeta}, \mathcal{I}_{\mathcal{E}}\}} ICO_{\chi}(\mathcal{B}) \tag{5.1}$$

$$= \bigcap_{\phi \in \mathcal{I}_f} ICO(\bigcup_i \bigcup_t \mathcal{B}_i(t), \phi) \cap \bigcap_{\psi \in \mathcal{I}_{\mathcal{T}}{}^{\zeta}} ICO(\bigcup_i \bigcup_t \mathcal{B}_i(t), \psi) \cap \bigcap_{\mu \in \mathcal{I}_{\mathcal{E}}} ICO(\bigcup_i \bigcup_t \mathcal{B}_i(t), \mu)$$

$$= \bigcap_{\phi \in \mathcal{I}_f} \bigcup_i \bigcup_t ICO(\mathcal{B}_i(t), \phi) \cap \bigcap_{\psi \in \mathcal{I}_{\mathcal{T}}{}^{\zeta}} \bigcup_i \bigcup_t ICO(\mathcal{B}_i(t), \psi) \cap \bigcap_{\mu \in \mathcal{I}_{\mathcal{E}}} \bigcup_i \bigcup_t ICO(\mathcal{B}_i(t), \mu)$$

$$= \bigcap_{\phi \in \mathcal{I}_f} \bigcup_i \bigcup_t \phi^{-1}(\mathcal{B}_i(t), t) \cap \bigcap_{\psi \in \mathcal{I}_{\mathcal{T}}{}^{\zeta}} \bigcup_i \bigcup_t \psi^{-1}(\mathcal{B}_i(t), t) \cap \bigcap_{\mu \in \mathcal{I}_{\mathcal{E}}} \bigcup_i \bigcup_t \mu^{-1}(\mathcal{B}_i(t), t)$$

where, $\mathcal{I}$ is the complete set of control inputs considered, $\mathcal{I}_s$ is the sub-set of control inputs where $\mathcal{A}$ moves in a straight line, $\mathcal{I}_T^{\zeta}$ is the sub-set of control inputs where $\mathcal{A}$ moves in with a non-zero steering angle, and $\mathcal{I}_E$ is the sub-set of imitating manoeuvres. The cardinality of each sub-set is explained in the next section.

Steps involved in calculating the $\mathcal{ICS}$ for a given (θ, v) slice

1. *Minkowski Sum calculation:* The obstacles are grown *w.r.t* the θ slice of the vehicle.
2. *ICO calculation for each obstacle:* Calculate $ICO(\mathcal{B},\phi)$ for any one point on each obstacle. The control inputs considered for each obstacle are,
   - Braking trajectories with steering angles $\{-u^{\xi}_{max}, 0, u^{\xi}_{max}\}$. (3 control inputs)
   - Trajectory with maximum acceleration u^{v}_{max}.(this considerably reduces the ICO for a dynamic obstacle).
   - Imitating manoeuvre of each dynamic obstacle. (N_d control inputs)
3. *Minkowski Sum calculation:* The obstacles are further grown by calculating the minkowski sum between the ICO generated by the control inputs for the point considered and the grown obstacle in step 1.
4. In case of more than one obstacle, calculate $ICO(\mathcal{B}, \phi_j) = \bigcup_{i=1}^{N} \bigcup_t ICO(\mathcal{B}_i(t), \phi_j)$, where $N = N_d + N_s$ is the number of obstacles, for $\forall j = 1 \ldots M$ control inputs where $M = N_d + 4$, the total number of control inputs.
5. Calculate $\bigcap_{\phi_j=1}^{M} ICO(\mathcal{B}, \phi_j)$ to find the $ICO(\mathcal{B})$ or the $\mathcal{ICS}$ for the slice
6. *Point Location:* Check if the workspace position (x, y) is inside the $\mathcal{ICS}$ region

5.1.4 Complexity of the Algorithm

1. Step 1 of the algorithm: Let n_B and n_A be the number of vertices of the obstacles (all obstacles together) and the vehicle respectively, the minkowski sum then has a complexity of $O(n_B + n_A)$ if both polygons are convex, $O(n_B n_A)$ if one of them is

convex and the other non-convex and $O(n_B^2 N_A^2)$, if both of them are non-convex. In our implementation, although we have a choice of considering vehicles of any type, in the general case a rectangle representation of the vehicle is chosen. Hence the complexity is $O(4n_B)$ or $O(n_B)$.

2. Step 2 of the algorithm: While calculating the ICO for any point, we use arcs (with known radius and arc length) and straight lines as a part of the trajectory. Hence, we add 1 vertex to denote the end of the arc (the arc begins at the point on the obstacle) and another to denote the end of the straight line (the line begins at the end of the arc). Hence for each point on the obstacle, we add a maximum of $O(M)$ points for M control inputs.

3. Step 3 of the algorithm: In this step, the minkowski sum between the grown obstacles in step 1 and the trajectories in step 2 is performed. The result of step 1 is generally not convex, neither is the result of step 2. Hence the complexity is $O(M^2 n_B^2)$

4. Step 4 of the algorithm: Union and intersection operations can be computed in $O(nlogn + klogn)$, where n is total number vertices of the two polygons and k is the complexity of the output. In our case, $M(N-1)$ union operations will have to be performed. In the worst case, there are n_B vertices for each union operation. Hence the complexity is $O(n_B logn_B + klogn_B)$ for each union operation and step 4 has a complexity of $O(MN(n_B logn_B + klogn_B))$. A constraint on N, the number of obstacles is that, $N \leq n_B/3$ as all polygonal obstacles have atleast 3 vertices. Hence the complexity of calculating the union is $O(Mn_B(n_B logn_B + klogn_B))$

5. Step 5 has the same complexity as step 4.

6. Step 6, the point location query has a complexity $O(logn)$, where n is the number of vertices. In our case the maximum number of vertices is $O(M^2 n_B^2)$

5.2 Software

The above mentioned algorithm has been implemented in software, typical environments have been defined, for example, highway environment with lanes, and a maze environment for a cycle, that could be considered as the piano-mover problem's equivalent in state space, are proposed.

5.2.1 Programming

The programming is done using C++ and the graphics interface was designed using FLTK [19]. Commercially available software like LEDA [20] *etc*, have built-in functions to perform the steps highlighted in the algorithm. In this project no commercial software was used. The source codes for the Minkowski sum, Planar point localization *etc*, were adapted from [21] and changed to suit the application.

Environment	Number of Static obstacles	Number of Dynamic obstacles	Total number of vertices	Execution time (in ms)
Highway	8	4	48	≈ 9
	8	8	64	≈ 17.5
	8	16	96	≈ 25
Maze	2	0	8	≈ 1
	4	0	16	≈ 1.5
	8	0	32	≈ 2

Table 5.1: Execution time

5.2.2 Experimental Results

Maze Environment

The Maze environment, consists of two rectangular stationary obstacles. The robotic system considered is a bicycle. The environment depicts the scenario where the two obstacles act as a barricade for the cyclist and the navigation problem is to find if a given trajectory is feasible *i.e.* to verify that none of the states is an $\mathcal{ICS}$. The maze environment is depicted in the Fig.5.1. This problem is the state space equivalent of the classical piano mover's problem in the configuration space. The goal there is to identify if there exists a "path" between two configurations.

Figure 5.1: Maze environment

Highway Environment

The highway model, depicts a scenario with both stationary obstacles (limits of the roadway) and moving obstacles (other vehicles). It is assumed that the moving obstacles are also car-like vehicles and that they obey the highway code and therefore follow the environment lanes at prescribed speeds. Here the maximum allowed speed is 55 m/s ($\approx 200 km/h$). Fig.5.2 shows the output at different stages of execution of the algorithm.

Execution Time

The time of execution for a single query *w.r.t* number of obstacles was studied. The results are presented in Table 5.1

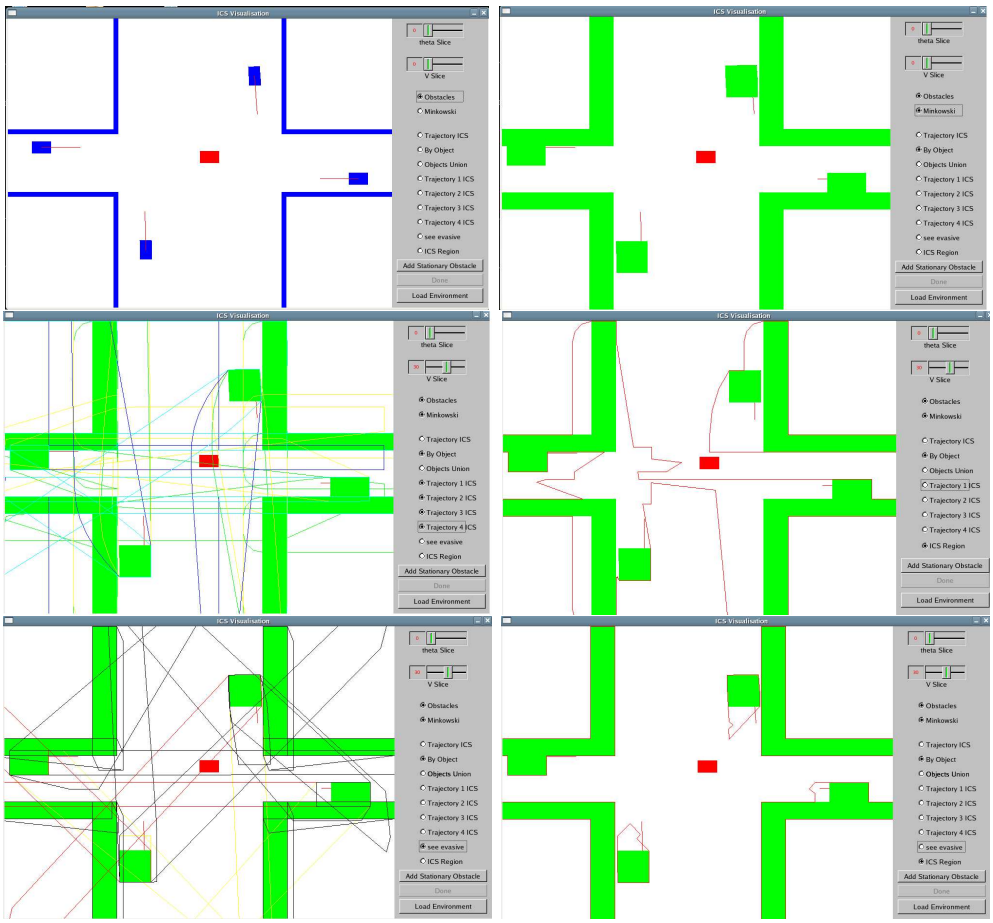

Figure 5.2: a) The environment for which the set of $\mathcal{ICS}$ is calculated. b) The result of the minkowski sum of the robotic system with obstacles. Note that we are in the $(30, 0)$ slice. c) The $ICO(\mathcal{B},\phi)$ for each control input. d) $\mathcal{ICS}$ region calculated using these control inputs. e) $ICO(\mathcal{B},\phi_{im})$ where ϕ_{im} are the imitating manoeuvres. f) $\mathcal{ICS}$ after taking into account the imitating manoeuvres. Notice the difference in the quality of approximation.

Hence, in conclusion a polynomial time algorithm for collision detection in navigation is proposed and it's efficiency is verified by means of a software implementation. The algorithm is fast and assures safety of the robotics system when used with a *k-step* navigation method.

Chapter 6

Conclusion and Future work

A solution to the safety issue concerning the navigation of a robotic system in a highly dynamic environment has been proposed in this work. Initially, the importance of considering the dynamics of the robotic system and the moving obstacles while performing navigation was highlighted. Secondly, the problems posed by incorporating a time-horizon in navigation algorithms was explained. A review of the deliberative, reactive and k-step approaches to navigation were presented. The recent and popular algorithms for reactive navigation were reviewed and their drawbacks with respect to the navigation problem were described.

The concepts of Inevitable Collision States and Obstacles were formalized and various properties relevant to their characterization were presented. The novel idea of imitating manoeuvres were introduced. Thanks to these manoeuvres, the quality of approximation of the $\mathcal{ICS}$ is improved. The imitating manoeuvres are based on the relative velocity between the obstacle and the robotic system. It is argued that if the relative velocity between the obstacle and the robotic system is zero, then they will collide if and only if they start at the same workspace position. Hence, for every other non-zero relative velocity, the imitating manoeuvre consists of a catching-up part when the robotic system tries to attain a zero relative velocity $w.r.t$ to the obstacle.

Then it was proved that the $\mathcal{ICS}$ region corresponding to this manoeuvre depends on the time taken by the catching-up part. Hence, to minimize the $\mathcal{ICS}$ region, a constrained optimization procedure was performed to find the best bang controls that minimize the time taken. The "quality" of the $\mathcal{ICS}$ approximation was defined and it was shown that the imitating manoeuvres help improve the quality of the approximation. An algorithm for the characterization of the $\mathcal{ICS}$ for a car-like vehicle was proposed. The algorithm was implemented in C++. Typical environments that test the robustness and the execution time of the algorithm were defined. A classical problem of verifying the safety of a trajectory of a bicycle in a maze environment, was described.

Further work on the $\mathcal{ICS}$ topic will be carried out on studying the imitating manoeuvres for obstacles moving arbitrarily in an environment. Thoeretical work on determining the quality of a set of control inputs in approximating the $\mathcal{ICS}$ will be performed. Highly dynamic and cluttered environments will be considered to study the dependence of the execution time on the number of static and dynamic obstacles. Complexity of the algorithm is another area where further work has to be performed. It is conjectured that the complexity will depend more on the number of dynamic obstacles as they contribute one

more control input in contrast with the number of static obstacles. Theoretical analysis with experimental results will have to suppliment this conjecture. Lastly, the proposed algorithm will be coupled with the Partial Motion Planner and implemented on the Cycab vehicle.

Bibliography

[1] J.C. Latombe. *Robot motion planning*. Kluwer Academic Publishers, 1991.

[2] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–98, Spring 1986.

[3] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *Robotics and Automation*, 7(3):278–288, June 1991.

[4] I. Ulrich and J. Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1572–1577, May 1998.

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

[6] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics*, pages 233–255, Hanover, March 2000.

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

[8] F. Large, S. Sekhavat, Z. Shiller, and C. Laugier. Towards real-time global motion planning in a dynamic environment using the NLVO concept. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, pages 607–612, Lausanne, September-October 2002.

[9] S. Petti and Th. Fraichard. Safe motion planning in dynamic environments. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Edmonton, AB (CA), August 2005.

[10] I. Ulrich and J. Borenstein. Vfh*: Local obstacle avoidance with look-ahead verification. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 2505–2511, San Francisco, CA, April 2000.

[11] Th. Fraichard. *Contributions à la planification de mouvement*. Habilitation à diriger des recherches, Inst. Nat. Polytechnique de Grenoble, Grenoble (FR), March 2006.

[12] Th. Fraichard and H. Asama. Inevitable collision states - a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[13] S. Blondin. Planification de mouvements pour vhicule automatis en environnement partiellement connu. Master thesis, Inst. Nat. Polytechnique de Grenoble, Grenoble (FR), June 2002.

[14] T. Lozano-Perez. Spatial planning, a configuration space approach. *IEEE Trans. on Computing.*, 32(2):108–120, February 1983.

[15] Th. Fraichard. Trajectory planning in a dynamic workspace: a 'state-time space' approach. *Advanced Robotics*, 13(1):75–94, 1999.

[16] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 341–346, Detroit, May 1999.

[17] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Switzerland, 2002.

[18] L.S Pontryagin. The mathematical theory of optimal processses. *International Series of Monographs in Pure and Applied Mathematics*, 55, 1986.

[19] Fast light toolkit. *http://www.easysw.com/ mike/fltk/index.php.*

[20] Algorithmic Solutions Software GMBH. Leda : Library of efficient data types and algorithms. *http://www.algorithmic-solutions.com/enleda.htm.*

[21] J. O Rourke. *Computational Geometry in C (Second Edition).* Cambridge University Press, 1998.