# Calcul des états de collision inévitable pour un véhicule de type voiture

Joel Schaerer

Joël Schaerer Insa de Lyon, Département Télécommunications, Services et Usages

# Calcul des états de collision inévitable pour un véhicule de type voiture

*Rapport de stage ingénieur*

*Tuteur :* Dr. Thierry Fraichard

*Août-Décembre 2003*


# 1 Introduction

J'ai effectué mon stage de dernière année à l'INRIA Rhônes-Alpes, sous la direction de Thierry Fraichard, chargé de recherche au projet e-Motion(anciennement Sharp). Mon travail sur les états de collision inévitable s'inscrit dans la problématique plus large d'un projet de véhicule entièrement automatique. Les états de collision inévitable cherchent plus particulièrement à répondre au problème de la sécurité de tels véhicules.

Ce rapport est divisé en deux parties : une première partie présente le contexte de mon stage. La deuxième partie présente le concept d'état de collision inévitable, indispensable à la compréhension de mon travail, puis décrit mon travail ainsi que les résultats obtenus.

Je tiens à remercier tout particulièrement mon tuteur ainsi que l'ensemble de l'équipe pour leur sympathie et leur aide qui m'ont accompagné tout au long du stage.

# Table des matières

## Première partie

# Contexte général du stage

## 2 Présentation de l'Inria

L'INRIA, institut national de recherche en informatique et en automatique placé sous la double tutelle des ministères de la recherche et de l'industrie, a pour vocation d'entreprendre des recherches fondamentales et appliquées dans les domaines des sciences et technologies de l'information et de la communication (STIC). L' institut assure également un fort transfert technologique en accordant une grande attention à la formation par la recherche, à la diffusion de l'information scientifique et technique, à la valorisation, à l'expertise et à la participation à des programmes internationaux. Jouant un rôle fédérateur au sein de la communauté scientifique de son domaine et au contact des acteurs industriels, l'INRIA est un acteur majeur dans le développement des STIC en France.

L'INRIA accueille dans ses 6 unités de recherche situées à Rocquencourt, Rennes, Sophia Antipolis, Grenoble , Nancy et Bordeaux, Lille, Saclay et sur d'autres sites à Paris, Marseille, Lyon et Metz, 3000 personnes dont 2500 scientifiques, issus d'organismes partenaires de l'INRIA (CNRS, universités, grandes écoles) qui travaillent dans près de 100 "projets" (ou équipes) de recherche communs. Un grand nombre de chercheurs de l'INRIA sont également enseignants et leurs étudiants (environ 700 préparent leur thèse dans le cadre des projets de recherche de l'INRIA.

L'INRIA développe de nombreux partenariats avec le monde industriel et favorise le transfert et la création d'entreprises (une soixantaine à ce jour) dans le domaine des STIC, notamment au travers de sa filiale INRIA-Transfert, promoteur de 4 fonds d'amorçage : I-Source 1 et 2 (domaine des technologies de l'information et de la communication), C-Source (multimédia) et T-Source (télécommunications). L'INRIA est actif au sein d'instances de normalisation comme l'IETF, l'ISO ou le W3C dont il a été le pilote européen de 1995 à fin 2002. Enfin l'institut entretient d'importantes relations internationales : en Europe, l'INRIA est fortement impliqué dans ERCIM, consortium qui regroupe 16 organismes de recherche et participe à 100 projets européens. A l'international, l'institut collabore avec de nombreuses institutions scientifiques (laboratoires de recherche conjoints à Moscou, Pékin, Mexico, etc, "équipes de recherche associées", programmes de formation et d'accueil)

Le budget est de 120 M€, dont 1/4 provenant de contrats de recherche et de produits de valorisation. Au cours des années 2000 - 2003, dans le cadre de son contrat quadriennal, les moyens de l'institut ont été augmentés de près de 50%.

La stratégie de l'institut repose sur la combinaison étroite de l'excellence scientifique et du transfert technologique. L'objectif essentiel de l'INRIA pour les années 2003-2007 est de réaliser des percées scientifiques et technologiques du meilleur niveau mondial dans le

cadre de sept grands défis prioritaires :

- Concevoir et maîtriser les futures infrastructures des réseaux et des services de communication
- Développer le traitement des informations et données multimédia
- Garantir la fiabilité et la sécurité des systèmes à logiciel prépondérant
- Coupler modèles et données pour simuler et contrôler les systèmes complexes
- Combiner simulation, visualisation et interaction
- Modéliser le vivant
- Intégrer pleinement les STIC dans les technologies médicales

# 3 Le projet e-Motion

Le projet e-Motion, en cours de création, est commun entre l'Inria, le CNRS, l'Institut National Polytechnique de Grenoble, et l'université Joseph Fourier de Grenoble ; il est localisé à l'Inria Rhône-Alpes, et appartient également au laboratoire Gravir[1] de la fédération Imag.

Le projet e-Motion a pour ambition de développer des modèles et des méthodes algorithmiques permettant à terme de construire des *"systèmes artificiels"* dotées de capacités de perception, de décision, et d'action suffisamment évoluées et robustes pour autoriser un fonctionnement de ceux-ci dans des *environnements ouverts* (i.e. partiellement connus) à *forte dynamicité* (i.e. où le temps et la dynamique jouent un rôle essentiel). Cette orientation des recherches sur le mouvement et l'action est essentielle pour pouvoir réellement envisager l'introduction de systèmes robotisés évolués et sécurisés dans notre vie quotidienne (i.e. des systèmes en forte interaction avec l'homme), comme par exemple dans les voitures et les systèmes de transport futurs (en particulier pour en améliorer la sécurité et le confort d'utilisation), ou encore dans les robots d'assistance à domicile (e.g. pour réaliser des tâches ménagères ou pour améliorer la vie de personnes handicapées ou dépendantes). Les progrès récents en matière de puissance informatique embarquée, de capteurs, et de systèmes mécatroniques miniaturisés, rend cette évolution envisageable et permet potentiellement d'aboutir au saut technologique nécessaire à un réel passage à l'échelle (comme par exemple un fonctionnement sécurisé dans une foule). Bien que cette nouvelle orientation des recherches commence à voir le jour dans la communauté scientifique, il n'existe actuellement que très peu de travaux effectifs, et l'équipe se place (de part son expérience passée) en position de précurseur sur certains de ces sujets.

Sur le plan méthodologique, l'équipe propose de combiner les avantages respectifs de la *géométrie algorithmique*, des *probabilités*, et dans certains cas de *l'inspiration biologique* (en travaillant pour cela avec des neuro-physiologistes). Ceci conduit à étudier sous ces différents angles de vue, trois sujets fondamentaux fortement corrélés :

[1] Laboratoire d'Informatique Graphique, Vision et Robotique.

- *La modélisation multi-modale de l'espace et du mouvement.* Il s'agit de construire en continue (à partir de connaissances préalables et de données perceptives diverses) et de faire cohabiter des modèles de natures différents, i.e. de modèles ayant des spécialisations fonctionnelles complémentaires comme le suggèrent les neuro-physiologistes.
- *La planification de mouvements dans le monde physique.* Les caractéristiques intrinsèques du monde physique conduisent à prendre simultanément en compte des contraintes de non collision, de dynamique, et de temps de réaction. Afin de maîtriser la complexité algorithmique du problème, des techniques permettant de raisonner sur des représentations appropriées de l'espace-temps (espace des vitesses instantanées, principe de planification itérative sous contraintes temporelles fortes) sont en cours de développement.
- *L'inférence probabiliste pour la décision.* Afin de pouvoir raisonner correctement sur les connaissances courantes du système et sur les incertitudes qui leurs sont associées, l'équipe propose d'utiliser le principe de "programmation bayésienne" développé au sein du projet, et qui offre un cadre formel pour réaliser de l'apprentissage automatique et des raisonnements basés sur l'inférence probabiliste.

Les principaux domaines d'application visés par cette problématique de recherche sont ceux qui visent à introduire des systèmes robotisés évolués et sécurisés dans notre espace de vie, afin d'accroître la sécurité et le confort d'utilisation des nouvelles technologies. Cette caractéristique se retrouve en particulier dans des applications comme la robotique de service et d'intervention, ou la voiture et les systèmes de transport futurs. Les retombées que l'on peut attendre de ces recherches sont par contre plus vastes, et couvrent des domaines aussi variés que l'interaction avec des agents autonomes dans un monde virtuel, la modélisation de fonctions sensori-motrices dans le vivant, ou encore des secteurs économiques éloignés de la robotique comme ceux de la finance ou de la maintenance industrielle (secteurs d'applications actuellement couverts par la start-up *Probayes*).

• *La robotique de service et d'intervention.* Le marché correspondant devrait réellement démarrer dès l'apparition de produits industriels réellement robustes, facilement utilisables par des non spécialistes, et d'un coût raisonable. On peut citer dans ce domaine les robots ménagers (comme par exemple les robots aspirateurs du commerce encore très cher et moyennement efficaces), les systèmes de surveillance actifs (e.g. robot rondier, sécurité civile ou militaire, etc.), les robots de loisir (comme par exemple le robot chien *Aibo* ou le robot humanoide de Sony), ou encore les systèmes d'assistance aux personnes agées et/ou handicapées.

• *Les voitures et les systèmes de transport du futur.* Ce secteur d'activité devrait être rapidement en pleine mutation pour des raisons économiques et sécuritaires maintenant bien connues. Les systèmes envisagés incluent des *mécanismes d'assistance* visant à améliorer le confort et la sécurité des usagers des voitures, et des fonctions de *conduite automatique*

permettant d'automatiser entièrement les déplacements de véhicules privés dans certaines circonstances (e.g. ACC, freinage d'urgence) et/ou dans certains secteurs aménagés (e.g. parkings automatisés ou flottes captives dans des centres villes).

• *Retombées dans d'autres secteurs d'application..* Les technologies logicielles développées ont potentiellement des retombées dans des domaines aussi variés que l'interaction avec des agents autonomes dans un monde virtuel (e.g. dans les jeux vidéo), la modélisation de fonctions sensori-motrices dans les systèmes biologiques afin d'aider les neuro-physiologistes à analyser le vivant et à étudier certaines pathologies (en coopération étroite avec des neuro-physiologistes), ou encore des secteurs économiques éloignés de la robotique comme ceux de la finance ou de la maintenance industrielle (secteurs d'applications actuellement couverts par la start-up *Probayes*, démarrée en octobre 2003 sur la base des outils de programmation bayésienne développés antérieurement par l'équipe).

## Deuxième partie

# Travail effectué

## 4 Présentation des états de collision inévitable

Le sujet de mon stage appartient au deuxième domaine d'interêt du projet e-Motion, la planification de mouvements dans le monde physique. L'objectif est d'intégrer à la planification de mouvement les contraintes de sécurité, indispensables à toutes les applications en milieu humain. La sécurité est une préocupation critique dans de nombreux domaines de la robotique. Alors qu'elle n'a qu'une importance toute relative dans les domaines où l'environnement est parfaitement connu, comme les robots industriels classiques, elle prend une importance capitale en robotique mobile, où l'environnement est seulement partiellement connu. On doit pouvoir garantir que le robot saura éviter la collision, quel que soit le comportement des obstacles. Cette garantie est particulièrement importante dans les environnements où le robot interagit avec des humains.

Un état de collision inévitable d'un véhicule est un état pour lequel, quel que soit la trajectoire empruntée par le véhicule, il entrera en collision avec un obstacle à un moment ou un autre. Les états de collision inévitable, décrits dans [3], introduisent un formalisme permettant de garantir que le robot ne se trouvera jamais en situation de collision inévitable, quel que soit le mouvement des obstacles présents dans le milieu. Il est seulement nécessaire de connaître certaines caractéristiques des obstacles comme leur vitesse maximale.

### 4.1 Planification et navigation

Les états de collision inévitable sont applicables aussi bien en planification qu'en navigation. La planification concerne la recherche initiale d'une trajectoire permettant d'atteindre un but défini. La planification se fait avant l'éxécution de la trajectoire, sur la base d'un modèle de l'environnement. La navigation s'occupe de l'éxécution de la trajectoire planifiée, et s'occupe entre autres de gérer l'imprécision dans l'exécution de celle-ci ou de réagir à des évenements imprévisibles en phase de planification, comme l'apparition d'obstacles inconnus dans le champ de vision du robot. Le concept de sûreté concerne à la fois le domaine de la planification et la navigation : il est intéressant de ne choisir que des trajectoires sûres, c'est à dire pour lesquelles le robot aura toujours une solution sans collision même si un obstacle imprévu apparaît à la limite de son champ de vision. Le concept est également important en navigation : il est important de garantir que le robot reste dans des états sûrs, même quand il est obligé de s'écarter de la trajectoire initialement planifiée.

## 4.2 Autres approches et limitations

Avant les états de collision inévitable, plusieurs pistes ont déjà été explorées pour tenter de répondre à ce besoin de sécurité : les méthodes à base de fenêtre dynamique ou de V-obstacles tentent toutes de répondre à ce problème. Cependant, aucune de ces méthodes ne permet de garantir ni l'absence de collision, ni que les états qu'elles donnent comme interdits sont effectivements des états pour lesquels la collision ne pourra pas être évitée. En effet, aucune de ces méthodes ne prend en compte l'ensemble des stratégies d'évitement de collision. Les états de collision inévitable, en contrepartie, permettent de garantir de façon formelle l'absence de collision.

## 4.3 Objectifs du stage

L'objectif du stage était d'avancer dans la caractérisation et les méthodes de calcul des états de collision inévitable, sur la base du travail accompli jusque là ([3],[1]). Plus particulièrement, mon travail a porté sur le développement d'une méthode de calcul numérique approchée et sur son implantation en C++. J'ai aussi réalisé une interface graphique pour pouvoir facilement éditer des configurations d'obstacles et voir rapidement le résultat.

## 4.4 Les états de collision inévitable

### 4.4.1 Etat et configuration

Avant d'aller plus loin, il est important de bien comprendre la notion d'état. L'état d'un robot caractérise entièrement la situation d'un robot à un moment donné. Ainsi, il contient non seulement toutes les données relatives à la position du robot, et l'orientation des différents composants (*sa configuration*), mais aussi les données relatives à son mouvement, comme sa vitesse linéaire, vitesse de rotation, etc.

### 4.4.2 Exemple

Dans cette section, nous allons définir formellement le concept d'état de collision inévitable, ainsi que le concept voisin d'obstacle de collision inévitable.

La figure 1 permet d'illustrer le concept sur un exemple. Soit $P$ un robot ponctuel ne pouvant que se déplacer vers la droite à une vitesse variable. Un état de $P$ est donc caractérisé par sa position $(x, y)$ et par sa vitesse $v$. Tous les états dont la position correspond au mur sont évidemment des états de collision. Considérant maintenant qu'il faut à $P$ une distance $d(v)$ pour s'arrêter, tous les états dont la position correspond au mur *et* les états situés à gauche du mur à une distance inférieure à $d(v)$ sont tels que, quoi que fasse $P$ dans le futur, il y aura collision. Ce sont des états de collision inévitable. Il est évident que le robot ne doit jamais se trouver dans un tel état.

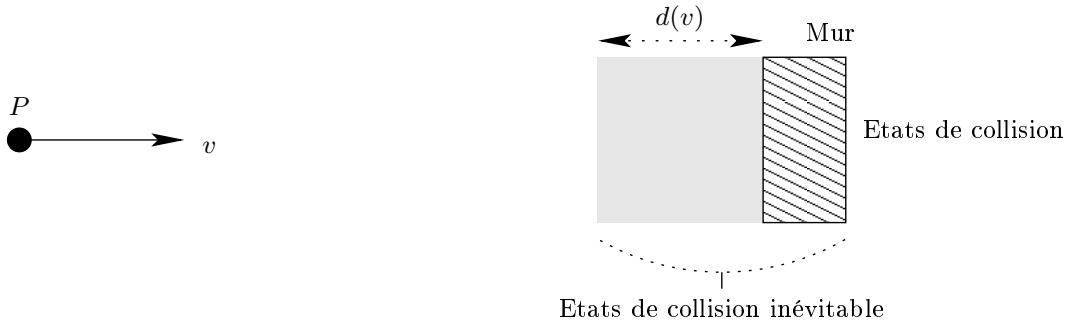


FIG. 1 – Etats de collision et états de collision inévitable.

En général, un *état de collision inévitable* peut se définir comme un état pour lequel, quelle que soit la suite de commandes appliquée au robot par la suite, une collision aura lieu.

### 4.4.3 Notations et définitions préliminaires

Pour pouvoir définir formellement le concept d'état de collision inévitable, il faut tout d'abord introduire quelques notations et définitions préliminaires.

Soit $\mathcal{A}$ un robot. Ses mouvements sont régis par l'équation suivante :

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

où $s \in \mathcal{S}$ est l'état de $\mathcal{A}$, $\dot{s}$ sa dérivée par rapport au temps et $u \in \mathcal{U}$ une commande. $\mathcal{S}$ et $\mathcal{U}$ représentent l'*espace des états* et l'*espace des commandes* de $\mathcal{A}$. L'espace des commandes contient l'ensemble des commandes qui peuvent être appliquées à $\mathcal{A}$.

Soit $\phi$ une séquence temporelle de commandes : $\phi : [0, T] \rightarrow \mathcal{U}$. $\phi$ représente une *trajectoire* de $\mathcal{A}$. On note $\Phi$ l'ensemble des trajectoires $\phi$ de $\mathcal{A}$. On note $s(t) = \phi(s_0, t)$ l'état atteint au temps $t$ en partant au temps 0 de l'état $s_0$ et en appliquant la séquence de commandes $\phi$. On note $\phi^{-1}(s, t)$ l'état $s_0$ tel que $\phi(s_0, t) = s$.

Soit $\mathcal{W}$ *l'espace de travail* de $\mathcal{A}$ (selon le type de robot, on a $\mathcal{W} = \mathbb{R}^2$ ou $\mathcal{W} = \mathbb{R}^3$). $\mathcal{W}$ contient un ensemble d'obstacles. Un obstacle $\mathcal{B}$ est défini comme un sous-ensemble de $\mathcal{W}$. On dit qu'un état $s$ est un état de collision avec $\mathcal{B}$ si et seulement si $\mathcal{A}(s) \cap \mathcal{B} \neq \emptyset$, où $\mathcal{A}(s)$ est le sous-ensemble de $\mathcal{W}$ occupé par le robot dans l'état $s$.

### 4.4.4 Définitions

Nous pouvons maintenant définir formellement le concept d'état de collision inévitable :

**Def. 1 (Etat de collision inévitable)** *Etant donné une trajectoire $\phi$, un état $s$ est un* état de collision inévitable(ECI) pour $\phi$ *si et seulement si $\exists t$ tel que $\phi(s, t)$ soit un état de collision.*

*Un état $s$ est un* ECI *ssi* $\forall\phi, \exists t$ *tel que* $\phi(s,t)$ *soit un état de collision.*

*De la même manière, $s$ est un* ECI avec un obstacle $\mathcal{B}$ *ssi* $\forall\phi, \exists t$ *tel que* $\phi(s,t)$ *soit un état de collision avec* $\mathcal{B}$.

**Def. 2 (Obstacle de collision inévitable)** *Etant donnés un obstacle $\mathcal{B}$ et une commande $\phi$, $OCI(\mathcal{B}, \phi)$, l'* obstacle de collision inévitable de $\mathcal{B}$ pour $\phi$ *est défini de la manière suivante :*

$$\begin{aligned} OCI(\mathcal{B},\phi) &= \{s \in \mathcal{S} | s \text{ est un ECI avec } \mathcal{B} \text{ pour } \phi\} \\ &= \{s \in \mathcal{S} | \exists t, \phi(s,t) \text{ est un état de collision avec } \mathcal{B}\} \end{aligned}$$

*$OCI(\mathcal{B})$, l'*obstacle de collision inévitable de $\mathcal{B}$, *est défini par :*

$$\begin{aligned} OCI(\mathcal{B}) &= \{s \in \mathcal{S} | s \text{ est un ECI avec } \mathcal{B}\} \\ &= \left\{ \begin{array}{c} s \in \mathcal{S} | \forall\phi, \exists t, \phi(s,t) \\ \text{est un état de collision avec } \mathcal{B} \end{array} \right\} \end{aligned}$$

### 4.4.5 Propriétés fondamentales

Voici maintenant deux propriétés fondamentales liées au calcul des états de collision inévitable. On peut en trouver la démonstration dans [3].

**Propriété 1** *Soit $\mathcal{B} = \bigcup_i \mathcal{B}_i$ l'ensemble des obstacles présents dans l'environnement et $\Phi$ l'ensemble des trajectoires.*

$$OCI(\mathcal{B}) = \bigcap_{\Phi} \bigcup_{i} OCI(\mathcal{B}_i, \phi)$$

Cette propriété donne la manière de calculer un $OCI()$ : pour chaque trajectoire, on fait l'union des états qui aboutissent à une collision avec chaque point des obstacles, puis on fait l'intersection des résultats obtenus.

**Propriété 2** *Soit $\mathcal{I} \subset \Phi$ un sous ensemble de l'espace $\Phi$ des trajectoires.*

$$OCI(\mathcal{B}) \subset \bigcap_{\mathcal{I}} OCI(\mathcal{B}, \varphi)$$

La propriété (2) va nous permettre de calculer en pratique une approximation des ECI pour des systèmes robotiques trop complexes. En effet, le nombre de trajectoires possibles est en général infini, ce qui rend pratiquement impossible le calcul exact de $OCI(\mathcal{B})$. Cette propriété nous montre que l'on peut en obtenir une approximation conservative(c'est à dire un surensemble de l'ensemble des ECI. On ne risque alors pas d'oublier des ECI, on ne fait que marquer comme étant des ECI des états qui ne le sont pas) en n'utilisant qu'un sous-ensemble de l'ensemble de toutes les séquences de commandes.

### 4.4.6 Extension aux obstacles mobiles et aux obstacles inconnus

On s'est limité jusqu'à présent au cas des obstacles immobiles et parfaitement connus. Dans les conditions réelles, seule une partie des obstacles est immobile, et il intéressant de considérer le cas où le mouvement et la position de certains obstacles ne sont pas connus. En phase de planification, le robot peut avoir accès à une carte du site sur lequel il va se déplacer, mais il ne peut pas connaître à l'avance la position des obstacles mobiles qu'il rencontrera. Pendant la navigation, les connaissances du robot sont limitées par la portée de ses capteurs. Il est important que la determination des états sûrs prenne en compte cette incomplétude des données. Heureusement, le formalisme des états de collision inévitable est tout à fait adapté pour prendre en compte cette incertitude.

**Obstacles mobiles à mouvement connu** Considérons tout d'abord le cas des obstacles mobiles au mouvement connu : la définition d'un obstacle devient dépendante du temps. On peut noter $\mathcal{B}_i(t)$ le sous ensemble $\mathcal{W}$ occupé par $\mathcal{B}_i$ au temps $t$. Avec cette notation, on redéfinit l'obstacle de collision inévitable :

$$OCI(\mathcal{B}_i, \phi) = \{s \in \mathcal{S} | \exists t, \mathcal{A}(\phi(s,t)) \cap \mathcal{B}_i(t) \neq \emptyset\}$$

Les deux propriétés présentées précédemment restent vraies.

**Obstacle mobiles à mouvement inconnu** Si le mouvement des obstacles est inconnu, il faut s'assurer que la collision pourra être évitée quel que soit le mouvement des obstacles. Pour pouvoir faire cela, il faut faire des hypothèses sur le mouvement des obstacles. En effet, si rien n'est connu à priori, aucun état ne peut être considéré comme sûr. Selon le contexte, on peut faire des hypothèses sur la vitesse maximale des obstacles, leur direction, la forme de leur trajectoire, etc. On note alors $\mathcal{T}_i$ l'ensemble des trajectoires possibles de l'obstacle $\mathcal{B}_i$.

Il suffit qu'il existe une combinaison de mouvements des obstacles pour laquelle un état $s$ est un ECI pour que l'état en question soit un ECI. Pour s'assurer qu'un état est sûr, il faut donc vérifier que l'état n'est un ECI pour aucune des combinaisons de mouvement possibles des obstacles. L'ensemble des combinaisons de mouvements possibles de tous les obstacles est le produit cartésien des $\mathcal{T}_i$ :

$$\mathcal{T} = \prod_i \mathcal{T}_i$$

S'assurer qu'un état est sûr pour toutes ces combinaisons revient à faire l'union des OCI correspondant à chacune des combinaisons. On a alors :

$$OCI(\mathcal{B}) = \bigcup_{\tau \in \mathcal{T}} OCI(\mathcal{B}^\tau) \tag{2}$$

**NB :** Si un obstacle peut avoir plusieurs trajectoires différentes, il ne peut en avoir qu'une seule à la fois. Il est donc faux de calculer l'OCI pour la superposition de toutes les différentes trajectoires.

### 4.5 Différentes approches du calcul des ECI

Le concept d'état de collision inévitable est très général, puisqu'il peut s'appliquer à des robots dans des espaces de configurations et de commandes de dimension quelconque. Dans certains cas simples, comme ceux qui sont examinés dans [1], cependant, il est possible de trouver des solutions analytiques. Posséder une solution analytique est très utile, cela permet par exemple de déterminer instantanément si un état est sûr ou non.

Malheureusement, dans les cas réels, l'équation de transition d'état et l'espace des commandes sont complexes, et la recherche d'une solution analytique s'avère très difficile, voire impossible. D'autre part, la solution n'est plus valable dès qu'on change de modèle de robot. Il faut alors reprendre le problème du début. Il est donc illusoire de chercher une solution analytique dans le cas le plus général. Pour toutes ces raisons, il est intéressant d'utiliser une méthode numérique approchée, basée sur une discrétisation de l'espace des états et du temps. Une telle méthode aura l'avantage d'être directement applicable à une grande variété de problèmes. Elle demande par contre de trouver un bon compromis entre vitesse de calcul et précision du résultat. Ce problème est particulièrement important quand un calcul en temps réel est requis.

## 5 Considérations algorithmiques

Au cours de ce stage, nous avons opté pour une méthode de calcul approchée basée sur la discrétisation de l'espace des états et de l'espace des commandes. Cette méthode est facilement adaptable à différents types de robots et d'obstacles. Dans cette section, l'algorithme utilisé est présenté avec un exemple de véhicule, puis plusieurs manières d'optimiser le calcul sont présentées. Le temps de calcul est en effet critique pour une utilisation en navigation.

### 5.1 Véhicule de type voiture

Dans le cadre de se stage, on s'est intéressé plus particulièrement à des véhicules de type voiture. Deux modèles ont été étudiés : un modèle pour lequel on contrôle l'orientation de la roue avant, et un modèle pour lequel on contrôle la vitesse de rotation de la roue avant (plus réaliste). Dans les deux cas, les véhicules étaient supposés ponctuels. Cette partie présente à titre d'illustration le fonctionnement de l'algorithme développé sur le premier modèle. Il est ensuite facile d'étendre son fonctionnement à d'autres types de véhicules.

$\mathcal{A}$ représente maintenant un véhicule du premier type. Un état de $\mathcal{A}$ est défini par $s = (x, y, \theta, v)$ où $(x, y)$ est la position de la roue arrière, $\theta$ l'orientation de $\mathcal{A}$, et $v$ la vitesse linéaire de la roue avant. L'équation (1) s'écrit maitenant :

$$\begin{cases} \dot{x} = v \cos\theta \cos\xi \\ \dot{y} = v \sin\theta \cos\xi \\ \dot{\theta} = v \sin\xi / b \\ \dot{v} = a \end{cases} \tag{3}$$

Dans le cas de ce système, il est possible d'intégrer analytiquement ce système d'équations et d'obtenir ainsi les équations de la trajectoire. En général, ce n'est pas possible. On utilise alors une méthode d'intégration numérique, comme la méthode de Runge-Kutta [4]. L'intégration numérique offre une plus grande généralité. Elle donne cependant une moins bonne précision que les méthodes analytiques, et s'avère être une opération relativement coûteuse en termes de temps de calcul.

## 5.2 Représentation des données

Reprenons le modèle défini par le système d'équations différentielles (1). Un état est définit par le 4-uplet $(x, y, \theta, \xi)$. L'algorithme présenté ici va calculer pour $\theta$ et $\xi$ donnés les positions $(x, y)$ qui correspondent à des ECI. On obtient donc l'ensemble des ECI appartenant à une *tranche* de l'espace des états. L'ensemble complet des ECI est obtenu en considérant les differentes tranches pour toutes les valeurs de $\theta$ et $\xi$.

On associe à la tranche considérée un bitmap booléen, dont chaque pixel représente une position $(x, y)$. Un pixel à **vrai** signifie que l'état correspondant est un ECI.

## 5.3 Algorithme utilisé

### 5.3.1 Discrétisation

Comme indiqué précédemment, l'algorithme fonctionne par discrétisation de l'espace des états et du temps. La discrétisation intervient à trois niveaux différents :

**Espace des états** En ce qui concerne l'espace des états, nous avons utilisé une discrétisation régulière pour toutes les dimensions. On utilise donc un paramètre unique, le *pas de discrétisation des états*, $\delta s$. De ce paramètre dépend directement la précision des régions de collision inévitable obtenues. Il est facile de voir que l'utilisation de mémoire pour une tranche varie avec le carré de $\delta s$.

**Temps** L'algorithme fonctionne en calculant pour chaque point des obstacles les états $s_0$, tels que $s = \phi(s_0, t)$, qui représentent les états pour lesquels on aura une collision à l'instant $t$ avec l'obstacle situé en $s$. Jusqu'à quel instant $t$ faire les calculs ? Il est nécessaire de fixer

une limite dans le temps : c'est ce que l'on va appeler l'horizon temporel. Ce paramètre, dénoté $t_{max}$, est d'une importance cruciale : un état calculé comme sûr sera garanti sans collision jusqu'à $t_{max}$, mais rien ne sera garanti pour la suite. A l'intérieur de la plage $[0, t_{max}]$, le temps doit être discrétisé. On obtient ainsi un ensemble d'instants $t \in \mathbb{T}$ espacés de $\delta t$ pour lesquels on cherchera les états $s_0$, tels que $s = \phi(s_0, t), \forall t \in \mathbb{T}$.

**Trajectoires** L'espace des trajectoires possibles du véhicule, $\Phi$, doit lui aussi être discrétisé. On s'est limité aux trajectoires à commandes constantes dans le temps. Le problème revient donc à discrétiser l'espace des commandes $\mathcal{U}$. Pour cela on a utilisé une discrétisation uniforme de chacun des paramètres : dans l'exemple cité précédemment, on prend un certain nombre de valeurs uniformément réparties de a et de $\xi$, et on utilise l'ensemble $\mathcal{I} \subset \Phi$ des trajectoires obtenues avec toutes les combinaisons de ces valeurs.

### 5.3.2 Principe de base

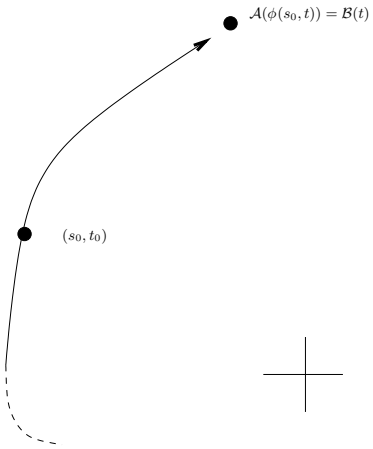


FIG. 2 – Calcul d'un OCI

1. On considère tout d'abord un obstacle ponctuel $\mathcal{B}$ et une trajectoire donnée $\phi$. On rappelle que $\mathcal{A}$(s) est le point occupé par le robot dans l'état $s$. Pour chaque instant de $\mathbb{T}$, on calcule pour $\phi$ les états $s_0$, tels que $\mathcal{A}(\phi(s_0, t)) = \mathcal{B}(t)$ (voir figure 2). L'ensemble de ces états forme une approximation discrète de $OCI(\mathcal{B}, \phi)$ pour $\mathcal{B}$ ponctuel.
2. On peut maintenant étendre le calcul aux obstacles de forme arbitraire, toujours pour une trajectoire donnée. En application de la propriété (1) , on a $OCI(\mathcal{B}, \phi) = \bigcup_i OCI(\mathcal{B}_i, \phi)$. Pour calculer cette union, on initialise un bitmap $M(\mathcal{B}, \phi)$ à **faux** : initialement aucun état n'est marqué comme ECI. Ensuite, pour chaque point $\mathcal{B}_i$ des obstacles et chaque instant $t \in \mathbb{T}$, on calcule l'état $s_0$ comme indiqué précédemment, et on marque la case correspondante de $M(\mathcal{B})$ à **vrai**. A la fin de l'opération, l'ensemble des

cases de $M(\mathcal{B},\phi)$ à **vrai** forme une approximation de $OCI(\mathcal{B},\phi)$.

3. Nous n'avons calculé jusqu'à présent qu'un OCI pour une trajectoire donnée. En application de la propriété (1), on peut écrire $OCI(\mathcal{B}) = \bigcap_{\mathcal{I}} OCI(\mathcal{B},\phi)$. Pour obtenir le résultat désiré, on fait l'intersection pixel à pixel des $M(\mathcal{B}\phi)$ pour tout $\phi$ en utilisant l'opérateur ET logique.

On appelle $M(\mathcal{B})$ le bitmap résultant. L'ensemble des cases vraies de $M(\mathcal{B})$ représentent $OCI(\mathcal{B},\phi)$ pour la tranche considérée.

### 5.3.3 L'algorithme

Voici maintenant une description plus formelle de l'algorithme utilisé :

Début de l'algorithme

INITIALISATION
Initialisation de $M(\mathcal{B})$ à **true**
Soit une tranche donnée de l'espace des états, définie par $(\theta,\xi)$
DEBUT
POUR toute trajectoire $\phi \in \mathcal{I}$ FAIRE
Initialisation du bitmap temporaire $M(\mathcal{B}_i,\phi)$ à **false**
POUR tout point $\mathcal{B}_i \in \mathcal{B}$ FAIRE
POUR tout instant $t \in \mathbb{T}$ FAIRE
Marquer la position $\mathcal{A}(s_0)$, tq $\mathcal{A}(\phi(s_0,t)) = \mathcal{B}_i(t)$ à **true**
FIN_POUR
$M(\mathcal{B})$ =$M(\mathcal{B})$ **ET** $M(\mathcal{B},\phi)$
FIN_POUR
FIN POUR
FIN

---

### 5.3.4 Calcul de $s_0$

Nous avons jusqu'à présent exposé le principe de base de l'algorithme sur un exemple. Comment calculer $s_0$ tq $\mathcal{A}(\phi(s_0,t)) = \mathcal{B}(t)$ ?

Pour chaque trajectoire $\phi$, on peut calculer les $\Delta x$ et $\Delta y$ pour chaque instant $t$ de $\mathbb{T}$, comme le montre la figure (3). Etant donné que $\Delta x$ et $\Delta y$ ne dépendent pas de la position de départ, alors l'état $s_0$ tel que $\mathcal{A}(\phi(s_0,t)) = \mathcal{B}(t)$ est défini par :

$$\begin{cases} x(s_0) = x(\mathcal{B}(t)) - \Delta x \\ y(s_0) = y(\mathcal{B}(t)) - \Delta y \end{cases}$$

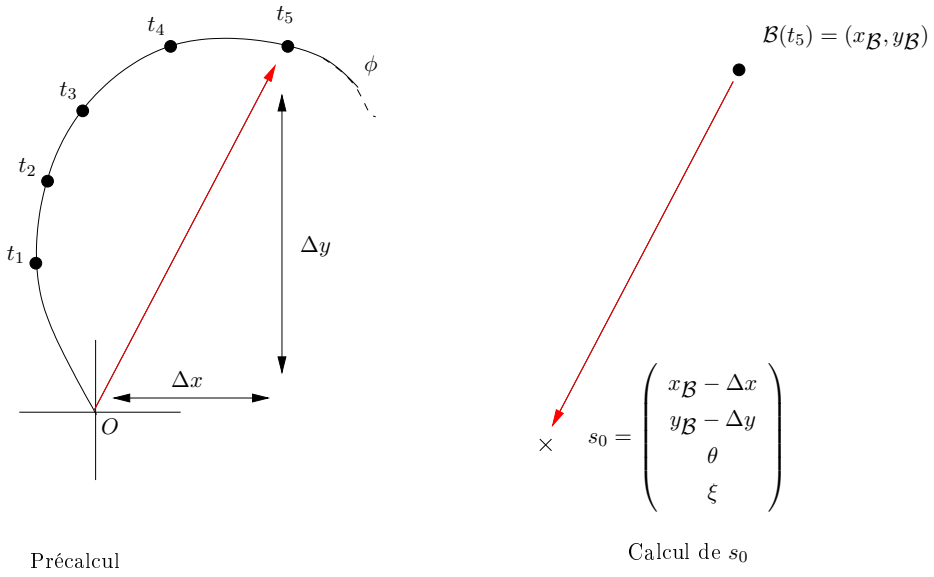


FIG. 3 – Calcul de $S_0$

### 5.4 Choix des pas de discrétisation

On voit dans l'algorithme qu'une discrétisation intervient à plusieurs reprises : discrétisation de l'espace des états, de l'espace des trajectoires, de l'espace de travail, du temps... Pour chacune de ces opérations, il faut fixer le pas de discrétisation, en cherchant un bon compromis entre précision du calcul, temps de calcul et utilisation de mémoire.

Pour simplifier les choses, il est important de remarquer que plusieurs de ces paramètres sont liés :

1. Le temps est lié à la discrétisation des états : il est inutile d'avoir un pas de temps plus fin que le temps mis pour passer d'un état à un état voisin à la vitesse maximale rencontrée dans l'environnement.
2. La discrétisation de l'espace de travail (utilisée pour discrétiser la forme du robot et des obstacles) est elle aussi liée à celle de l'espace des états.

Finalement il ne reste plus que deux paramètres à fixer : les pas de discrétisation des états et des commandes.

### 5.5 Optimisation

Il y a plusieurs manières d'optimiser l'algorithme exposé précédemment :

**Interpolation linéaire** Il est possible d'interpoler la trajectoire du robot par des segments de droite. Cela permet de ne calculer la position exacte du robot que pour des instants plus éloignés. On marque alors les états intermédiaires en utilisant un algorithme de tracé de ligne, comme l'algorithme de Bresenham ([2]). L'interêt de cette optimisation

est évident : calculer la position exacte du robot à un instant donné est une opération coûteuse, surtout si elle est obtenue par intégration numérique.

**Précalcul des trajectoires** Il est inutile de recalculer la trajectoire du robot à chaque point des obstacles. Il suffit de la calculer une fois au début, et de stocker le résultat. L'interêt de cette optimisation est le même que pour l'intepolation linéaire.

**Intersection** Le **et** logique entre le bitmap temporaire et le bitmap résultat devient rapidement une opération coûteuse quand la précision demandée augmente. Pour accélerer cette opération il y a plusieurs manières de procéder :

1. Il est possible de rejeter cette opération à la fin. On calcule d'abord les ICO pour chaque trajectoires, puis on fait l'intersection d'un coup. Dès qu'on rencontre une trajectoire pour laquelle un état n'est pas un état de collision inévitable, on marque l'état comme sûr et l'on passe à l'état suivant. De cette manière on évite de tester pour toutes les trajectoires. Comme la proportion d'états de collision inévitable est généralement très faible, cette méthode est particulièrement efficace. Cependant, comme elle oblige à conserver tous les résultats intermédiaires, elle est particulièrement gourmande en mémoire.
2. Il est possible de représenter le bitmap résultat comme une liste d'états de collision inévitable, plutôt qu'un tableau. De cette manière, au moment de faire le ET logique, on ne considère que les états qui peuvent encore être des ECI. Cependant, supprimer un élément d'une liste est une opération plus longue que d'écrire dans un tableau. Il faudrait donc tester cette méthode pour vérifier qu'elle améliore la vitesse d'execution.

## 5.6 Perspectives

Dans cet algorithme, nous n'avons considéré que le cas d'un robot ponctuel. Dans le cas d'un robot réel, les choses deviennent légèrement plus compliqués car il faut alors considérer la forme du robot en plus de celles des obstacles.

Cependant, dans le cadre de la robotique mobile, qui est l'application étudiée au cours de ce stage, l'approximation de la voiture à un point n'est pas trop mauvaise, et elle permet de limiter le temps de calcul à des valeurs acceptables.

Seules les trajectoires à commandes constantes ont étés considérées. Tant que la limite de calcul dans le temps n'est pas trop lointaine, ce problème reste peu important.

La discrétisation des obstacles, de manière à éviter de manquer des ECI sans pour autant faire des calculs inutiles, n'est pas un problème facile. Le programme actuel manque des états (ce qui se traduit par des "trous" dans les OCI). Il reste donc du travail dans ce domaine.

# 6 Choix technologiques et méthodes de travail

Compte tenu de la difficulté de trouver des solutions analytiques dans les cas réels, il a rapidement été décidé de développer un programme de calcul numérique des ECI. Dans cette section, je détaille les choix qui ont été faits ainsi que les raisons qui ont motivé ces choix.

## 6.1 Choix du langage

Le langage choisi devait répondre à trois contraintes :

- La vitesse d'éxécution
- La possibilité de s'interfacer avec des bibliothèques existantes, en particulier la MSL[2], une librairie open-source contenant de nombreux modèles de véhicules et plusieurs algorithmes de planification de trajectoires.
- La possibilité de pouvoir étendre facilement le programme, en particulier en rajoutant des types de robots et d'obstacles.

Toutes ces contraintes m'ont fait m'orienter naturellement vers le C++, la MSL ainsi que de nombreuses autres bibliothèques libres d'accès étant écrites en C++. Le fait d'uiliser un langage orienté objet donnait de bonnes perspectives d'extensibilité. La première contrainte éliminait d'emblée des langages potentiellement plus simples à utilser comme Java. Les spécifications du programme sont reproduites intégralement dans l'annexe A.

## 6.2 Librairies

Pour éviter de perdre du temps sur des choses sans réel rapport avec mon sujet, j'ai utilisé plusieurs librairies pour réaliser plus facilement certaines fonctionalités de mon programme. La plus importantes était une librairie de développement d'interface graphique. En effet, la création d'une interface graphique intuitive et efficace reste une tâche difficile et longue. L'utilisation d'une librairie bien établie permet de simplifier les choses.

Il existe plusieurs librairies de ce type disponibles en open-source. Mon choix a porté sur plusieurs critères :

- Degré de maturité de la librairie
- Qualité de la documentation et possibilité de support
- Simplicité d'utilisation
- Possibilité de porter facilement le code d'une plateforme à une autre (Linux, Windows, ...)

Après concertation avec mon tuteur, mon choix s'est finalement porté sur wxWindows[3]. C'est une librairie open-source, robuste car développée depuis plus de dix ans et utilisée

[2] Motion Strategy Library, http ://msl.cs.uiuc.edu/msl/

[3] http ://www.wxwindows.org/

par un certain nombre de projets importants. Un argument qui a joué en sa faveur est que j'avais déjà une certaine expérience avec cette librairie.

J'ai également utilisé la STL (Librairie standard du C++) et la MSL, qui m'ont gagné beaucoup de temps en m'évitant d'avoir à résoudre de nombreux complexes comme la gestion dynamique de la mémoire ou l'intégration numérique.

## 6.3 Outils de développement

J'ai travaillé en grande partie seul sur mon projet. Mon tuteur est l'auteur de l'article sur les ECI, et le seul à travailler sur le concept à l'Inria. Mon travail consistait à faire avancer le concept en développant des moyens de calcul et une interface de visualisation de manière à pouvoir explorer rapidement le concept.

Aillant travaillé seul, je n'ai pas eu d'aspects complexes de travail en parallèle et de coordination à gérer. J'ai cependant utilisé un certain nombre d'outils de développement pour optimiser ma productivité et assurer la possibilité d'une continuité sur mon travail, c'est à dire permettre à mon ou mes successeurs d'avancer sur la base de mon travail.

### 6.3.1 Gestion des sources

Tout projet de développement logiciel un peu élaboré nécessite un système de gestion des sources. Même dans le cadre de mon projet, j'ai décidé d'utiliser un outil de ce type, CVS(*Concurent Version System*). Cela m'a permis de garder une trace des changements effectués dans mon programme et de revenir en arrière en cas de modification erronée. Les bénéfices que j'en ai retiré ont largement compensé le temps passé à maîtriser l'outil au début.

### 6.3.2 Documentation automatique

Dans le cadre d'un travail de stage, donc sur une période nécessairement courte par rapport au temps nécessaire pour aller de l'idée à son application, il était important pour moi de penser dès le début aux gens qui travailleraient après moi sur le même sujet. Pour permettre la réutilisation de mon code source, j'ai décidé d'utiliser un outil de génération de documentation à partir du code source, *doxygen*. Cet outil génère automatiquement une documentation dans un format agréable à consulter à partir du code source, et avertit quand des variables ou des fonctions ne sont pas documentées.

L'avantage principal de ce type d'outil est qu'il garantit que la documentation est à jour par rapport au programme. En effet, il est simple de modifier les commentaires à chaque fois que l'on apporte une modification au logiciel, et la documentation est alors automatiquement actualisée.

Une documentation au niveau des sources présente de plus l'avantage de permettre de comprendre non seulement ce que fait le programme, mais aussi comment il le fait de

manière à permettre à quelqu'un qui n'a pas développé le programme de lui rajouter des fonctionalités sans trop de difficultés. Elle m'a entre autres permis de revenir sur des parties auquelles je n'avais pas touché depuis un moment. Il est en effet difficile, même pour le développeur lui-même, d'avoir une vision globale de son programme quand celui-ci prend une taille importante (à la fin de mon stage, l'ensemble de mes sources représente plus de 6000 lignes de code, soit une centaine de pages).

### 6.4 Planification et organisation

J'ai organisé mon temps de stage de la façon suivante :

- Une phase de compréhension du problème, avec lecture des principaux articles sur le sujet et des rapports de mes prédécesseurs.
- Une phase de conception de l'algorithme et de modélisation objet du problème.
- La phase de programmation à proprement parler
- Un certain nombre de cycles d'amélioration, avec à chaque fois un cycle analyse/conception/développement/tests

Pour faciliter le développement et simplifier l'analyse, j'ai divisé le problème en plusieurs parties :

- La réalisation d'un moteur de calcul, avec une interface utilisateur minimale
- Une interface utilisateur graphique (GUI) digne de ce nom
- Le système de configuration par fichiers
- Les fonctionalités d'export des résultats.

La division du problème en problèmes plus petits permet une accélération du cycle de développement, ce qui permet entre autres de détecter les erreurs de conception plus vite, d'améliorer l'extensibilité du programme en cloisonnant ses différentes parties, et de simplifier grandement la phase d'analyse en limitant le nombre de paramètres à prendre en compte. L'utilisation d'un langage de programmation orienté objet permet de refléter cette séparation logique directement dans le code du programme de manière à mieux l'exploiter.

Etant seul sur mon projet, je n'ai pas eu d'aspects complexes de travail collaboratif à gérer.

## 7 Résultats obtenus

La principale réalisation de mon stage a été un programme de calcul des états de collision inévitables avec une interface graphique pour l'édition de l'environnement et la visualisation des résultats. Il dispose d'un certain nombre de fonctionalités :

- Gestion de deux types de véhicules entièrement configurables
- Prise en compte d'obstacles linéaires ou en forme d'arc de cercle, avec des mouvements élaborés