UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE 427 :
INFORMATIQUE PARIS SUD

Laboratoire : Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur (CNRS UPR 3251)

THÈSE DE DOCTORAT

INFORMATIQUE

par

Kangni KUEVIAKOE

Localisation multi-capteurs garantie
Résolution d'un problème de satisfaction de contraintes

Date de soutenance :30/09/2014

Composition du jury :

Directeur de thèse :	Alain LAMBERT	MCF-HDR (IEF-CNRS)
Co-directeur de thèse :	Philippe TARROUX	Professeur (LIMSI-CNRS)
Rapporteurs :	Vincent VIGNERON	MCF-HDR (IBISC, Université d'Evry)
	David FILLIAT	Professeur (ENSTA ParisTech)
Examinateurs :	Anne VILNAT	Professeur (LIMSI-CNRS)
	Olivier ORFILA	Chargé de recherche (IFSTTAR)

Résumé

Cette thèse traite de la localisation de véhicule. Plusieurs méthodes sont utilisées pour résoudre ce type de problème. Elles peuvent être classées en deux grandes catégories d'approches : les approches probabilistes et les approches déterministes. Ce travail aborde les approches déterministes et plus précisément l'approche ensembliste basée sur l'analyse par intervalles. Les travaux ont été conduits sur des jeux de données réelles collectées en environnements d'extérieur comprenant des capteurs proprioceptifs et extéroceptifs.

Lorsque l'on met en jeu plusieurs capteurs fournissant des informations complémentaires ou redondantes, il est important de fusionner les données afin d'améliorer la pose estimée. L'approche détaillée dans ce document utilise les méthodes intervalles et présente le problème de la localisation sous la forme d'un problème de satisfaction de contraintes.

La résolution se fait à l'aide d'un solveur intervalle. Plusieurs algorithmes ont été comparés. Un constat s'est dégagé : les algorithmes de consistance locale ne corrigent pas l'incertitude sur l'orientation. Cette thèse propose une méthode de localisation utilisable dans des applications temps réel et qui corrige l'incertitude sur le cap du véhicule. Nous avons comparé nos résultats à ceux du filtre de Kalman étendu (méthode probabiliste de référence) et mis en avant un des intérêts de notre méthode : l'assurance d'une consistance de la pose (position et orientation du mobile).

Cette thèse propose deux contributions. La première est d'ordre méthodologique. Dans l'état de l'art tous les travaux affirment la nécessité (voire l'obligation) d'une décomposition préalable des contraintes du problème avant l'étape de résolution. Nos travaux permettent de prouver le contraire. La deuxième contribution concerne la réduction du domaine de l'incertitude en orientation en couplant la propagation de contraintes et une approche de bissection.

Mots clés

Localisation, Fusion de données, CSP, Analyse par intervalles, Propagation de contraintes, GPS, Véhicule.

Abstract

This thesis deals with the vehicle location. Several methods are used to solve this kind of problem. They can be classified into two broad categories of approaches : probabilistic approach and deterministic approaches. This work addresses the deterministic approaches and more precisely the approach based on interval analysis. The work has been conducted on real data sets collected in outdoor environments with proprioceptive and exteroceptive sensors.

When multiple sensors providing complementary or redundant information are put into play, it is important to merge the data to improve the estimated pose. The approach detailed in this document uses the intervals methods and presents the localization problem as a constraint satisfaction problem.

The resolution is done using a solver interval. Several solvers were compared. One thing is clear : local consistency algorithms do not address the uncertainty of the orientation. This thesis proposes a method of locating usable in real time applications and corrects the uncertainty in the heading of the vehicle. We compared our results with those of the extended Kalman filter (probabilistic reference method) and highlighted one of the interests of our method : the assurance of consistency of the pose (position and orientation of the mobile).

This thesis proposes two contributions. The first is methodological. In the state of the art all works affirm the need (or obligation) to pre-decompose the constraints of the problem before the resolution step. Our work allows to prove otherwise. The second contribution relates to the reduction of the orientation uncertainty by combining constraint propagation and a bisection approach.

Keywords

Localization, Data Fusion, CSP, Interval Analysis, Constraint Propagation, GPS, Vehicle.

Remerciement

Je tiens à remercier Alain Lambert et Philippe Tarroux, mes directeurs de thèse, pour leur disponibilité, leurs encouragements et surtout leur patience dans l'encadrement de ce travail. Ils ont su m'aider à tenir le cap jusqu'au bout. Les conseils, propositions et suggestions de mes directeurs de thèse m'ont guidé dans la prise de décisions et m'ont permis de mener à bien mon travail. Je remercie particulièrement Alain pour son suivi continu presque quotidien pendant ces quatre années de thèse.

J'aimerais adresser ma gratitude à mes rapporteurs Vincent Vigneron et David Filliat pour le temps qu'ils ont accordé à la lecture minutieuse de mon manuscrit et pour leurs remarques ainsi que l'intérêt qu'ils ont porté à mon travail en général.

Je voudrais aussi remercier les autres membres de mon jury de thèse notamment Olivier Orfila et Anne Vilnat qui m'a fait l'honneur de présider ce jury.

Mes remerciements vont également à l'endroit de tous mes collègues de bureau au LIMSI (bureau 205) Mathieu, Cédric, Sonia, Sylvain, Hervé, Céline, Caroline et (bureau 208) Asma, Franck, Yacine. J'adresse un merci particulier à Yacine et Franck pour leurs conseils et pour l'atmophère de travail qui règne dans le bureau. Merci à mes autres amis du LIMSI, Adrien, Houda, David, Hui..., la liste est longue.

Je remercie mes collègues et les responsables du laboratoire LIVIC principalement Dominique, Rachid, Samia, Gerome, Redouane, Sourayat... Les mois, que j'ai passés au sein du LIVIC, ont été très décisifs pour la réalisation de ce travail de thèse. Et vous avez contribué à créer une ambiance de travail autour.

J'aimerais aussi remercier tous les membres du LIMSI et du LIVIC pour l'ambiance conviviale et le beau cadre de travail qu'ils m'ont procuré durant le temps que j'ai passé dans ces deux laboratoires.

A mes amis de la communauté CCH qui ont été là, aussi bien humainement que spirituellement, pour moi dans des moments particuliers, je leur dis merci.

Je remercie aussi mes parents qui m'ont ouvert les portes de ce monde. Ils m'ont permis de voir la beauté dans l'eau, la terre, le feu, le vent, tout en me posant des questions sur le monde. J'adresse un merci à mes frères pour avoir toujours cru en moi et m'avoir supporté et encouragé à avancer dans la vie et dans mes projets.

Pour finir, je tiens à adresser un grand merci à tous ceux pour qui j'ai omis involontairement les noms. Veuillez tous trouver dans ces quelques lignes, l'expression de ma sincère gratitude.

Table des matières

1 Introduction générale **21**
- 1.1 Introduction . . . 21
- 1.2 Objectifs et hypothèses . . . 22
- 1.3 Plan du document . . . 23

2 Robotique et véhicules intelligents **25**
- 2.1 Notion de systèmes intelligents . . . 26
- 2.2 Systèmes de perception . . . 28
  - 2.2.1 Description des capteurs . . . 28
  - 2.2.2 Classification des capteurs . . . 33
- 2.3 Plateforme expérimentale (CARLLA) . . . 34
- 2.4 Conclusion . . . 36

3 Analyse par intervalles **37**
- 3.1 Introduction . . . 38
  - 3.1.1 Les origines . . . 38
  - 3.1.2 Motivations . . . 39
- 3.2 Arithmétique des intervalles . . . 42
  - 3.2.1 Intervalles . . . 42
  - 3.2.2 Propriétés d'un intervalle . . . 43
  - 3.2.3 Fonctions élémentaires . . . 45
  - 3.2.4 Opérations ensemblistes . . . 45
  - 3.2.5 Librairies pour programmer avec les intervalles . . . 46
- 3.3 Généralisation des intervalles . . . 47
- 3.4 Fonctions d'inclusion . . . 49
- 3.5 Sous-pavages . . . 51
- 3.6 Problèmes liés à l'arithmétique des intervalles . . . 52
  - 3.6.1 Le problème de dépendance . . . 52
  - 3.6.2 L'effet enveloppe . . . 53
- 3.7 Conclusion . . . 53

4 Satisfaction de contraintes à intervalles **55**
- 4.1 Introduction . . . 56
- 4.2 Problème de satisfaction de contraintes . . . 58
- 4.3 Problème de satisfaction de contraintes à intervalles (ICSP) . . . 59
- 4.4 Notion de CSP dynamique . . . 60
- 4.5 Les algorithmes de propagation de contraintes à intervalles (ICP) 62

4.5.1 Notion de consistance . . . 62
4.5.2 La consistance d'enveloppe . . . 63
4.5.3 La consistance de boite . . . 70
4.5.4 kB-Consistance . . . 74
4.6 Les solveurs à intervalles . . . 78
4.6.1 Numerica . . . 78
4.6.2 RealPaver . . . 78
4.6.3 ALIAS . . . 79
4.6.4 ICOS . . . 79
4.6.5 GloptLab . . . 79
4.6.6 Ibex . . . 79
4.6.7 Choix de solveur . . . 79
4.7 Conclusion . . . 80

5 Localisation multi-sensorielle garantie 83
5.1 Introduction . . . 84
5.2 Les différents types de localisation . . . 85
5.2.1 Suivi de position . . . 85
5.2.2 Localisation globale . . . 86
5.3 Caractéristiques des environnements . . . 86
5.3.1 Environnement d'intérieur ou d'extérieur . . . 86
5.3.2 Environnement statique ou dynamique . . . 87
5.4 Autres aspects du problème de localisation . . . 87
5.5 Les approches de localisation en robotique mobile . . . 89
5.5.1 Les approches probabilistes . . . 89
5.5.2 Les approches ensemblistes . . . 92
5.6 Application des algorithmes ICP à la localisation . . . 94
5.6.1 Problématique . . . 94
5.6.2 Approches possibles . . . 96
5.6.3 Processus de localisation . . . 97
5.6.4 Résultats . . . 100
5.6.5 Conclusion . . . 105
5.7 Proposition de correction du cap du véhicule . . . 106
5.7.1 Présentation du problème . . . 107
5.7.2 Formulation du problème de localisation sous forme de νDICSP 110
5.7.3 Résultats . . . 114
5.7.4 Comparaison d'ICPS avec l'EKF . . . 118
5.7.5 Conclusion . . . 120
5.8 Conclusion . . . 120

6 Conclusion et perspectives 123
6.1 Conclusion . . . 124
6.2 Perspectives . . . 125
6.3 Publications scientifiques . . . 126

A L'algorithme SIVIA 127

B Méthode intervalle de Branch&Bound **131**

C Les filtres probabilistes **133**
C.1 Le filtre Bayesien basique 133
C.2 Filtre de Kalman (KF) 133
C.3 Le filtre de Kalman étendu (EKF) 134

D Algorithme de Waltz **137**

E Les techniques de consistance sur les domaines finis **141**
E.1 La consistance de nœud 141
E.2 La consistance d'arc 142
E.2.1 Algorithme REVISE 143
E.2.2 Algorithme AC-1 144
E.2.3 Algorithme AC-2 145
E.2.4 Algorithme AC-3 145
E.2.5 Algorithme AC-4 146
E.3 La consistance de chemin 148

F Un exemple de DICSP **151**

G L'algorithme HC3 **155**
G.1 Exemple de filtrage par HC3. 156
G.2 Évaluation des fonctions de projection 157
G.3 Décomposition des contraintes 157

Table des figures

2.1 Exemple de système intelligent 26
2.2 Principe de fonctionnement de l'odomètre 28
2.3 Schéma de fonctionnement général d'une Centrale inertielle . . . 30
2.4 Constellation de satellites du GPS 31
2.5 Récepteur GPS-RTK 31
2.6 Plate-forme CARLLA du LIVIC (Lambert et al., 2009) 35
2.7 Trois véhicules instrumentés du LIVIC-IFSTTAR. De gauche à droite : VIPER, CARLLA,
2.8 Centrale inertielle Crossbow VG 400 35
2.9 Récepteur GPS Ag132 (avec antenne) 36

3.1 Quelques types de vecteurs d'intervalles 48
3.2 *Pavé* $[x]=([x_1],[x_2])^T$ 48
3.3 Fonction d'inclusion 50
3.4 Convergence et monotonie d'une fonction d'inclusion 50
3.5 Exemples de sous-pavages à différentes résolutions 52
3.6 Effet enveloppe : Non-optimalité de la représentation par des rectangles 53

4.1 Illustration d'un CSP dynamique (Restriction) 61
4.2 Illustration d'un CSP dynamique (Relaxation) 61
4.3 Application de HC4 (propagation-avant) 69
4.4 Application de HC4 (Propagation-arrière) 69
4.5 Exemple de réduction de la borne gauche (Rueher, 2005) 78

5.1 Processus classique de localisation (par carte)(Siegwart et al., 2011) 84
5.2 Principe de la localisation ensembliste 98
5.3 Piste de Satory 101
5.4 Piste de Satory avec la trajectoire du véhicule 101
5.5 Exemple de boîte résultat de localisation ICP 102
5.6 Surface moyenne de localisation pour l'algorithme HC4 et le temps de calcul pour les différ
5.7 Surface de localisation ICP par étape 104
5.8 Incertitude d'orientation 105
5.9 Intervalle d'erreur (de la 163s à 212s) 106
5.10 Heading uncertainty 107
5.11 Principe de la localisation avec découpage 112
5.12 Incertitude d'orientation 114
5.13 Incertitude moyenne en orientation pour différents facteurs de découpage 115
5.14 Incertitude moyenne en orientation pour différents facteurs de découpage en fonction de la

5.15 Temps de calcul moyen pour différents facteurs de découpage en fonction de la taille de
5.16 Consistance en orientation (les valeurs sont en degrés sur le cercle) 117
5.17 Comparaison ICPS (3σ) et l'EKF inconsistant à 3σ d'erreur . . . 119
5.18 Comparaison ICPS (3σ) et l'EKF consistant à 25σ d'erreur . . . 119

A.1 Inversion ensembliste (le problème) 128
A.2 Inversion ensembliste (la solution) 129

D.1 Figure reconnaissable ou non par la méthode Waltz (Waltz, 1975) 138
D.2 Les sommets possibles (Waltz, 1975) 138
D.3 Propagation des étiquettes (Waltz, 1975) 138

E.1 Aucun arc n'est consistant 142
E.2 $A \rightarrow B$ est consistant 142
E.3 $A \rightarrow B$ et $B \rightarrow A$ sont consistants 142
E.4 Exemple pour AC-1 144
E.5 Exemple pour AC-3 146
E.6 Exemple AC-4 146
E.7 Exemple AC-4 (deuxième partie) 146
E.8 Consistance d'arc 148
E.9 Consistance de chemin testée le long du chemin de V_0 à V_5 148

Liste des tableaux

2.1 Tableau de classification des capteurs 34

3.1 Les différents modes d'arrondi (IEEE-754) 41

4.1 Exemple de 3B-consistance 75

5.1 Caractéristiques des différentes techniques de localisation (Aldon, 2001) 88
5.2 Comparaison des algorithmes 103
5.3 Temps de calcul moyen de l'algorithme pour un pas de temps de localisation 114

Liste des algorithmes

4.1 HC4 . . . 66
4.2 HC4 Sous-programmes . . . 67
4.3 LNAR . . . 72
4.4 NEWTON . . . 73
4.5 BC4 . . . 74
4.6 3B . . . 76
4.7 3B–fixpoint . . . 77
5.1 Localisation Markovienne . . . 89
5.2 Localisation Monte Carlo . . . 91
5.3 ICP with Splitting . . . 112
5.4 ICP with Splitting . . . 113
A.1 SIVIA . . . 128
B.1 Interval Branch And Bound . . . 132
C.1 Le filtre bayesien . . . 133
C.2 Algorithme de filtre de Kalman . . . 134
C.3 Algorithme de filtre de Kalman étendu . . . 135
D.1 Waltz . . . 139
E.1 Consistance de nœuds . . . 142
E.2 REVISE . . . 143
E.3 AC-1 . . . 144
E.4 AC-3 . . . 145
E.5 INITIALIZE . . . 147
E.6 AC-4 . . . 147
G.1 HC3 . . . 155
G.2 narrowing . . . 156

Liste des acronymes

BC Box consistency : Consistance de boîte. La consistance de boîte est implémentée par les algorithmes BC3, BC4

BESE Bounded Error Set Estimation : Estimation d'ensemble à erreurs bornées

CSP Constraint Satisfaction Problem : Problème de satisfaction de contraintes

DCSP Dynamic CSP : CSP dynamique. Ce terme fait référence aux CSP auxquels on peut ajouter ou retirer des contraintes. Il s'agit d'un CSP auquel on peut à tout moment appliquer une restriction ou une relaxation.

DICSP Dynamic Interval Constraint Satisfaction Problem : Problème de satisfaction de contraintes intervalles dynamiques

EKF Extended Kalman Filter : Filtre de Kalman étendu

GPS-RTK Real Time Kinematic GPS : GPS Cinématique temps réel

HC Hull Consistency : Consistance d'enveloppe. La consistance d'enveloppe est implémentée par les algorithmes HC3 et HC4

IBBA Interval Branch And Bound Algorithm : Algorithme à intervalle basé sur la méthode de Branch&Bound

ICP Interval Constraint Propagation : Propagation de contraintes sur les intervalles

ICPS ICP with Splitting : Propagation de contraintes sur les intervalles avec découpage

ICSP Interval Constraint Satisfaction Problem : Problème de satisfaction de contraintes à intervalles

ImageSP Algorithme qui calcule l'image de sous-pavage

IMU Inertial Measurement Unit : Centrale inertielle

KF Kalman Filter : Filtre de Kalman

MCL Monte Carlo Localization : Localisation avec l'approche de Monte Carlo

SIVIA Set Inverter Via Interval Analysis : Inversion ensembliste avec de l'analyse par intervalles

Chapitre 1

Introduction générale

Sommaire

1.1 Introduction 21
1.2 Objectifs et hypothèses 22
1.3 Plan du document 23

1.1 Introduction

La localisation est une tâche de grande importance en robotique mobile et plus précisément dans la navigation autonome. Elle revêt une utilité comparable à celle de la planification, du contrôle commande ou de l'évitement d'obstacle. Elle consiste pour un mobile à estimer de façon continue ou discrète sa position et son orientation dans un référentiel bien défini, tout en se déplaçant. Il s'agit de répondre à la question "Où suis-je ?". En réalité, le terme de "localisation" peut désigner plusieurs capacités d'un mobile autonome en déplacement. Il peut s'utiliser pour le suivi de position ou pour la localisation globale. Dans le cas du suivi de position, il s'agit de calculer la position courante à partir d'une position initiale connue et des informations capteurs. La localisation globale quant à elle ne nécessite pas une connaissance d'une position précédente pour évaluer la pose courante. Elle est globale par rapport à l'environnement et donne au mobile sa position quelque soit l'emplacement.

Afin de se repérer dans l'un ou l'autre de ces cas, le mobile se base sur des sources d'information provenant des moyens de perception dont il dispose. Ainsi distingue-t-on généralement :

- des capteurs proprioceptifs tels que les odomètres, les gyromètres...
- des capteurs extéroceptifs tels que les caméras, les télémètres...

Les capteurs proprioceptifs effectuent leurs mesures en se basant sur ce qu'ils perçoivent localement sur le mobile, alors que les capteurs extéroceptifs prennent leurs mesures par rapport à l'environnement global du mobile. En opérant une fusion des informations perçues par les différents capteurs du mobile, on obtient sa position.

Aujourd'hui, l'usage d'un capteur comme le GPS est largement répandu. Dans

le cadre des véhicules ou des mobiles se déplaçant en environnement ouvert, les systèmes de positionnement par satellites sont assez fiables et efficaces. Cependant, les signaux des satellites sont parfois masqués ou subissent des phénomènes de reflet. Ceci a pour effet de fausser les mesures et d'introduire des biais dans la position calculée.

Le nombre de satellites visibles peut être insuffisant pour un calcul fiable du positionnement. Ceci justifie l'idée d'utiliser d'autres capteurs afin de procurer un niveau d'incertitude acceptable en milieu urbain.

En outre, le déplacement d'un véhicule autonome en milieu urbain peut représenter un danger. La tâche de localisation doit donc fournir une garantie sur les résultats et un niveau d'incertitude suffisamment faible.

Plusieurs méthodes sont utilisées pour résoudre ce type de problème. Elles peuvent être classées en deux grandes catégories d'approches : les approches probabilistes et les approches déterministes. Ce travail aborde les approches déterministes et plus précisément l'approche ensembliste basée sur l'analyse par intervalles.

Les méthodes ensemblistes permettent d'obtenir des résultats garantis. Ces méthodes permettent d'opérer des calculs sur des mesures ayant des erreurs bornées et dont les bornes sont connues. Elles fournissent ainsi un ensemble de solutions cohérentes avec les mesures capteurs qui contiennent avec certitude la position réelle du mobile.

1.2 Objectifs et hypothèses

L'objectif de cette thèse est de proposer des améliorations aux processus de localisation en utilisant les approches ensemblistes (analyse par intervalles).

Les méthodes ensemblistes ont été utilisées dans plusieurs domaines avec succès. Elles permettent d'obtenir des résultats de façon déterministe. Ces résultats offrent la certitude de contenir la solution aux problèmes formalisés avec ces outils. Cette certitude constitue une "garantie" sur le résultat. La contrainte principale de l'approche intervalle est que les erreurs soient bornées et que les bornes soient connues.

Parmi les méthodes intervalles, on distingue généralement celles qui utilisent la bissection des vecteurs d'intervalles (algorithme SIVIA par exemple) et celles qui caractérisent le problème sous la forme d'un problème de satisfaction de contraintes.

Plusieurs études (Vincke and Lambert, 2011) montrent que les méthodes basées sur la propagation de contraintes sont plus rapides et moins coûteuses en ressources système que celles basées sur la bissection. L'approche "propagation de contraintes", basée sur la contraction répétitive des vecteurs intervalles, est donc privilégiée.

Nos travaux ont été conduits sur des jeux de données réelles collectées en environnement d'extérieur avec plusieurs capteurs dont les données ont été fusionnées. L'approche détaillée dans ce document utilise les méthodes intervalles pour effectuer cette fusion.

Les domaines des variables capteurs considérés dans les problèmes de satisfaction de contraintes correspondent aux mesures capteurs intégrant les bornes sur les erreurs. La résolution se fait avec l'aide d'un solveur à intervalle. Nous avons comparé divers algorithmes ; un constat s'est dégagé : les algorithmes de consistance locale ne corrigent pas l'incertitude sur l'orientation. Cette thèse propose une méthode utilisable dans des applications temps-réel et qui corrige l'incertitude sur le cap du véhicule.

Nous avons aussi abordé dans cette thèse le problème de SLAM ensembliste en environnement extérieur.

Le terme "consistance" est utilisé dans ce document pour désigner deux concepts différents. Lorsqu'on traite des problèmes de satisfaction de contraintes, on parle de consistance locale ou globale. Il s'agit, dans cas, d'une solution qui respecte et satisfait une contrainte (consistance locale) ou toutes les contraintes (consistance globale) du problème. Ce terme est détaillé dans le chapitre 4. On parle aussi de consistance lorsque la boite estimée résultat du processus de localisation englobe la totalité de la position réelle du mobile. Ce concept est détaillé dans le chapitre 5.

1.3 Plan du document

Ce document est structuré en 6 chapitres.

Dans le chapitre 2, nous présentons les capteurs utilisés en robotique et les prototypes sur lesquels ont porté nos expérimentations. Une brève introduction à la notion d'intelligence des véhicules est aussi abordée dans ce chapitre. Nous présentons les capteurs utilisés généralement sur les véhicules intelligents et plus particulièrement, les capteurs qu'utilisent les véhicules dont nous avons extrait les données.

Les ressources mathématiques que sont l'analyse par intervalles et la satisfaction de contraintes seront présentées respectivement dans le chapitre 3 et 4. Le chapitre 3 est consacré à la présentation de l'analyse par intervalles. Ce chapitre aborde aussi l'intérêt de cette approche ainsi que les problèmes inhérents à l'utilisation de cet outil.

Dans le chapitre 4, un état de l'art, en matière de propagation par contraintes sur les intervalles est proposé. Les algorithmes basés sur cette méthode sont présentés. Ces algorithmes sont comparés au niveau de leur capacité à contracter les boîtes intervalles de localisation et des ressources utilisées.

Dans le chapitre 5, l'application de ces outils mathématiques au domaine de localisation de véhicule ainsi que les travaux réalisés et les résultats obtenus sont abordés. Les travaux réalisés, les expérimentations, l'algorithme proposé, ses fonctionnalités et résultats sont détaillés. Un état de l'art de la localisation de robotique basée sur la fusion de données multi-sensorielles est proposé.

Les perspectives d'évolution du travail réalisé seront présentées dans le chapitre 6.

Chapitre 2

Robotique et véhicules intelligents

Sommaire

2.1 Notion de systèmes intelligents 26
2.2 Systèmes de perception 28
2.2.1 Description des capteurs 28
2.2.2 Classification des capteurs 33
2.3 Plateforme expérimentale (CARLLA) 34
2.4 Conclusion 36



SmarTer (Lamon et al., 2006)

FIGURE 2.1 – Exemple de système intelligent

Nous allons, dans ce chapitre, situer notre travail dans le contexte des véhicules intelligents. La notion de système intelligent sera brièvement abordée à cet effet. Puis, nous présenterons les constituants des systèmes de perception généralement utilisés dans le cadre des véhicules autonomes. Pour finir, la plateforme utilisée pour la validation de notre travail sera présentée.

2.1 Notion de systèmes intelligents

Le nombre important et croissant des articles scientifiques portant sur les systèmes intelligents illustre fort bien et à juste titre l'intérêt que porte la communauté scientifique à ce sujet. La notion d'intelligence est complexe ; voici la définition qu'en donne le dictionnaire Larousse.

Intelligence : *Ensemble des fonctions mentales ayant pour objet la connaissance conceptuelle et rationnelle.*

Aptitude d'un être humain à s'adapter à une situation, à choisir des moyens d'action en fonction des circonstances.

Au vu de cette définition, peut-on attendre des systèmes dits intelligents qu'ils aient une intelligence similaire à celle des humains ? A cette question, certains spécialistes de l'intelligence artificielle comme Herbert Simon (Andresen, 2001) répondent par l'affirmatif. Cependant, cette idée n'est pas pour autant partagée par la grande majorité des scientifiques spécialistes de ce domaine. Certains considèrent même qu'il est absolument impossible de reproduire dans un système autre qu'un être humain les capacités comme la conscience, l'émotion ou toute forme d'inventivité.

Plusieurs types de systèmes intelligents existent, allant des véhicules intelligents ou systèmes de transport intelligents (STI) aux systèmes robotiques, passant par les appareils ménagers intelligents, les systèmes de diagnostic en ligne, etc.

On trouve plusieurs définitions possibles dans la littérature pour un système intelligent. Cependant, dans un cadre général, Ng (2003) donne la définition suivante : *"Un système intelligent est tout système pouvant recueillir des informations sensorielles et ayant la capacité de les traiter avec un dispositif logiciel efficace et efficient, combiné avec un ou plusieurs algorithmes (biologiquement inspirés, mathématiquement modélisés, basés sur les statistiques, etc.) pour réaliser les fonctions comme le contrôle, la gestion de ressources, le diagnostic et/ou la prise de décision en vue d'atteindre un ou plusieurs objectifs.*"

Notre travail se situe dans le cadre général des systèmes de transport intelligents (STI). La notion de STI fait référence au sens large à l'application intégrée des technologies de communication, de contrôle et de traitement de l'information au système de transport.

Elle recouvre aussi bien le véhicule, l'infrastructure, le conducteur, que l'interaction dynamique qui les caractérise et tous les modes de transport. Les systèmes de transport intelligents offrent plusieurs avantages en améliorant la prise de décisions (parfois en temps réel) tout en permettant de gagner du temps, d'économiser de l'argent et de l'énergie, de sauver des vies, et de protéger l'environnement.

La catégorie des systèmes de transport intelligents qui nous intéressent est celle des véhicules intelligents. Bishop (Bishop, 2000b,a) présente trois principaux domaines d'application des véhicules intelligents qui sont :

- Informer et avertir le conducteur (avertissement de collision),

- Contrôler partiellement le véhicule (par exemple intervenir d'urgence pour éviter un obstacle)

- Contrôler totalement le véhicule

Et les avertissements au conducteur peuvent être liés à différentes situations :

- détection de dérive de la trajectoire (avertir le conducteur d'une dérive involontaire du véhicule hors de sa trajectoire)

- collision arrière

- détection de piéton

- avertissement sur le recul du véhicule

- etc.

Ces domaines d'application ont été, par exemple, explorés sur SmartTer (Smart All Terrain) (Lamon et al., 2006) (figure 2.1). SmartTer est un véhicule autonome développé par l'EPFL, ETH Zurich et l'université de Freiburg. Equipé de 5 scanners laser, 3 caméras, un GPS différentiel, un IMU et 4 ordinateurs de bord, il est utilisé pour expérimenter de nouveaux algorithmes au sein des équipes de ces universités. Les états du système sont accessibles via le bus CAN. Il navigue et se localise en utilisant la fusion probabiliste des informations fournies par les différents capteurs.

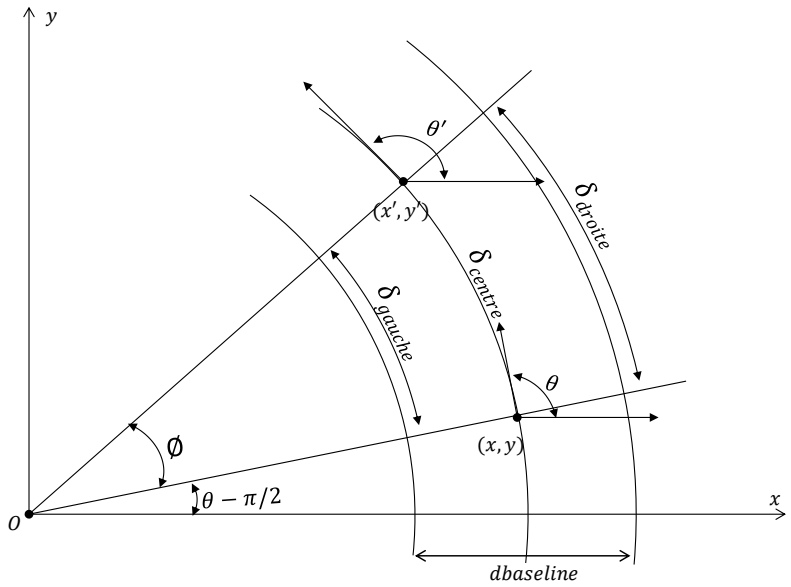


FIGURE 2.2 – Principe de fonctionnement de l'odomètre

2.2 Systèmes de perception

Les systèmes mobiles obtiennent des informations sur leur environnement par le biais de capteurs. Dans cette section, les capteurs les plus fréquemment utilisés seront présentés. Plus de détails sont donnés par Everett (1995).

2.2.1 Description des capteurs

Dans cette partie, nous présentons quelques capteurs utilisés par les véhicules autonomes.

2.2.1.1 Les odomètres

Les odomètres permettent de mesurer la rotation des roues et ainsi d'estimer les déplacements curvilignes. En général, ils ont parmi leurs composants, des codeurs incrémentaux qui leur permettent de mesurer l'angle de rotation avec une précision liée à la résolution de ces derniers. Les odomètres sont très souvent utilisés dans les applications de robotique mobile étant donné qu'ils ont l'avantage d'être peu coûteux et plutôt simples à mettre en œuvre.

Pour calculer le déplacement, certaines informations comme le diamètre des roues, la largeur de l'entraxe des roues et la cinématique du mobile sont donc requises.

Cependant, à cause des glissements, les valeurs perçues par les odomètres sont souvent imprécises. En considérant (x, y, θ) comme étant la position initiale, la position courante (x', y', θ'), après un déplacement, peut être obtenue à l'aide des formules suivantes :

$$d_{\text{centre}} = \frac{d_{\text{gauche}} + d_{\text{droite}}}{2} \tag{2.1}$$

$$\phi = \frac{d_{\text{droite}} - d_{\text{gauche}}}{d_{\text{baseline}}} \tag{2.2}$$

2.2.1.2 L'accéléromètre

Ce capteur mesure les forces externes (y compris la gravité) agissant sur le véhicule. Il s'agit d'un capteur proprioceptif qui, fixé sur un mobile, mesure l'accélération linéaire en un point donné et à un instant donné. Une double intégration des informations perçues permet d'obtenir le déplacement élémentaire du mobile. D'importantes accumulations d'erreurs interviennent généralement et sont dues à cette double intégration.

En général, on dispose d'une combinaison de 3 accéléromètres qui fournissent les 3 accélérations linéaires sur les 3 axes orthogonaux. Les accéléromètres modernes sont souvent des micro-systèmes électromécaniques (MEMS).

2.2.1.3 Gyromètre

Un gyromètre permet de mesurer une vitesse angulaire. Il fournit des informations relatives à un référentiel inertiel fixe à partir desquelles, on peut, par simple intégration, déduire le cap d'un mobile.

2.2.1.4 Boussole

Elle permet de mesurer l'angle par rapport au Nord magnétique. On parle de boussole 3D lorsque la boussole est capable de mesurer l'angle au Nord même si elle est fortement inclinée. Une boussole peut être peu fiable si l'environnement du robot ou le robot lui-même provoquent des perturbations magnétiques (à cause de pièces métalliques, courants électriques...).

2.2.1.5 Le système inertiel

Le système inertiel utilise des gyromètres et des accéléromètres pour estimer dans un repère inertiel, la position relative (x, y, z) l'orientation relative (roll, pitch, yaw), la vitesse et l'accélération d'un mobile en mouvement. Pour estimer le mouvement du mobile, le vecteur de gravité est soustrait à l'accélération et la vitesse initiale doit être connue. La mesure de l'accélération du mobile par des systèmes inertiels est souvent considérée comme la plus fiable potentiellement puisqu'elle ne dépend pas de la nature locale de l'environnement. Néanmoins, cette mesure souffre de bruit de mesure.

La figure 2.3 présente le fonctionnement général d'une IMU (Inertial Measurement Unit). Les données du gyromètre font l'objet d'une intégration pour estimer l'orientation du véhicule. On projette des accélérations sur les axes dans un repère global du véhicule en utilisant l'estimation courante de l'orientation du véhicule relative à la gravité. Le vecteur de gravité est soustrait de la mesure.

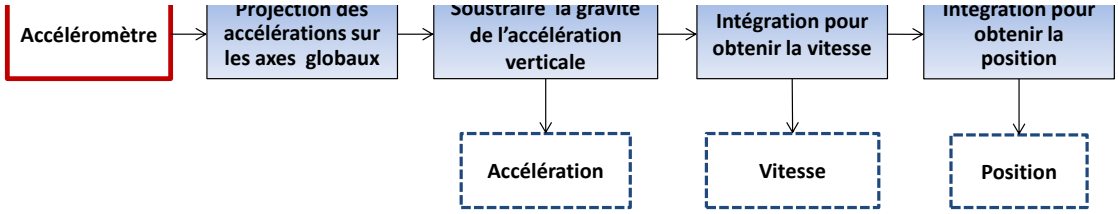


FIGURE 2.3 – Schéma de fonctionnement général d'une Centrale inertielle

L'accélération résultante est ensuite intégrée pour obtenir la vitesse qui sera à son tour intégrée pour avoir la position du mobile à partir des informations sur la vitesse initiale.

2.2.1.6 Le GPS (Global Positioning System)

Le GPS est un outil très utilisé par le grand public. Son usage va grandissant au point qu'on ne puisse plus s'imaginer s'en passer dans les années à venir. Il fournit la latitude, la longitude, l'altitude en se basant sur des satellites. Les coordonnées calculées se réfèrent au système géodésique WGS84. Une constellation de 24 satellites opérationnels depuis 1995 est utilisée. Plusieurs autres satellites ont été ajoutés à cette constellation. Les satellites orbitent à environ 20.200 km.

Le système GPS est composé de trois parties appelées segments : le segment spatial, le segment de contrôle et le segment utilisateur. Le segment spatial est constitué d'une constellation d'une trentaine de satellites NAVSTAR (31 opérationnels en 2013). Ces satellites se déplacent sur 6 plans orbitaux avec une inclinaison d'environ 56° sur l'équateur. Chaque satellite tourne 2 fois autour de la terre. Le segment de contrôle est composé de cinq stations au sol et permet de piloter et de surveiller le système. Le segment utilisateur est composé des utilisateurs de tout ordre dont le but est la réception et l'exploitation des signaux des satellites.

Le positionnement par satellite est basé sur la mesure du temps de propagation et le principe de triangulation.

Le signal GPS était, à ses débuts, entaché d'une erreur volontaire introduite par l'armée américaine qui craignait donner un avantage à ses ennemis si ces derniers disposaient d'une grande précision. Afin de contourner cette erreur volontaire qui provoquait un décalage d'une centaine de mètres, l'idée d'un GPS différentiel (DGPS) a germé. Elle consiste à mettre en place des relais de réception dont la position est connue. Ces relais appelés "bases" ont une portée pouvant atteindre 300 km. Le DGPS utilise un ensemble de "bases" (stations fixes de référence) qui fournit la différence (l'écart) entre les positions données par les satellites et leurs vraies positions déjà connues. Après mai 2000, suite à



FIGURE 2.4 – Constellation de satellites du GPS



FIGURE 2.5 – Récepteur GPS-RTK

l'annonce du gouvernement américain de libérer l'accès à une grande précision du GPS pour le grand public, la technique du DGPS a perdu quelque peu son intérêt, vu qu'elle avait été développée pour corriger spécifiquement cette erreur. Néanmoins, elle offre tout de même plus de précision que le GPS.

Pour augmenter la précision, on a recours à des satellites géostationnaires de communication pour corriger les erreurs depuis l'espace. Ils forment ainsi les systèmes nommés "SBAS" (Satellite Based Augmentation System). Plusieurs systèmes de ce genre ont vu le jour. Ainsi on dispose des systèmes suivants :

- le WAAS (Wide Area Augmentation System) pour les USA,
- le EGNOS (European Geostationary Navigation Overlay System) pour l'Europe,
- le MSAS (Multi-functional Satellite Augmentation System) pour le japon.

Ces trois systèmes SBAS sont compatibles. Ils fournissent une précision d'environ 3 m. Cependant, en vue d'avoir une précision centimétrique, une autre technique de GPS a été mise au point. Il s'agit du GPS-RTK (Real Time Kinematic). Il utilise un principe similaire à celui du DGPS c'est-à-dire qu'une station fixe de référence fournit en temps réel des corrections. Cette technique tire son atout majeur de la différence de phase qu'il utilise pour la correction. En effet, le GPS-RTK se sert de la phase de la porteuse du signal (autour de 1575,42MHz) alors que le DGPS se base sur la phase du code (voisinant 1,023MHz). Le système prend alors l'appellation de "Carrier-Phase Enhancement" ou "CPGPS". Le GPS-RTK apporte ainsi une plus grande précision.

Pour pouvoir disposer de cette technologie qui est en général coûteuse, l'utilisateur a souvent le choix entre disposer de sa propre base RTK, ou utiliser les services d'un réseau RTK disponible comme le réseau S@t-Info ou Téria par exemple.

Dans notre cas, nous avons utilisé un récepteur GPS-RTK pour déterminer la vérité terrain et un récepteur GPS Garmin AG132 pour la localisation.

2.2.1.7 Le RADAR

Le RADAR [1] permet d'obtenir une estimation de la vitesse linéaire du mobile par rapport à un objet de la scène en se basant sur l'effet Doppler.

2.2.1.8 Le LIDAR

Le LIDAR [2] fonctionne sur le même principe que le RADAR mais utilise la lumière et permet de mesurer la distance aux différents éléments de l'environnement.

2.2.1.9 La caméra

A travers les séquences d'images qu'elle fournit, une caméra peut offrir beaucoup d'informations sur l'environnement traversé par un mobile (voir par exemple (Mei and Rives, 2007)) même s'il faut reconnaître que les traitements nécessaires

1. RADAR : RAdio Detection And Ranging
2. LIDAR : LIght Detection And Ranging

à l'utilisation de ces informations sont parfois lourds à mettre en place. Les caméras ont l'avantage d'être les seuls capteurs fournissant des informations sensorielles de l'environnement très proches de celles perçues par les humains.

Lorsqu'elle est utilisée seule, une caméra fournit une information bidimensionnelle. Cependant, il est possible en calculant le flux optique entre plusieurs images de mesurer son angle ou sa distance par rapport à des amers (Rodriguez et al., 2009) voire même la vitesse. Plusieurs travaux portent sur l'utilisation d'une simple caméra pour la localisation. Cependant, afin d'améliorer les résultats ou de rendre les traitements moins lourds, on a recours à deux ou plusieurs caméras. On parle de stéréo-vision lorsqu'on dispose deux caméras orientées dans une même direction.

2.2.2 Classification des capteurs

Certains capteurs servent juste à la mesure d'informations simples comme la température interne du système électronique du mobile ou la vitesse de rotation de ses moteurs. D'autres, plus sophistiqués, permettent à un mobile de recueillir les informations sur son environnement voire sa position dans un référentiel donné.

Nous pouvons classer les capteurs suivant deux catégories. La première catégorie permet de répondre à la question "Que mesure-t-on ?" et indique si les informations perçues sont proprioceptives ou extéroceptives. La deuxième catégorie répond à "Comment le mesure-t-on ?" et qualifie le rôle passif ou actif que joue le capteur dans l'échange d'information avec son environnement.

Les capteurs proprioceptifs mesurent les valeurs internes (intrinsèques) au robot/véhicule (la vitesse du moteur, la charge de la roue, le voltage de la batterie, etc.). Les capteurs extéroceptifs mesurent, quant à eux, les informations provenant de l'environnement du mobile comme la distance à un objet, l'intensité de la lumière dans la salle qu'il traverse, l'amplitude du son dans un endroit où il se trouve.

Les capteurs passifs n'émettent pas mais ils se contentent de recueillir l'énergie de l'environnement ambiant. Dans cette catégorie, on peut retrouver les sondes de température, les microphones et les caméras CCD ou CMOS. Les capteurs actifs émettent de l'énergie dans l'environnement et mesurent la réaction des différents éléments qui y sont présents. Ils disposent donc d'un émetteur pour illuminer la scène environnante dans la bande spectrale du récepteur intégré. Ces capteurs sont connus pour fournir de meilleures performances étant donné qu'ils permettent de mieux gérer et contrôler les interactions avec l'environnement. Il convient de noter cependant, que leur utilisation n'est pas sans risque. En effet, l'énergie émise peut parfois influer sur les valeurs mesurées. Des problèmes d'interférence peuvent survenir entre les signaux émis par le capteur et ceux émis par d'autres équipements. Par exemple les signaux émis par d'autres robots ayant des capteurs semblables aux siens dans le même environnement peuvent influencer les résultats des mesures. On peut citer parmi ces capteurs, les capteurs ultrasoniques ainsi que les télémètres laser.

Le tableau 2.1 présente quelques capteurs souvent utilisés dans le domaine de la robotique mobile.

Capteur	Proprioceptif	Extéroceptif	Actif	Passif
Boussole		√		√
Gyro	√			√
Inclinomètre		√	√	√
GPS		√	√	
Accéléromètre	√			√
Odomètre	√			√
Camera CCD/CMOS		√		√
Sonar		√	√	
Télémètre ultrason		√	√	
Télémètre laser		√	√	
Télémètre infrarouge		√	√	
Radar		√	√	
Lidar		√	√	

Tableau 2.1 – Tableau de classification des capteurs

2.3 Plateforme expérimentale (CARLLA)

Le LIVIC dispose de plusieurs moyens expérimentaux constitués de véhicules instrumentés, équipés de divers dispositifs de perception, d'action et de traitement. Dans le cadre de cette thèse et pour notre expérimentation sur la localisation, nous avons utilisé des données réelles collectées sur la plateforme CARLLA (Lambert et al., 2009). CARLLA [3] (voir figure 2.6) a été réalisée afin de tester de nouvelles méthodes dans le cadre du développement des capacités du véhicule du futur. C'est le troisième d'une série de véhicules de test dont LOLA [4] et VIPER [5].

Ces trois véhicules instrumentés (voir figure 2.7) sont essentiellement utilisés pour des tâches de perception et de contrôle automatique.

Deux PCs sont embarqués à bord de CARLLA. Le premier est relié à un ensemble d'actionneurs destinés au contrôle du véhicule tandis que le second PC est relié à des capteurs dédiés aux tâches de perception. Plusieurs pistes d'essai sont aussi disponibles au LIVIC pour des tests. Et les données utilisées dans le cadre de cette thèse ont été produites sur une de ces pistes.

Cette plateforme dispose de plusieurs capteurs dont :

- une centrale inertielle IMU (Inertial Measurement Unit) Crossbow VG400 fournissant la vitesse de lacet,
- un odomètre fixé sur l'axe avant (fournit la vitesse du véhicule) et
- un encodeur qui enregistre l'angle de braquage de la roue avant.
- un récepteur GPS bas coût de marque "Trimble GPS AG132" (voir la figure 2.9) qui fournit des informations utilisées pour corriger les données de localisation.

Au cours des expérimentations, la trajectoire de référence (la vérité terrain) a été obtenue en utilisant un GPS-RTK (Thales) avec une précision proche du centimètre.

3. CARLLA : Contrôleur d'Assistance Routière Longitudinale et LAtérale
4. LOLA : LOngitudinal-LAtéral
5. VIPER : Véhicule Instrumenté pour la Perception de l'Environnement Routier



FIGURE 2.6 – Plate-forme CARLLA du LIVIC (Lambert et al., 2009)



FIGURE 2.7 – Trois véhicules instrumentés du LIVIC-IFSTTAR. De gauche à droite : VIPER, CARLLA, LOLA (Perrollaz, 2008)



FIGURE 2.8 – Centrale inertielle Crossbow VG 400

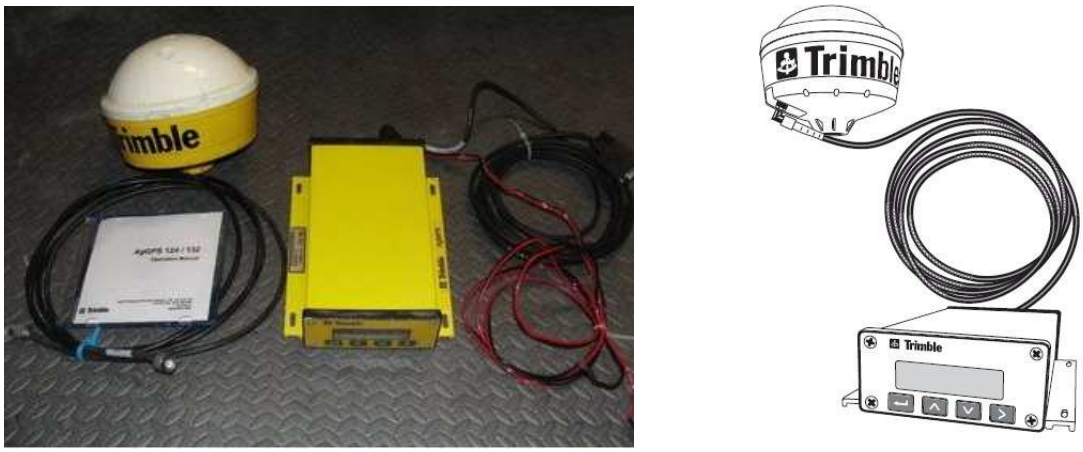


FIGURE 2.9 – Récepteur GPS Ag132 (avec antenne)

L'initialisation du processus est faite avec les mesures du GPS d'entrée de gamme (bas coût). Les données utilisées ont été collectées sur le véhicule roulant à la vitesse moyenne de 15 $m.s^{-1}$ sur 5.5 km. La piste comporte des lignes droites et de grands et petits virages.

2.4 Conclusion

Au cours de ce chapitre, nous avons introduit les véhicules intelligents puis les capteurs associés. Nous avons ensuite présenté les jeux de données (collectées sur véhicules expérimentaux) qui seront utilisées dans la suite du manuscrit. Les données de CARLLA nous permettrons de valider notre algorithme dans le cadre de la localisation de véhicules.

Chapitre 3

Analyse par intervalles

Sommaire

3.1 Introduction 38
3.1.1 Les origines 38
3.1.2 Motivations 39
3.2 Arithmétique des intervalles 42
3.2.1 Intervalles 42
3.2.2 Propriétés d'un intervalle 43
3.2.3 Fonctions élémentaires 45
3.2.4 Opérations ensemblistes 45
3.2.5 Librairies pour programmer avec les intervalles 46
3.3 Généralisation des intervalles 47
3.4 Fonctions d'inclusion 49
3.5 Sous-pavages 51
3.6 Problèmes liés à l'arithmétique des intervalles 52
3.6.1 Le problème de dépendance 52
3.6.2 L'effet enveloppe 53
3.7 Conclusion 53

3.1 Introduction

Afin d'aborder le problème de la localisation garantie, nous avons utilisé la théorie de l'analyse par intervalles. L'analyse par intervalles traite de la théorie et l'utilisation de la notion d'intervalle dans le calcul numérique et les calculs arithmétiques qui en découlent. Le concept d'analyse par intervalles consiste à calculer avec des intervalles de nombres réels plutôt qu'avec les nombres réels eux-mêmes. Elle rassemble toutes les notions permettant d'effectuer des opérations mathématiques non pas avec des valeurs réelles, mais avec des encadrements de ces valeurs réelles. Tout calcul par intervalles est un calcul garanti. Deux objectifs sont visés. Il s'agit d'une part de garantir les résultats malgré les arrondis et d'autre part, de gérer les imprécisions des paramètres physiques. Dans ce chapitre, seront présentées, les bases de l'analyse par intervalles utilisées dans les chapitres suivants.

3.1.1 Les origines

La gestion des incertitudes des données est un des problèmes auxquels les scientifiques font souvent face lors du traitement des informations. Une des solutions à ce problème consiste à prendre en compte des intervalles contenant les valeurs imprécises mesurées.

Ramon Moore a été le premier en 1962 à avoir formalisé l'analyse par intervalles dans sa thèse (Moore, 1962) puis développé de façon plus exhaustive. Il en a publié un livre (Moore, 1966) en 1966. Il a publié en janvier 1959, un rapport technique (Moore, 1959) dans lequel il explique comment implémenter l'arithmétique des intervalles sur un ordinateur.

Lorsque l'on cherche à connaître l'origine de cette méthode, on se rend vite compte que cette idée n'est pas aussi nouvelle en mathématique que l'on peut le croire. Certains scientifiques y avaient déjà pensé par le passé. Archimède avait utilisé deux bornes pour calculer le nombre π. Son article a été publié plus tard dans une édition spéciale de la presse de l'université de Cambridge "The Works of Archimedes" (Archimedes, 1897). Il convient aussi de noter qu'en 1908, l'italien W. H. Young (Young, 1908) a étudié le concept des fonctions ayant des valeurs bornées. En 1927, le russe Vladimir M. Bradis (Bradis, 1927) a explicitement formulé des règles sur l'arithmétique des intervalles (pour les quantités positives uniquement) et les a appliquées à l'évaluation des expressions rationnelles.

En 1931, le concept des opérations avec un ensemble de nombres multi-valués a été introduit par Rosalind Cecil Young (Young, 1931) qui a développé une algèbre formelle des nombres multi-valués dans sa thèse de doctorat obtenue à l'Université de Cambridge.

En 1951, P. S. Dwyer (Dwyer, 1951) a étudié le cas des intervalles fermés (positifs ou négatifs). En 1956, Mieczyslaw Warmus (Warmus, 1956) avait publié dans le Bulletin de l'Académie Polonaise de Sciences un document concernant le calcul des approximations de fonctions et de variables basées sur des intervalles. En 1958, Teruo Sunaga (Sunaga, 1958) avait abordé le concept dans son mémoire de master intitulé "Théorie de l'algèbre des intervalles" rédigé en japonais.

Jusqu'aux années 70, cette théorie est restée méconnue du public. Elle a été rendue populaire dans les années 1980 au sein des universités allemandes (Université de Karlsruhe, Université de Hamburg, ...) grâce à un travail de formation et d'information du professeur Ulrich Kulisch et de ses doctorants. L'analyse par intervalles s'est répandue avec la publication, en 1985, de la norme IEEE-754 autour des arrondis dirigés pour les nombres à virgule flottante. Plusieurs algorithmes utilisant les intervalles ont été proposés à partir des années 90. Parmi ces algorithmes, on peut noter l'algorithme de Hansen pour la résolution des problèmes d'optimisation globale, l'algorithme de Newton par intervalles pour localiser les zéros d'une fonction, etc. La norme IEEE-1788 spécifiant l'arithmétique par intervalles est (à ce jour) encours de discussion dans un groupe de travail international dédié.

Les domaines d'utilisation de l'analyse par intervalles sont de plus en plus larges et mêlent le calcul scientifique à la rigueur mathématique. Cette association permet de résoudre des problèmes d'incertitude qui ne peuvent être efficacement résolus avec l'arithmétique des nombres flottants.

Les champs d'application des méthodes par intervalles recouvrent aujourd'hui aussi bien la physique expérimentale, la théorie du contrôle, la preuve assistée par ordinateur, l'infographie, la robotique, le traitement du signal, le génie électrique que la télédétection.

Par exemple, Vigneron (2007) a utilisé les méthodes à intervalles pour résoudre différents types de problèmes de statistiques. L'estimation de paramètres d'un modèle linéaire (ou non) est traitée comme un problème d'inversion ensembliste. Vinas et al. (2004) les ont appliquées au contrôle robotique.

3.1.2 Motivations

3.1.2.1 La gestion des nombres à virgule flottante

Rump (Rump, 1988) a présenté une fonction illustrant clairement les divergences entre résultats réels et résultats de calculs en flottants. Soit la fonction f définie de $\mathbb{R}^2$ vers $\mathbb{R}$:

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y} \tag{3.1}$$

Sur un ordinateur IBM S/370 en calculant avec une précision simple, double et étendue (approximativement 7, 16 et 33 chiffres décimaux respectivement), Rump a pu obtenir (pour $x = 77617.0$ et $y = 33096.0$) les résultats suivants :

Simple précision : $f \simeq +1.172603...$

Double précision : $f \simeq +1.172603940053178...$

Précision étendue : $f \simeq +1.17260394005317863185...$

Ces résultats laissent à penser que la valeur exacte serait approximativement 1.172603 ou même 1.1726039400532. Mais il n'en est rien. En réalité, la vraie valeur du résultat est :

$$f \simeq -0.827396059946821368141165095479816... \tag{3.2}$$

$$f \simeq -0.827396059946821_3^4 \tag{3.3}$$

Ceci montre que même en augmentant la précision des résultats, on ne s'approche pas forcément de la vraie solution. Les résultats de Rump sont des nombres positifs alors que le vrai résultat est de signe négatif. Ainsi, même le signe n'est pas exact.

Nous avons aussi évalué cette expression sur une machine moderne (64bit) et nous avons obtenu :

$$f \simeq -1.180591620E + 21$$

Ce résultat se rapproche de ceux obtenus par d'autres chercheurs. En effet, Loh et Walster (Loh and Walster, 2002) ont aussi évalué cette expression sur les ordinateurs qui implémentent l'arithmétique de la norme IEEE-754. Notamment, ils ont obtenu en simple, double et quadruple précision, dans un compilateur Forte Developper 6 Fortran 95 sous Sun Microsystems les résultats suivants :

32-bit : $f = -6.338253E + 29$
64-bit : $f = -1.1805916207174113E + 21$
128-bit : $f = +1.1726039400531786318588349045201838$

Outre le fait que ces résultats se révèlent incorrects, on s'aperçoit en augmentant la précision, que le calcul est instable. En résumé, l'exemple de Rump 3.1 est toujours valable même sur les ordinateurs modernes.

Loh and Walster (2002) ont remarqué que les valeurs de x et y choisies par Rump satisfont la relation suivante :

$$x^2 = 5.5y^2 + 1. \tag{3.4}$$

En d'autres termes $77617^2 = 5.5 \times 33096^2 + 1$ et en remplaçant, dans la fonction 3.1, x^2 par son équivalent de 3.4, les termes contenant des x^2 et y^2 s'annulent et la fonction peut être réécrite sous la forme :

$$f(x, y) = 5.5y^8 - 2 - 5.5y^8 + \frac{x}{2y} \tag{3.5}$$

et finalement :

$$f(x, y) = \frac{x}{2y} - 2 \tag{3.6}$$

Le résultat se calcule plus aisément sur n'importe quel calculateur. Voici le résultat du calcul avec la fonction 3.6 :

$$f(77617.0,\ 33096.0) = -0.827396059946821368141165095479816... \tag{3.7}$$

En évaluant les expressions avec l'arithmétique des intervalles, on obtient des intervalles qui contiennent le résultat correct et montrent aussi l'instabilité à travers leur largeur. Sous INTLAB [6] (Rump, 1999), on obtient les résultats suivants :

– Avec une précision de 10 :

6. une boîte à outils de calcul par intervalles fonctionnant sous Matlab

Représentation	Description
$\lfloor x \rfloor$	arrondi vers $-\infty$
$\lceil x \rceil$	arrondi vers $+\infty$
$[x]_n$	arrondi au plus près
$[x]_0$	arrondi vers 0

Tableau 3.1 – Les différents modes d'arrondi (IEEE-754)

$$f(77617.0,\ 33096.0) = 1.0e + 014 \times [-6.72351402328065, 5.31613913972737]$$

– Avec une précision de 20 :

$$f(77617.0,\ 33096.0) = 1.0e + 014 \times [-6.72351402328065, 5.31613913972737]$$

– Avec une précision de 30 :

$$f(77617.0,\ 33096.0) = 1.0e + 007 \times [-0.04587540000001, 1.72359720000001]$$

– Avec une précision de 100 :

$$f(77617.0,\ 33096.0) = [-0.82739605994683, -0.82739605994682]$$

En utilisant l'arithmétique des intervalles, plus grande est la précision, plus petit est l'intervalle résultat.

L'intervalle $[\underline{x}, \overline{x}]$ ne peut être représenté sur machine que si $\underline{x}$ et $\overline{x}$ sont représentables en machine. Or ces deux nombres ont besoin d'être arrondis après chaque opération et le mode d'arrondi par défaut est l'arrondi au plus près (plus proche). Afin d'éviter des pertes d'information, l'hypothèse de l'arithmétique des intervalles veut que $\underline{x}$ soit arrondi vers le bas et $\overline{x}$ arrondi vers le haut. On parle d'arrondis dirigés.

La norme IEEE 754 pour l'arithmétique des nombres à virgule flottante dispose de quatre modes d'arrondi :

– l'arrondi au plus près,
– l'arrondi vers le haut (vers plus l'infini),
– l'arrondi vers le bas (vers moins l'infini),
– l'arrondi vers zéro

Le tableau 3.1 présente les différents modes d'arrondi. Ceci a donc rendu possible l'utilisation de l'arithmétique des intervalles sur tous les types d'ordinateurs.

Cet exemple montre qu'il est difficile d'obtenir des résultats sans erreurs d'arrondi lorsqu'un seul mode d'arrondi est appliqué sur le résultat du calcul. Cependant, le fait que la norme IEEE-754 offre la possibilité d'utiliser 4 modes d'arrondi, permet l'usage de l'arrondi dirigé et partant facilite le développement des applications basées sur l'analyse par intervalles. L'analyse par intervalles permet en utilisant l'arrondi vers $-\infty$ pour la borne inférieure et l'arrondi vers $+\infty$ pour la borne supérieure de pouvoir approximer de façon sûre une variable incertaine.

3.1.2.2 Comparaison avec les approches probabilistes.

Les opérations arithmétiques effectuées sur ordinateur considèrent souvent les valeurs en entrée comme des valeurs exactes. Mais, les valeurs souvent manipulées dans le domaine scientifique proviennent des mesures capteurs. Or les capteurs sont sujets à des imprécisions. Ces mesures (capteurs) ne sont donc jamais précises : il n'est pas possible de réduire le bruit de mesure à 0. Pour cette raison, la valeur qui est mesurée n'est pas identique à la valeur exacte de la quantité mesurée. Les incertitudes de mesure font l'objet de plusieurs études.

Les approches probabilistes permettant de résoudre ce genre de problème sont souvent basées sur les travaux réalisés par Gauss depuis le 19ème siècle. Elles utilisent des techniques statistiques considérant que les distributions de probabilités des erreurs de mesures sont connues. On suppose que la distribution est gaussienne (de moyenne 0) et l'écart-type est connu.

Il existe aussi des situations réelles où l'on n'a aucune connaissance sur des distributions de probabilité des erreurs de mesure. Et on ne connaît que, les bornes inférieures et supérieures atteignables par cette incertitude ou la borne maximale sur la valeur absolue de cette erreur de mesure. Dans ces cas, on peut facilement déduire des intervalles auxquels chaque valeur mesurée appartient de façon sure. Les approches à intervalles sont particulièrement adaptées à ces cas.

Avec les méthodes ensemblistes, il n'est pas nécessaire de linéariser les équations. On ne suppose pas que le bruit est additif. On n'envisage pas d'approximer les fonctions de densité de probabilité, comme c'est le cas avec la plupart des smoothers (lisseurs) Bayésiens.

Toutes les solutions consistantes avec les données et hypothèses ou suppositions (sur le modèle et les bornes des erreurs) sont trouvées. Aucune n'est perdue. Dans le cas d'un véhicule, ou d'un robot, dans un contexte non linéaire, les méthodes ensemblistes sont les seules à pouvoir fournir une enveloppe contenant toutes les trajectoires possibles. En plus, elles sont robustes par rapport aux données aberrantes.

Une autre raison (Moore et al., 2009) d'opter pour les approches à intervalles est qu'elles permettent d'obtenir avec une certitude absolue les résultats exacts de plusieurs problèmes mathématiques. En effet, l'analyse par intervalles permet de manipuler des ensembles de réels. Les vecteurs d'intervalles donnent des ensembles d'espace de dimension élevée. Elles fournissent des encadrements rigoureux. Et dans ce sens, on peut savoir à l'avance, si un modèle proposé est réaliste ou pas. Sans des bornes rigoureuses sur les erreurs de calcul, une comparaison des résultats numériques avec les mesures physiques ne nous dit pas si un modèle mathématique est réaliste ou pas.

3.2 Arithmétique des intervalles

3.2.1 Intervalles

Un intervalle x est un sous-ensemble "fermé" et "connexe" de $\mathbb{R}$:

$$x = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leq x \leq \overline{x}\},$$

où $\underline{x}$ représente la borne inférieure et $\overline{x}$, la borne supérieure. $[0, 5]$, $\{1\}$, $[-\infty, 3]$, $\mathbb{R}$, $\emptyset$ sont des exemples d'intervalles valides tandis que $]1, 3[$ et $[1, 2] \cup [3, 4]$ sont des contre-exemples. On peut aussi représenter un intervalle à partir de son milieu et son rayon. L'ensemble des intervalles de $\mathbb{R}$ est noté $\mathbb{IR}$ avec :

$$\mathbb{IR} = \{[\underline{x}, \overline{x}] : \underline{x}, \overline{x} \in \mathbb{R} : \underline{x} \leq \overline{x}\}$$

L'intervalle revêt une double nature (Moore et al., 2009). En effet, il est traité aussi bien comme un couple de nombres constitué par ses bornes (deux nombres sont ainsi manipulés) et comme un ensemble de nombres compris entre les deux bornes. Pour cette raison, on utilise des opérations ensemblistes aussi pour traiter les intervalles et résoudre des problèmes. L'arrondi dirigé (vers l'extérieur) permet de garantir que les encadrements soient rigoureux. Dans le contexte de la localisation de véhicule, la position en x sera représentée par un intervalle $[x]$.

3.2.2 Propriétés d'un intervalle

Plusieurs propriétés permettent de caractériser un intervalle. Ainsi la taille (aussi appelée, "longueur" ou "largeur") d'un intervalle notée w vaut :

$$w(x) = w([\underline{x}, \overline{x}]) = \overline{x} - \underline{x} \tag{3.8}$$

Lorsque $w(x)$ est nulle, l'intervalle est dit "dégénéré", "scalaire" ou "singleton". Avec $x = [0, 2]$ et $y = [-1, 1]$, $w(x) = w(y) = 2$.

L'amplitude (ou "magnitude" ou "absolute value") d'un intervalle vaut :

$$mag(x) = max(|\underline{x}|, |\overline{x}|) \tag{3.9}$$

Avec $x = [0, 2]$ et $y = [-1, 1]$, $mag(x) = max(0, 2) = 2$. La valeur du milieu $m(x)$ d'un intervalle se calcule de la façon suivante :

$$m(x) = m([\underline{x}, \overline{x}]) = \frac{\underline{x} + \overline{x}}{2} \tag{3.10}$$

Par exemple, pour $x = [0, 2]$ et $y = [-1, 1]$, on a $m(x) = 1$ et $m(y) = 0$. Toute une arithmétique a été développée pour le calcul par intervalles. Ainsi, les opérations usuelles telles que l'addition, la soustraction, la multiplication, la division, etc. sont transposées à l'analyse par intervalles. Ainsi a-t-on les opérateurs suivants :

Addition :

$$[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$$

Soustraction :

$$[x] - [y] = [\underline{x}, \overline{x}] + [-\overline{y}, -\underline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$$

Multiplication :

$$[x] \times [y] = [min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}),\ max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})]$$

Inverse :

$$\frac{1}{[x]} = \begin{cases} [-\infty, +\infty]\ si\ [\underline{x}, \overline{x}] = [0, 0] \\ [\frac{1}{\overline{x}}, \frac{1}{\underline{x}}]\ si\ 0 \notin [\underline{x}, \overline{x}] \\ [\frac{1}{\overline{x}}, +\infty]\ si\ \underline{x} = 0\ et\ \overline{x} > 0 \\ [-\infty, \frac{1}{\underline{x}}]\ si\ \underline{x} < 0\ et\ \overline{x} = 0 \\ [-\infty, +\infty]\ si\ \underline{x} < 0\ et\ \overline{x} > 0 \end{cases}$$

Division :

$$[x]/[y] = [x] \times \frac{1}{[y]} = [\underline{x}, \overline{x}] \times [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}]$$

Puissance :

$$[x]^n = \begin{cases} [1, 1] & si\ n = 0 \\ [0, max(\underline{x}^n, \overline{x}^n)] & si\ n = pair\ \ ET\ \ 0 \in [\underline{x}, \overline{x}] \\ [min(\underline{x}^n, \overline{x}^n), max(\underline{x}^n, \overline{x}^n)] & sinon \end{cases}$$

L'arithmétique des intervalles dispose aussi de quelques propriétés algébriques spécifiques.

- La soustraction d'un intervalle par lui-même n'est pas égale à 0.
  Par exemple, si $x = [2, 3]$, $x - x = [2, 3] - [2, 3] = [-1, 1] \neq 0$
- La division d'un intervalle par lui-même n'est pas égale à 1.
  Par exemple, si $x = [2, 3]$, $x/x = [2, 3]/[2, 3] = [\frac{2}{3}, \frac{3}{2}] \neq 1$
- La multiplication d'un intervalle par lui-même n'est pas égale à l'élévation au carré.
  Par exemple, si $x = [-3, 2]$, $x \times x = [-3, 2] \times [-3, 2] = [-6, 9]$ alors que $x^2 = [0, 9]$
- L'addition et la multiplication sont associatives et commutatives
- La multiplication n'est pas distributive par rapport à l'addition.
  Par exemple, considérons $x = [-2, 3]$, $y = [1, 4]$ et $z = [-2, 1]$.

$$\begin{aligned} \text{x} \times (\text{y} + \text{z}) &= [-2,\ 3] \times ([1,\ 4] + [-2,\ 1]) \\ &= [-2,\ 3] \times [-1,\ 5] \\ &= [-10,\ 15] \end{aligned}$$

$$\begin{aligned} \text{x} \times \text{y} + \text{x} \times \text{z} &= [-2,\ 3] \times [1,\ 4] + [-2,\ 3] \times [-2,\ 1] \\ &= [-8,\ 12] + [-6,\ 4] \\ &= [-14,\ 16] \end{aligned}$$

$\Rightarrow$ la multiplication est donc sous-distributive par rapport à l'addition, $x \times (y + z) \subseteq x \times y + x \times z$

3.2.3 Fonctions élémentaires

Par fonction élémentaire, nous faisons, ici, référence aux extensions intervalles des fonctions élémentaires des nombres réels. Soit f est une fonction de $\mathbb{R}$ vers $\mathbb{R}$, telle que $f \in \{cos, sin, sqr, sqrt, log, exp, ...\}$ on définit son extension comme suit :

$$f([x]) \triangleq [\{f(x)|x \in [x]\}]$$

Exemples de fonctions :

$sin([0, \pi]) = [0, 1]$,

$sqr([-1, 3]) = [-1, 3]^2 = [0, 9]$,

$sqrt([-10, 4]) = \sqrt{[\text{-}10, 4]} = [0, 2]$

3.2.4 Opérations ensemblistes

Rappelons dans cette section, quelques opérateurs ensemblistes qui sont aussi utilisées en arithmétique des intervalles :

Union : L'union Z de deux ensembles X et Y notée $X \cup Y$ contient tous les objets qui sont éléments d'au moins un de ces ensembles.

$$Z = X \cup Y \Leftrightarrow ((z \in X) \vee (z \in Y) \Leftrightarrow (z \in Z)) \tag{3.11}$$

$$\begin{aligned} X \cup Y &= Y \cup X \\ X \cup \emptyset &= X \\ X \cup X &= X \\ X &\subseteq X \cup Y \end{aligned}$$

En appliquant ces propriétés aux intervalles $[x]$ et $[y]$, on obtient :

$$[x] \cup [y] = \begin{cases} \{z : z \in x \ ou \ z \in y\} \\ [min\{\underline{x}, \underline{y}\}, \ max\{\overline{x}, \overline{y}\}] \end{cases} \tag{3.12}$$

Intersection : L'intersection Z de deux ensembles X et Y notée $X \cap Y$ contient tous les objets qui sont éléments des deux ensembles. Si $X \cap Y = \emptyset$ alors, X et Y n'ont aucun élément en commun. Ils sont dits disjoints.

$$Z = X \cap Y \Leftrightarrow ((z \in X) \wedge (z \in Y) \Leftrightarrow (z \in Z)) \tag{3.13}$$

$$\begin{aligned} X \cap Y &= Y \cap X \\ X \cap \emptyset &= \emptyset \\ X \cap X &= X \\ X \cap Y &\subseteq X \end{aligned}$$

En appliquant ces propriétés aux intervalles $[x]$ et $[y]$, on obtient :

$$[x] \cap [y] = \begin{cases} \emptyset \; si \; \overline{y} < \underline{x} \; ou \; \overline{x} < \underline{y} \\ \{z : z \in x \, et \, z \in y\} \\ [max\{\underline{x}, \underline{y}\}, \; min\{\overline{x}, \overline{y}\}] \end{cases} \tag{3.14}$$

Différence : La différence Z entre deux ensembles X et Y notée $X \backslash Y$ contient les objets qui sont éléments de X mais pas de Y.

$$Z = X \backslash Y \Leftrightarrow ((z \in X) \wedge (z \notin Y) \Leftrightarrow (z \in Z)) \tag{3.15}$$

$$\begin{array}{rl} X \backslash Y & = X \\ \emptyset \backslash X & = \emptyset \\ X \backslash X & = X \\ X \backslash X & \subseteq X \end{array}$$

Avec les intervalles $[x]$ et $[y]$, on obtient :

$$[x] \backslash [y] = \{z | z \in [x] \; et \; z \notin [y]\} \tag{3.16}$$

Produit cartésien de deux ensembles : Le produit cartésien Z de deux intervalles X et Y noté $X \times Y$, c'est l'ensemble de tous les couples ordonnés (x, y) dont la première composante est un élément de X et la seconde est un élément de Y.

$$Z = X \times Y \Leftrightarrow Z = \{(x, y) | (x \in X) \wedge (y \in Y)\} \tag{3.17}$$

$$[x] \times [y] = \{(x, y) | (x \in [x]) \wedge (y \in [y])\} \tag{3.18}$$

Enveloppe d'intervalle (hull) : L'enveloppe d'intervalle d'un sous-ensemble $\mathbb{X}$ de $\mathbb{R}$ est le plus petit intervalle contenant $\mathbb{X}$ et noté $[\mathbb{X}]$

$$[x] \underline{\cup} [y] = [min\{\underline{x}, \underline{y}\}, \; max\{\overline{x}, \overline{y}\}] \tag{3.19}$$

Considérons, par exemple $[x] = [-1, 0]$ et $[y] = [1, 2]$, alors on obtient $[x] \underline{\cup} [y] = [-1, 2]$. Or $x \cup y = [-1, 0] \cup [1, 2]$ (un ensemble non connexe n'est pas un intervalle).

3.2.5 Librairies pour programmer avec les intervalles

Un effort important est fait pour rendre accessible l'analyse par intervalles sur plusieurs plateformes. Il existe un large éventail de librairies qui implantent les opérations de base de l'arithmétique des intervalles.

En C/C++ et Fortran, les librairies comme Profil/Bias, C-XSC, IntLib, GlobSol, FILIB++ permettent de faire du calcul par intervalles. intpak fonctionne sous

Maple. Une librairie intégrée à Mathematica permet de faire des opérations de l'arithmétique des intervalles aussi. Sous Matlab, on dispose des outils comme INTLAB(Rump, 1999) et b4m (basé sur Bias). Int4Sci a été développé pour les utilisateurs de Scilab et Pyinterval pour les utilisateurs de Python.

La plus populaire aujourd'hui reste C-XSC, suivie par Profil/Bias et FI-LIB++. MPFI (développée par Nathalie Revol) permet une précision étendue. Toutes ces librairies ont en commun le fait de fonctionner dans $\mathbb{R}$ et traitent les opérations arithmétiques tout en implémentant les fonctions élémentaires. Elles utilisent l'arrondi dirigé (par défaut à gauche et par excès à droite). Cependant, chaque librairie utilise un format de représentation spécifique. Ceci justifie la nécessité de l'élaboration d'une norme.

3.3 Généralisation des intervalles

Pour pouvoir manipuler les intervalles à un niveau plus élevé, on a élaboré des formes généralisées des intervalles telles que les vecteurs, les matrices ou encore les tubes intervalles. Dans le cas du problème de localisation traité dans ce document, la pose (position et orientation) d'un véhicule sera représenté par un vecteur intervalle $([x],[y],[\theta])^T$. Nous présentons la notion de vecteurs d'intervalles.

Un *pavé* ou *vecteur d'intervalle* (ou encore *boîte*) $[x]$ de $\mathbb{R}^n$ est le produit cartésien de n intervalles :

$$[x] = [\underline{x_1}, \overline{x_1}] \times [\underline{x_2}, \overline{x_2}] \times \cdots \times [\underline{x_n}, \overline{x_n}] = [x_1] \times [x_2] \times \cdots \times [x_n]$$

Il s'agit d'un vecteur (voir figure 3.1 et 3.2) ayant des intervalles pour composants. Il peut donc être représenté aussi sous la forme suivante :

$$\begin{bmatrix} [\underline{x_1}, \overline{x_1}] \\ [\underline{x_2}, \overline{x_2}] \\ \cdots \\ [\underline{x_n}, \overline{x_n}] \end{bmatrix} \text{ ou } ([\underline{x_1}, \overline{x_1}], [\underline{x_2}, \overline{x_2}], \cdots, [\underline{x_n}, \overline{x_n}])$$

Un pavé à une dimension est un intervalle. $\mathbb{IR}^n$ est l'ensemble des pavés de $\mathbb{R}$. La *taille* d'un pavé $[x]$ vaut :

$$w([x]) = \max_{i \in \{1,\ldots,n\}} w([x_i]) \tag{3.20}$$

Exemple :

$$w([1,2] \times [-1,3]) = max(2-1, 3+1) = 4$$

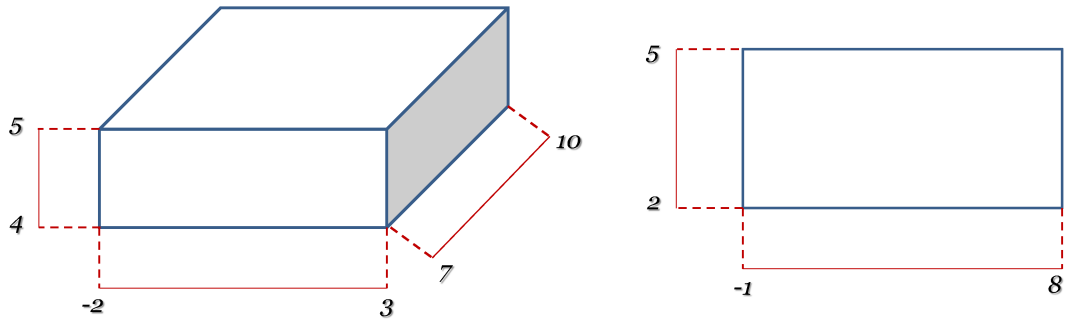
Par convention, $w(\emptyset) = -\infty$. Si $w([x]) = 0$, $[x]$ est dit *dégénéré* et est un singleton de $\mathbb{R}^n$.

La norme (ou magnitude) d'un pavé $[X] = ([x_1], ..., [x_n])$ est la plus grande des normes de ses composantes :

$$||[x]|| = \max_i |[x_i]| \tag{3.21}$$

Avec le pavé $[1,2] \times [4,7]$, on obtient :

$$m([x]) = (\frac{1+2}{2}, \frac{4+7}{2}) = (\frac{3}{2}, \frac{11}{2})$$



Pavé à 3 dimensions *Pavé à 2 dimensions*

FIGURE 3.1 – Quelques types de vecteurs d'intervalles

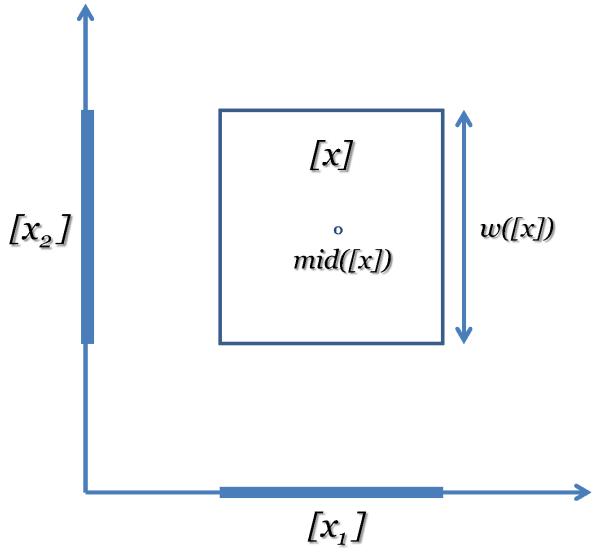


FIGURE 3.2 – *Pavé* $[x]=([x_1],[x_2])^T$

Par définition, on reconnaît aussi les propriétés suivantes :
- La borne inférieure d'un pavé $[x]$: $inf([x]) = (\underline{x}_1, \underline{x}_2, ..., \underline{x}_n)^T$
- La borne supérieure d'un pavé $[x]$: $sup([x]) = (\overline{x}_1, \overline{x}_2, ..., \overline{x}_n)^T$
- Le milieu d'un pavé $[x]$: $m([x]) = (m([x_1]), m([x_2]), ..., m([x_n]))^T$

Le plan principal : La notion de plan principal nous servira à comprendre la notion de bissection. Le plan principal d'un pavé $[x]$ est le plan symétrique et perpendiculaire à son côté le plus grand.

Bissection d'un pavé : Bissecter un pavé consiste à le découper en deux parties séparées par le plan principal. Par exemple, la bissection de $[x] = [1, 2] \times [-1, 3]$ donne les deux pavés $[x]_{(1)} = [1, 2] \times [-1, 1]$ et $[x]_{(2)} = [1, 2] \times [1, 3]$. De manière générale, la bissection d'un pavé $[x]$ de $\mathbb{R}^n$ permet d'obtenir les pavés :

$$L[x] = [\underline{x}_1, \overline{x}_1] \times ... \times \left[\underline{x}_j, \frac{\underline{x}_j + \overline{x}_j}{2}\right] \times ... \times [\underline{x}_n, \overline{x}_n] \tag{3.22}$$

$$R[x] = [\underline{x}_1, \overline{x}_1] \times ... \times \left[\frac{\underline{x}_j + \overline{x}_j}{2}, \underline{x}_j\right] \times ... \times [\underline{x}_n, \overline{x}_n] \tag{3.23}$$

où j représente l'indice du plus grand côté de $[x]$.

3.4 Fonctions d'inclusion

Les fonctions d'inclusion constituent un concept essentiel dans l'arithmétique des intervalles. De manière générale, elles sont définies comme suit : soit f une fonction de $\mathbb{R}^n$ vers $\mathbb{R}^m$. La fonction $[f]$ de $\mathbb{IR}^n$ dans $\mathbb{IR}^m$, est une fonction d'inclusion de f si :

$$\forall\, [x] \in \mathbb{IR}^n, \quad f([x]) \subset [f]([x]) \tag{3.24}$$

$[f]$ correspond à une boîte englobante. Les fonctions d'inclusion ont des propriétés diverses telles que la minimalité, la monotonicité, la convergence, etc.

$[f]$ est dite minimale si :

$$\forall\ [x] \in \mathbb{IR}^n,\ [f]([x]) = [f([x])] \tag{3.25}$$

Dans la figure 3.3, $[f]$ et $[f]^*$ sont des fonctions d'inclusion. $[f]^*$ est une fonction d'inclusion est minimale. La fonction d'inclusion $[f]$ est " monotone " si :

$$([x] \subset [y]) \Rightarrow ([f]([x]) \subset [f]([y])) \tag{3.26}$$

Elle est dite " convergente " si :

$$\lim_{k\to\infty} w([x]_{(k)}) = 0 \Rightarrow \lim_{k\to\infty} w([f]([x]_{(k)})) = 0 \tag{3.27}$$

La figure 3.4 montre une fonction d'inclusion convergente et monotone. On peut créer une fonction d'inclusion de plusieurs manières. Une des méthodes est la fonction d'inclusion naturelle. Considérons une fonction f définie par :

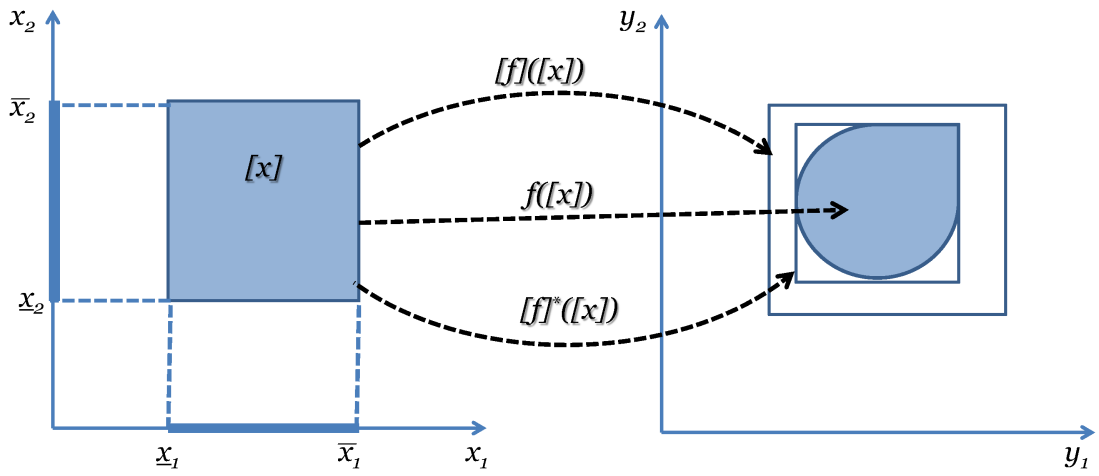


FIGURE 3.3 – Fonction d'inclusion

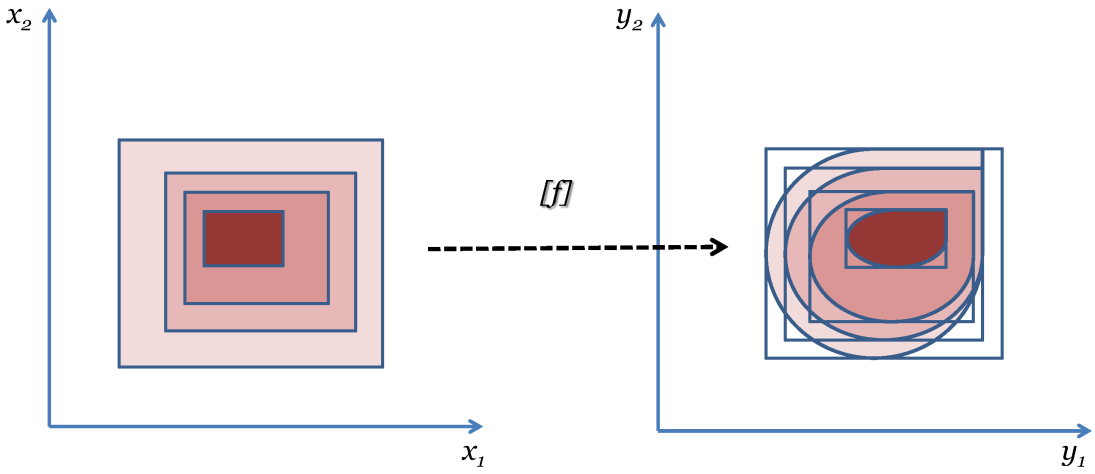


FIGURE 3.4 – Convergence et monotonie d'une fonction d'inclusion

$$\begin{array}{c} f : \mathbb{IR}^n \to \mathbb{IR}^m \\ (x_1, x_2, ..., x_n) \mapsto f(x_1, x_2, ..., x_n) \end{array} \tag{3.28}$$

La fonction d'inclusion naturelle $[f]$ est obtenue en remplaçant chaque variable réelle x_i par une variable intervalle $[x_i]$ et chaque opérateur ou fonction par son équivalent intervalle. Concrètement, une fonction d'inclusion pour $f(x) = x^2 + 2x + 4$ est $[f]([x]) = [x]^2 + 2[x] + 4$. Si $[x] = [-3, 4]$, nous obtenons :

$$\begin{aligned} [f]([-3,4]) &= [-3,4]^2 + 2[-3,4] + 4 \\ &= [0,16] + [-6,8] + 4 \\ &= [-2,28] \end{aligned}$$

Notons que $f([-3, 4]) = [f(-3), f(4)] = [7, 28]$ qui est un sous-ensemble de $[f]([-3, 4]) = [-2, 28]$.

Plusieurs travaux (Alander, 1985), (Tóth and Csendes, 2005), (Tóth et al., 2007) montrent que suivant la manière dont la fonction d'inclusion est écrite, on peut obtenir un résultat plus ou moins précis lors des calculs.

Voici un exemple : considérons la fonction $f : \mathbb{R} \to \mathbb{R}$ telle que $f(x) = x(x+1)$, on peut écrire les différentes fonctions d'inclusion suivantes :

$[f_1]([x]) = [x]([x] + 1)$
$[f_2]([x]) = [x] \times [x] + [x]$
$[f_3]([x]) = [x]^2 + [x]$
$[f_4]([x]) = ([x] + \frac{1}{2})^2 - \frac{1}{4}$

Pour $[x] = [-1, 1]$, on obtient les intervalles suivants :

$[f_1]([x]) = [-2, 2]$
$[f_2]([x]) = [-2, 2]$
$[f_3]([x]) = [-1, 2]$
$[f_4]([x]) = [-\frac{1}{4}, 2]$

Il convient de noter qu'il n'existe pas de méthode universellement meilleure.

3.5 Sous-pavages

Un *sous-pavage* de $\mathbb{R}^n$ est un ensemble de pavés $\mathbb{R}^n$ ne se recouvrant pas. La figure 3.5 montre des exemples de sous-pavages à différentes résolutions et qui représentent l'ensemble $\mathbb{X}$ tel que :

$$\mathbb{X} = \{(x_1, x_2) \,|\, x_1^2 + x_2^2 \in [1, 2]\}$$

On peut approximer les opérations sur les ensembles telles que $\mathbb{Z} = \mathbb{X} + \mathbb{Y}$, $\mathbb{X} = f^{-1}(\mathbb{Y})$, $\mathbb{Z} = \mathbb{X} \cap \mathbb{Y}$ en utilisant des opérations sur les sous-pavages. Un sous-pavage est dit "régulier" si chacun de ses pavés peut être obtenu par une succession de bissections et sélections. Il peut être représenté sous la forme d'un arbre binaire.

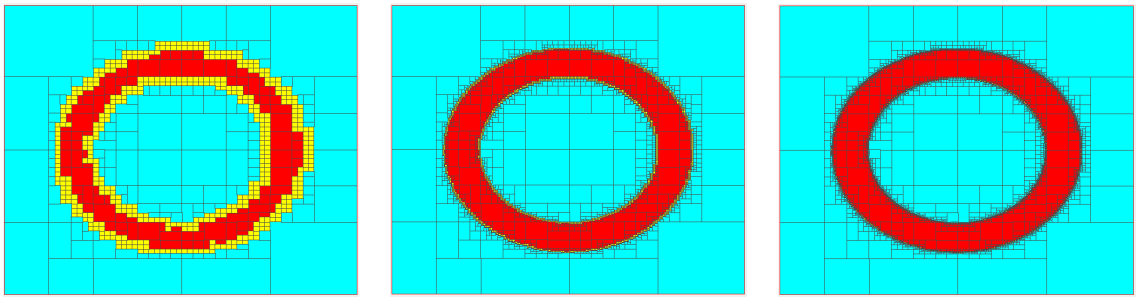


FIGURE 3.5 – Exemples de sous-pavages à différentes résolutions

3.6 Problèmes liés à l'arithmétique des intervalles

Le calcul par intervalles comporte des problèmes intrinsèquement liés à la manipulation des intervalles. Ces lacunes conduisent à surestimer les résultats des calculs. Le résultat issu d'une série d'opérations peut ne pas être minimal. Ce résultat est alors dit pessimiste. Les deux limitations suivantes sont bien connues pour être liées aux cas de pessimisme :

- la dépendance,
- l'effet enveloppe.

3.6.1 Le problème de dépendance

Le problème de dépendance (Hansen and Walster, 2003; Hansen, 1992) est lié à la multiplicité des occurrences d'une même variable au sein d'une expression. En effet, lors de la procédure d'évaluation d'une expression, il n'y a pas de corrélation entre les différentes occurrences de la même variable. Les occurrences d'une même variable fournissent juste des intervalles de valeurs identiques mais qui ne sont pas traitées comme la même variable. Soit l'intervalle $[x]$ et l'opération intervalle notée $\circ$. Effectuer l'opération $\circ$ entre $[x]$ et lui-même revient à procéder de la manière suivante :

$$[x] \circ [x] = \{x \circ y | x \in [x], y \in [x]\} \tag{3.29}$$

On considère que x et y sont indépendants.

Exemple Par exemple, considérons l'intervalle $[x] = [0, 5]$, et l'opérateur intervalle "-". L'opération de soustraction est effectuée de la manière suivante :

$$[x] - [x] = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] = [-5, 5] \tag{3.30}$$

Or le résultat attendu est $[0, 0]$.

Prenons un autre exemple. Considérons $[y] = [-5, 4]$ et calculons le produit de $[y]$ par lui-même ; l'opération s'effectue comme suit :

$$[y] \times [y] = [min(25, -20, -20, 16), max(25, -20, -20, 16)] = [-20, 25] \tag{3.31}$$

Évaluons maintenant $[y]^2$. On obtient :

$$[y]^2 = [0, max((-5)^2, 4^2)] = [0, 25]$$

On peut noter que $[y]^2$ permet d'obtenir un résultat minimal. Evidemment, $[y]^2$ est une fonction d'inclusion minimale.

3.6.2 L'effet enveloppe

L'effet enveloppe (Kearfott, 1996a) consiste à surestimer l'image d'un vecteur d'intervalles (qui n'est pas en général un vecteur) en la représentant par un unique vecteur d'intervalles. Il est dû au fait que les intervalles sont définis parallèlement aux axes des repères. Or l'image d'un vecteur d'intervalles par une fonction peut prendre diverses formes. De plus, lors du déplacement d'un mobile, il est normal qu'il change de cap. Le repère restant normalement statique, les intervalles restent parallèles aux axes : le résultat obtenu enveloppe la position réelle du mobile. Ce problème est donc en partie lié au non-changement de repère. La figure 3.6, montre le problème dû à l'effet enveloppe. Le polytope (en vert) est supposé contenir la totalité des solutions possibles et supposé optimal. En rouge, nous représentons le plus petit rectangle tourné à 45 degrés et qui contient le polytope. Et en noir, le plus petit rectangle aligné selon les axes du repère et contenant le même polytope est très pessimiste. Le rectangle noir n'est pas optimal. Cependant, le fait que ce rectangle soit très grand ne signifie pas qu'il soit erroné.

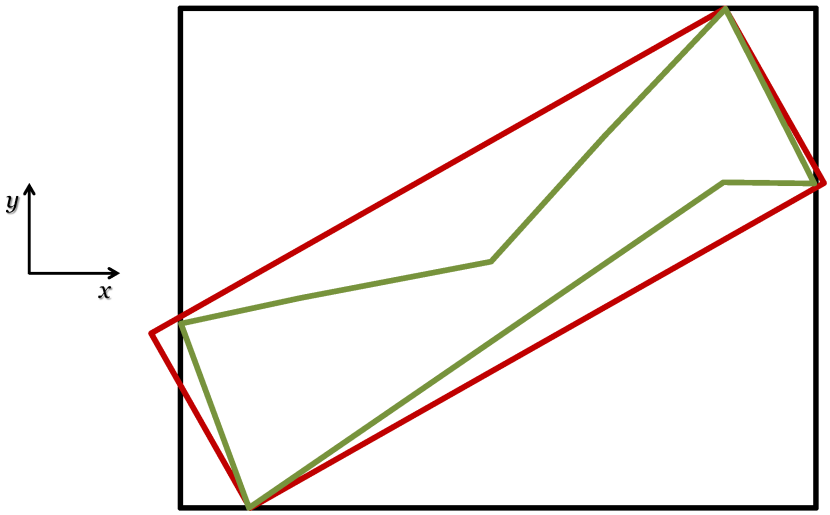


FIGURE 3.6 – Effet enveloppe : Non-optimalité de la représentation par des rectangles

3.7 Conclusion

L'analyse par intervalles est l'outil mathématique retenu pour notre travail. Dans ce chapitre, nous avons présenté un bref historique de l'analyse par intervalles ainsi que son intérêt.

Les opérateurs de l'arithmétique des intervalles et les opérateurs ensemblistes ont été abordés. Ces notions sont essentielles à la compréhension du reste de ce document.

Chapitre 4

Satisfaction de contraintes à intervalles

Sommaire

4.1 Introduction 56
4.2 Problème de satisfaction de contraintes 58
4.3 Problème de satisfaction de contraintes à intervalles (ICSP) 59
4.4 Notion de CSP dynamique 60
4.5 Les algorithmes de propagation de contraintes à intervalles (ICP) 62
4.5.1 Notion de consistance 62
4.5.2 La consistance d'enveloppe 63
4.5.3 La consistance de boite 70
4.5.4 kB-Consistance 74
4.6 Les solveurs à intervalles 78
4.6.1 Numerica 78
4.6.2 RealPaver 78
4.6.3 ALIAS 79
4.6.4 ICOS 79
4.6.5 GloptLab 79
4.6.6 Ibex 79
4.6.7 Choix de solveur 79
4.7 Conclusion 80

La localisation de véhicule peut être formulée comme un problème de satisfaction de contraintes (CSP). Les algorithmes de propagation de contraintes sur des intervalles permettent de résoudre ces CSP. Dans ce chapitre, nous verrons un état de l'art de la propagation par contraintes et des problèmes de satisfaction de contraintes en général puis des problèmes de satisfaction de contraintes à intervalles en particulier. Ensuite, les différents algorithmes de propagation de contraintes à intervalles seront présentés.

4.1 Introduction

Plusieurs problèmes d'intelligence artificielle sont souvent exprimés sous la forme de contraintes. Parmi ces problèmes figurent la gestion d'agenda, la gestion de trafic, la gestion de l'emploi du temps et certains problèmes de planification et d'optimisation (comme le problème de routage des réseaux de télécommunication).

Des problèmes plus ludiques aussi sont souvent formulés de cette manière. Ainsi trouve-t-on le problème des huit reines. Le but est de disposer huit reines d'un jeu d'échecs sur un échiquier de 8×8 cases sans que les reines ne puissent se menacer mutuellement, tout en respectant les règles du jeu d'échecs (la couleur des pièces étant ignorée). Deux reines ne devraient donc jamais partager ni la même rangée, ni la même colonne, ni la même diagonale. Ce problème a été proposé en 1848 par Max Bazzel (un joueur d'échec) et plusieurs mathématiciens y compris Gauss ont tenté de le résoudre par diverses approches. L'échiquier des huit reines a 92 solutions distinctes. Si les solutions qui diffèrent par des opérations de symétrie de l'échiquier sont comptées comme uniques, alors on a que 12 solutions uniques pour 8 reines. Un autre problème très connu est le théorème des quatre couleurs qui demande d'utiliser quatre (ou dans le cas général n) couleurs différentes, pour colorer une carte découpée en plusieurs régions connexes, de manière que deux régions adjacentes reçoivent deux couleurs distinctes. Deux régions sont adjacentes si elles ont une frontière (un segment) en commun et non pas seulement un point commun.

La plupart des premiers articles publiés sur la propagation de contraintes datent des années 1970. Ils ont principalement traité les domaines discrets (Clowes, 1971; Waltz, 1975; Mackworth, 1977). Dans les années 1980, Gallaire (Gallaire, 1985), Jaffar et Lassez (Jaffar and Lassez, 1987) ont remarqué que la programmation logique peut être considérée comme un type particulier de programmation par contraintes.

La programmation logique et la programmation par contraintes impliquent que l'utilisateur formule ce qui doit être résolu et non pas la manière de le résoudre. Plusieurs problèmes combinatoires tels que le problème d'ordonnancement et le problème d'établissement des horaires ont été très tôt formulés comme des CSP. Plusieurs méthodes existent pour résoudre ce genre de problème. Cependant, la propagation de contraintes est la technique la plus utilisée pour résoudre les CSP. Elle combine les techniques de consistance et de recherche systématique.

La consistance d'arc, la consistance de nœud et de chemin sont les trois principaux types de consistances connus et utilisés à cet effet. Le but des techniques

de consistance est d'éliminer dans les domaines des variables, toute valeur incohérente avec les contraintes d'un problème. Nous donnons une description plus détaillée des techniques de consistance sur les domaines finis dans l'annexe E. La consistance d'arc, le type de consistance le plus utilisé, est réalisée avec l'algorithme AC-3 (Mackworth, 1977) ou avec AC-5 (Van Hentenryck et al., 1992), AC-6 (Bessiere, 1994), etc. Ces algorithmes nécessitent d'avoir des CSP binaires c'est-à-dire dont chaque contrainte contient au plus deux variables. Cependant, plusieurs problèmes du monde réel sont formulés tout naturellement sous forme de CSP non binaires, les contraintes contenant plus de 2 variables (Yuanlin and Yap, 2000).

Deux approches ont été développées pour gérer les CSP non binaires. La première approche consiste à réécrire un CSP non binaire sous forme de plusieurs CSP binaires (Dechter and Pearl, 1989; Rossi et al., 1990). Et après cette étape appelée "binarisation", les méthodes classiques de résolution de CSP binaire sont utilisées pour résoudre ces nouveaux CSP. La deuxième approche s'applique quant à elle directement aux contraintes non binaires. Par exemple, GAC-4 (Mohr et al., 1988) constitue une généralisation de AC-4 aux contraintes non-binaires.

En 1987, Cleary (Cleary, 1987) et Davis (Davis, 1987) étaient les premiers à utiliser la propagation de contraintes dans le domaine de l'analyse par intervalles. Cleary a alors proposé un algorithme de contraction de domaine. Hyvönen (Hyvönen, 1989) en 1989, a aussi étudié un modèle de propagation de contraintes sur les intervalles. (Benhamou and Older, 1997) ont amélioré l'algorithme de Cleary et l'ont renommé HC3 (Benhamou et al., 1999) en raison de sa grande similarité avec l'algorithme classique AC-3. Ils ont ensuite proposé HC4 (Benhamou et al., 1999) qui n'a pas besoin de la décomposition des contraintes. Cependant, HC4 souffre du problème de dépendance [7].

Pour éviter ce problème, BC3 (Benhamou et al., 1994) a été développé mais BC3 souffre de lenteur. Une combinaison des deux algorithmes a donc été mise au point et nommée BC4 (Benhamou et al., 1999). BC5 (Granvilliers, 2000) utilise la méthode de Gauss-Seidel appliquée aux intervalles, et ainsi réduit encore plus le temps de traitement.

Le développement de l'algorithme 3B (Lhomme, 1993) a été guidé par la recherche de meilleurs résultats en termes de réduction des intervalles (et pas tant pour l'amélioration du temps de calcul).

La propagation de contraintes sur les intervalles a été introduite (Gning and Bonnifait, 2004) en robotique mobile il y a une dizaine d'années. D'autres travaux (Vincke et al., 2010; Vincke and Lambert, 2009),(Jaulin, 2006) aussi s'en sont inspirés. Ces travaux utilisent la technique de propagation Forward-Backward sur des contraintes primitives (binaires) en suivant le principe de l'algorithme de Waltz (Waltz, 1972). Il est donc nécessaire de procéder à une décomposition préalable des contraintes du problème en contraintes primitives. Cette décomposition des contraintes implique la création de nouvelles variables auxiliaires qui doivent aussi être réduites ; ce qui a un effet négatif sur la réduction des variables réellement utiles. Prenons par exemple le système d'équations suivant [8] :

7. Le problème de dépendance est expliqué dans la section 3.6.1.
8. Ce système d'équations est souvent est utilisé pour calculer le déplacement élémentaire

$$\left\{ \begin{array}{ll} [\delta s_k] = & \frac{[\pi]\left([w_l]([\delta p_l]+[\varepsilon_{p,l}])+[w_r([\delta p_r]+[\varepsilon_{p,r}])\right)}{P} \\ [\delta \theta_k] = & \frac{[\pi]\left([w_l]([\delta p_l]+[\varepsilon_{p,l}])-[w_r([\delta p_r]+[\varepsilon_{p,r}])\right)}{[e].P} \end{array} \right\} \tag{4.1}$$

La décomposition du système d'équations 4.1 en contraintes binaires donnera :

$$\left\{ \begin{array}{ll} [a_1] & = [\delta p_l] + [\varepsilon_{p,l}] \\ [a_2] & = [w_l] \times [a_1] \\ [a_3] & = [\delta p_r] + [\varepsilon_{p,r}] \\ [a_4] & = [w_r] \times [a_3] \\ [a_5] & = [a_4] + [a_3] \\ [a_6] & = [\pi] \times [a_5] \\ [\delta s_k] & = [a_6] / P \\ [a_7] & = [a_3] - [a_4] \\ [a_8] & = [\pi] \times [a_7] \\ [a_9] & = [e] \times P \\ [\delta \theta_k] & = [a_8] / [a_9] \end{array} \right\}$$

Nous avons introduit 9 nouvelles variables qui feront aussi l'objet de contraction pendant la résolution du système.

Nous présenterons dans ce chapitre, les problèmes de satisfaction de contraintes dans le cadre général, puis dans le cadre particulier des contraintes à intervalles. La notion de CSP dynamique sera abordée étant donnée que nous en ferons usage pour représenter notre problème de localisation. Les algorithmes de consistance seront ensuite présentés. Il s'ensuit le choix d'un solveur intervalle pour implémenter notre solution.

4.2 Problème de satisfaction de contraintes

Les problémes de satisfaction de contraintes, en anglais "Constraint Satisfaction Problem" (CSP) sont des problèmes mathématiques dont la résolution nécessite de trouver les états ou les objets qui satisfont un certain nombre de critères ou de contraintes. Un CSP est défini par :

- un ensemble de variables $\{x_1, x_2, .., x_n\}$,
- un ensemble de domaines $\{D_1, D_2, .., D_n\}$, tels que pour chaque variable x_i, un domaine D_i contienne les valeurs possibles,
- un ensemble de contraintes $\{C_1, C_2, .., C_p\}$, qui représentent des relations existant entre un sous-ensemble de variables et un sous-ensemble de valeurs.

Les contraintes peuvent être données en intension [9] c'est-à-dire sous forme d'une formule mathématique, ou en extension comme un ensemble. Définir une contrainte en intension consiste à l'écrire avec les propriétés mathématiques connues. Lorsqu'on énumère les tuples de valeurs qui appartiennent à une contrainte, on dit

et la rotation élémentaire d'un mobile.

9. Ne pas confondre avec "intention"

que cette contrainte est définie en extension. L'exemple suivant représente une contrainte définie en intension.

$$\begin{array}{c} x \in \{0,\ 1,\ 2\}\, et\, y \in \{0, 1, 2\} \\ x < y \end{array} \tag{4.2}$$

La contrainte de l'équation 4.2 peut s'écrire en extension sous forme de l'équation 4.3 ou l'équation 4.4 comme suit :

$$(x = 0\, et\, y = 1)\, ou\, (x = 0\, et\, y = 2)\, ou\, (x = 1\, et\, y = 2) \tag{4.3}$$

$$(x, y) \in \{(0, 1), (0, 2), (1, 2)\} \tag{4.4}$$

Remarquons que pour ces deux exemples, nous avons utilisé des contraintes basées sur des entiers pour simplifier la compréhension. Cependant, il faut noter qu'une caractéristique des contraintes sur les réels est qu'elles ne peuvent pas être exprimées en extension.

Le problème de satisfaction de contraintes consiste à trouver pour chaque i de 1 à n une valeur dans D_i pour x_i afin que toutes les contraintes soient satisfaites. En général, les CSP sont définis sur des domaines discrets.

Notion d'arité d'une contrainte : L'arité d'une contrainte est le nombre de variables qu'elle met en relation. Une contrainte est dite :

- **unaire** si son arité vaut 1,
  Exemple : "$2 \times x > 8$"
- **binaire** si son arité vaut 2,
  Exemple : "$x + y = 12$" ou encore "$A \cup B = A$"
- **ternaire** si son arité est égale à 3,
  Exemple : "$x - z < 3 \times y + 4$" ou encore "$NON(x)\ ET\ y\ OU\ z = FAUX$"
- ... etc
- **n-aire** si son arité vaut à n. Dans ce cas la contrainte est aussi dite "globale". Un exemple très connu est la contrainte "$toutesDifférentes(A)$", où A représente un ensemble de variables. Pour satisfaire cette contrainte, il faut que toutes les variables appartenant à A prennent des valeurs différentes.

4.3 Problème de satisfaction de contraintes à intervalles (ICSP)

La terminologie ICSP (Interval Constraint Satisfaction Problem) a été utilisée pour la première fois par Hyvönen en 1992 (Hyvönen, 1992) dans un article de référence. En 1993, Hyvönen et al. (1993) ont proposé l'outil "INC++"[10]. Les problèmes de conception numérique et de planification peuvent être formulés simplement sous la forme d'un ensemble d'équations contraignant les valeurs des

10. INC++ est une librairie basique permettant d'évaluer des fonctions intervalles et visant à résoudre des problèmes de satisfaction de contraintes. Cependant, cette librairie n'est pas open-source et n'est pas maintenue.

variables concernées, c'est-à-dire sous la forme d'un CSP numérique ou plus généralement d'un CSP à intervalles. Cependant, il n'existait pas (en 1992) d'outils permettant de résoudre correctement les ICSP dans le cas général.

Rappelons que jusqu'en 1984, la façon dont étaient codés les nombres flottants dépendait des architectures. Pour cette raison les programmeurs étaient obligés d'écrire les codes en fonction de l'architecture de destination. Plus tard, le calcul en nombre flottant a été normalisé (norme IEEE754) et la plupart des processeurs actuels traitent le calcul des nombres flottants de la même manière.

D'après Hyvönen and De Pascale (1999), la technologie principale sous-jacente à un solveur intervalle repose sur la satisfaction de contraintes étudiée dans le domaine de l'intelligence artificielle, la programmation logique et l'analyse par intervalles. Les techniques de résolution de contraintes à intervalles diffèrent des autres techniques numériques par un élément clé : aucune des solutions n'est accidentellement perdue. Par conséquent, toutes les solutions peuvent être trouvées si on dispose de suffisamment de temps et de ressources mémoire. De plus, si une situation est impossible, alors le problème n'a pas de solution. Cette garantie est valable même en cas d'erreur d'arrondi. En analyse par intervalles on utilise les arrondis vers l'extérieur. Et les nombres flottants imprécis sont représentés par des intervalles réduits et sûrs de contenir la vraie valeur.

Un ICSP (Hyvönen, 1992) ou CSP à intervalles est formulé comme suit :

- un ensemble X de n variables $\{x_1, x_2, .., x_n\}$,
- un ensemble D de n domaines $\{D_1, D_2, .., D_n\}$, tels que pour chaque variable x_i, un domaine D_i contienne les valeurs possibles et D_i est un intervalle ou une union d'intervalles.
- un ensemble C de p contraintes $\{C_1, C_2, .., C_p\}$, qui représentent des relations existant entre les variables. Les contraintes sont données en intension.

On notera ici, que les domaines sont des intervalles ou unions d'intervalles. En général, il s'agit d'intervalle unique par domaine (et non d'une union d'intervalle). Aussi on remarquera que les contraintes sont exprimées uniquement en intension.

Les équations forment un réseau de contraintes consistant en un ensemble de variables et d'opérations les reliant sur des domaines continus et fermés.

Chaque domaine D_i est inclus dans $\mathbb{R}$. La propagation de contraintes consiste à itérer les réductions de domaines en utilisant l'ensemble des contraintes du problème jusqu'à ce qu'aucune contraction ne soit plus possible. Avec le concept de l'analyse par intervalles, chaque domaine continu est défini comme un intervalle. Le produit cartésien des domaines intervalles ainsi contractés est la solution et elle est garantie de contenir la position du véhicule, si on se place dans un contexte de localisation de véhicule. Ce genre de contraction est réalisé avec plusieurs types d'algorithmes.

4.4 Notion de CSP dynamique

Un CSP est dit "dynamique" (Dechter and Dechter, 1988) ou $DCSP$ lorsque ses contraintes résultent d'un changement sur les contraintes d'un CSP qui le précède dans le temps. Ceci représente de nouveaux faits (événements) intervenus dans l'environnement en train d'être modélisé. Comme résultat de ce changement

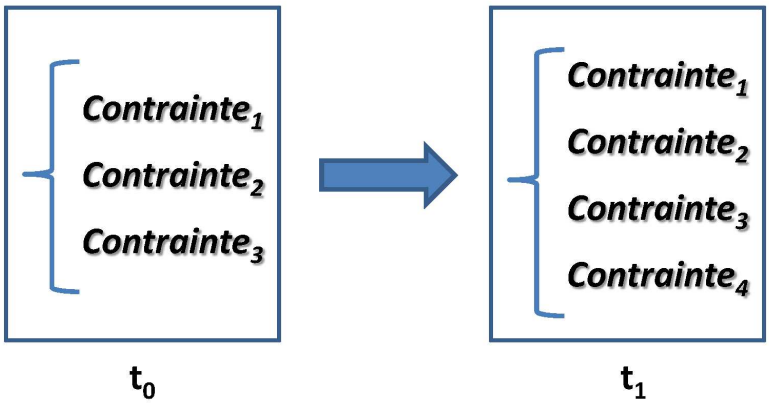


FIGURE 4.1 – Illustration d'un CSP dynamique (Restriction)

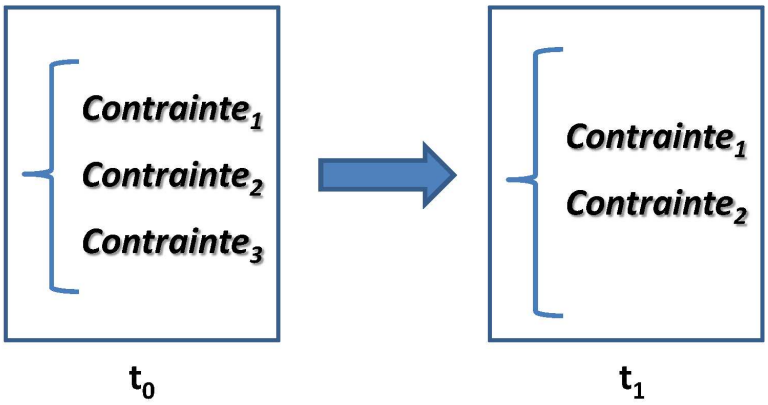


FIGURE 4.2 – Illustration d'un CSP dynamique (Relaxation)

incrémental, l'ensemble des solutions du CSP peut être soit réduite (dans ce cas, on parle de "restriction") soit plus dense (on parle alors de "relaxation"). Dans la figure 4.1, la contrainte "$Contrainte_4$" est ajouté au temps t_1 alors que dans la figure 4.2 c'est la contrainte "$Contrainte_3$" qui est supprimée.

On réalise des restrictions sur un CSP lorsqu'une nouvelle contrainte est imposée à un sous-ensemble des variables existant (par exemple en forçant une variable à prendre une certaine valeur), ou lorsqu'une nouvelle variable est ajoutée au système via certaines relations. Les restrictions étendent toujours le modèle dans la mesure où elles ajoutent des variables et des contraintes de sorte que le graphe de contraintes devienne plus grand.

Lorsque les contraintes qui étaient considérées initialement se trouvent être invalides (incorrectes ou inappropriées) à un moment donné et doivent par conséquent être retirées du CSP, on réalise des relaxations sur le CSP. En pratique, il n'est pas nécessaire de supprimer les contraintes du CSP pour avoir l'effet de relaxation. On peut modéliser chaque contrainte susceptible d'être relaxée de

sorte que l'on puisse inclure une variable booléenne dont la valeur indique si la contrainte est active ou pas à un moment donné. Cela permet de garder toutes les contraintes qui ont été ajoutées.

Dans le cas de cette thèse, ce concept est utilisé pour définir des ICSP représentant notre modèle qui évolue à chaque pas de temps, l'environnement d'un véhicule étant naturellement dynamique.

4.5 Les algorithmes de propagation de contraintes à intervalles (ICP)

Nous présenterons dans cette section quelques algorithmes de propagation de contraintes qui ont été développés typiquement pour les ICSP.

4.5.1 Notion de consistance

Nous présentons ici, la notion de consistance qui constitue le concept fondamental et sous-jacent à la satisfaction de contraintes. Considérons une équation $f(x, y) = 0$; soient x et y des éléments respectivement des intervalles de $[x]$ et $[y]$. On peut dire que les valeurs de $x \in [x]$ et $y \in [y]$ sont consistantes par rapport à la fonction f si :

- pour tout $x \in [x]$, il existe $y \in [y]$ tel que $f(x, y) = 0$ et
- pour tout $y \in [y]$, il existe $x \in [x]$ tel que $f(x, y) = 0$

Ce concept peut être généralisé à plusieurs variables (voir (Collavizza et al., 1999)). Supposons que pour un sous-ensemble de valeurs $x \in [x]$, il n'existe pas de valeurs $y \in [y]$ tel que $f(x, y) = 0$. Dans ce cas, ces valeurs de x peuvent être éliminées pendant la recherche des solutions de l'équation $f(x, y) = 0$. Notons que f peut être une fonction de plusieurs variables. Ce concept s'applique aussi à un système de fonctions non-linéaires considérant la recherche de solution dans un vecteur d'intervalles. Dans ce cas, le concept de consistance peut s'appliquer à chaque équation du système pour exclure les sous-boîtes d'une boîte donnée qui ne contiennent pas la solution.

On parle aussi de "maintenance de la consistance" ou de "filtrage des domaines". Le principe est de supprimer des domaines les valeurs non solutions. Le terme consistance est aussi utilisé pour désigner la propriété que vérifie le CSP après l'application d'un filtrage donné. On notera que si l'application du filtrage (ou maintenance de la consistance) a pour effet de vider le domaine d'une variable donné, alors cette variable n'a plus aucune valeur consistante. On en déduit que le CSP n'a pas de solution. Une contrainte peut être considérée plus forte qu'une autre. La consistance peut être localement ou globalement maintenue.

Consistance locale Lorsqu'un algorithme permet de maintenir ou réduire les domaines des variables sur uniquement une contrainte, on parle de consistance locale (ou encore "consistance strictement locale") [11]. Les deux types de consistance locale les plus connus sont : la consistance d'enveloppe et la consistance de boîte.

11. Elle est locale par rapport à la contrainte

Lorsque une méthode permet de vérifier certaines propriétés sur un sous-ensemble des contraintes, on parle de "consistance non strictement locale".

Consistance globale Une méthode permettant de réduire les domaines en tenant compte de toutes les contraintes d'un système est dite consistance globale. 3B est parfois considéré comme un algorithme de consistance globale.

4.5.2 La consistance d'enveloppe

La consistance d'enveloppe[12] (Benhamou et al., 1999) est réalisée pour une contrainte c par rapport à un vecteur d'intervalles B si et seulement si :

$$B = Hull_{\square}(B \cap \rho_c) \tag{4.5}$$

où $Hull_{\square}$ est une fonction qui calcule une approximation du plus petit vecteur d'intervalles contenant son argument et ρ_c représente la relation sous-jacente de la contrainte donnée c. Le vecteur d'intervalles B est le produit cartésien de n intervalles $B = I_1 \times \cdots \times I_n$. Si P est un ICSP impliquant un vecteur d'intervalles B, P est enveloppe-consistant si chaque contrainte c de P satisfait la consistance d'enveloppe. La consistance d'enveloppe utilise essentiellement des projections des fonctions qui sont des extensions aux intervalles des contraintes. Mais les projections donnent généralement des unions d'intervalles et non pas des intervalles. Il faut donc regrouper les intervalles résultats en un avec la fonction "enveloppe intervalle"[13] vue dans la section 3.2.4. La consistance d'enveloppe est aussi appelée 2B-consistance (Collavizza et al., 1998).

D'après Lhomme (1993), la contrainte c est enveloppe-consistante pour la variable x, de domaine $D_x = [a, b]$, s'il existe des valeurs dans les domaines de toutes les autres variables de c qui satisfont c lorsque x est instancié à $[a, a^+]$ et lorsque x est instancié avec $[b^-, b]$. Notons que a^+représente la plus petite valeur réelle supérieur à a et b^- le plus grand nombre réel inférieur à b. Prenons par exemple, la contrainte suivante :

$$z = x + y \tag{4.6}$$

avec $x \in [1, 3]$, $y \in [0, 3]$ et $z \in [5, 3]$.

Avec le calcul par intervalles et en faisant de la projection directe, on peut déduire de la contrainte 4.6 que[14] $z \in [1, 3]$.

Les domaines des opérandes[15] sont calculés par les fonctions de projection inverse. Dans notre cas, par exemple, le domaine de y est calculé de la façon suivante (où $[x]$, $[y]$ et $[z]$ représentent les domaines des variables x, y et z) :

$[y] \leftarrow [y] \cap ([z] - [x])$
$[y] \leftarrow [0, 3] \cap ([1, 3] - [1, 3])$
$[y] \leftarrow [0, 3] \cap [2, 2]$

12. Hull Consistency
13. D'où le nom consistance d'"Enveloppe"
14. En faisant $[z] = [5, 3] \cap ([1, 3] + [0, 3])$
15. x et y sont des opérandes

$[y] \leftarrow [0, 2]$

Finalement, on déduit que $y \in [0, 2]$ et que le domaine de x reste inchangé.

Un des problèmes connus de la consistance d'enveloppe est qu'elle est sensible aux multiples occurrences des variables (Benhamou et al., 1994) dans une contrainte. Les occurrences multiples d'une même variable sont traitées comme des variables différentes (mais de même domaine). Elle tend à fournir des résultats sur-estimés même s'ils contiennent le vrai résultat. Par exemple, lorsque l'on considère une contrainte comme $x \times x + y^2 = 2$, avec $(x, y) \in [-2, 4] \times [-1, 1]$, HC4 ne peut plus réduire le vecteur d'intervalles initial, pour la simple raison que la variable x figure plus d'une fois dans la contrainte. Par contre, il peut réduire le domaine initial si la contrainte est formulée sous la forme $x^2+y^2 = 2$ où chaque variable figure une seule fois. Détaillons l'exemple :

Premier cas : la contrainte s'écrit $x \times x + y^2 = 2$ La variable x, étant présente 2 fois, sera représentée par deux variables x et $x2$, telles que $x \in [-2, 4]$ et $x2 \in [-2, 4]$. De plus, une nouvelle contrainte s'ajoute au problème : $x = x2$. On calcule ensuite les différents domaines comme suit :

$[x] = [x] \cap ((2 - [y]\hat{}2)/[x2])$
$[x2] = [x2] \cap ((2 - [y]\hat{}2)/[x])$
$[y] = [y] \cap \sqrt{2 - [x] \times [x2]}$
$[x] = [x] \cap [x2]$

Numériquement, on obtient pour $x1$:

$[x] = [-2, 4] \cap ((2 - [-1, 1]\hat{}2)/[-2, 4])$
$[x] = [-2, 4] \cap ((2 - [0, 1])/[-2, 4])$
$[x] = [-2, 4] \cap ([1, 2]/[-2, 4])$
$[x] = [-2, 4] \cap [-\infty, +\infty]$ car $1/[-2, 4] = [-\infty, +\infty]$
$[x] = [-2, 4]$

Le même calcul sera effectué pour $[x2]$. Et on obtient $[x2] = [-2, 4]$. On calcule $[y]$ comme suit :

$[y] = [y] \cap \sqrt{2 - [x] \times [x2]}$
$[y] = [-1, 1] \cap \sqrt{2 - [-2, 4] \times [-2, 4]}$
$[y] = [-1, 1] \cap \sqrt{2 - [-2, 4] \times [-2, 4]}$
$[y] = [-1, 1] \cap \sqrt{2 - [-16, 8]}$
$[y] = [-1, 1] \cap \sqrt{[-14, 10]}$
$[y] = [-1, 1] \cap [-\sqrt{10}, \sqrt{10}]$
$[y] = [-1, 1]$

Le domaine de x est déduit de ceux de x et $x2$ par intersection.

$[x] = [x] \cap [x2]$
$[x] = [-2, 4] \cap [-2, 4] = [-2, 4]$

Finalement, aucun domaine n'est réduit : on retrouve $(x, y) \in [-2, 4] \times [-1, 1]$.

Deuxième cas : la contrainte s'écrit $x^2 + y^2 = 2$ Chaque variable n'est représentée qu'une fois. On calculera alors les domaines de x et y de la manière suivante :

$[x] = [x] \cap \sqrt{2 - [y]^2}$

$[y] = [y] \cap \sqrt{2 - [x]^2}$
Numériquement cela donne pour $[x]$:
$[x] = [-2, 4] \cap \sqrt{2 - [-1, 1]^2}$
$[x] = [-2, 4] \cap \sqrt{[1, 2]}$
$[x] = [-2, 4] \cap \left([-\sqrt{2}, -1] \sqcup [1, \sqrt{2}]\right)$
$[x] = [-2, 4] \cap \left([-\sqrt{2}, \sqrt{2}]\right)$
$[x] = [-\sqrt{2}, \sqrt{2}]$
Pour $[y]$, le calcul est similaire et donne :
$[y] = [-1, 1] \cap \sqrt{2 - [-2, 4]^2}$
$[y] = [-1, 1] \cap \sqrt{[-14, 2]}$
$[y] = [-1, 1] \cap \left([-\sqrt{2}, 0] \sqcup [0, \sqrt{2}]\right)$
$[y] = [-1, 1] \cap \left([-\sqrt{2}, \sqrt{2}]\right)$
$[y] = [-1, 1]$

Finalement, on remarque que le domaine de x a été réduit. Le domaine contracté vaut alors : $[-\sqrt{2}, \sqrt{2}] \times [-1, 1]$.

La consistance d'enveloppe est implémentée dans deux algorithmes connus sous le nom de HC3 (Cleary, 1987) et HC4 (Benhamou et al., 1999). HC3 [16] maintient la consistance d'enveloppe sur des contraintes primitives. Nous n'allons donc pas l'utiliser puisqu'il nécessite la décomposition en contraintes primitives. Un de nos objectifs sera d'éviter cette décomposition. Nous présenterons donc l'algorithme HC4.

4.5.2.1 HC4

HC4 (Benhamou et al., 1999) réalise la consistance d'enveloppe sur les contraintes complexes sans décomposition préalable en contraintes primitives. Il répète la propagation et traite les contraintes une par une en utilisant la sous-routine HC4revise.

Les domaines sont réduits en y supprimant les valeurs inconsistantes. HC4revise utilise une représentation en arbre binaire des contraintes, où les feuilles sont les constantes et les variables et les nœuds représentent les opérations élémentaires comme $/$, $\times$, $log()$, etc. HC4revise fonctionne en deux phases :

- La phase de "Evaluation-avant" [17] traverse le graphe partant des feuilles jusqu'à la racine de l'arbre. Elle évalue récursivement l'intervalle de chaque sous-expression représentée par le nœud courant en utilisant une extension naturelle des fonctions sous-jacentes.
- La phase de "Propagation-arrière" [18] quant à elle parcourt l'arbre de la racine aux feuilles. Elle applique sur chaque nœud un opérateur de contraction (une projection). L'opérateur de contraction réduit donc l'intervalle du nœud courant en supprimant les valeurs inconsistantes vis-à-vis de l'opérateur de base du nœud ascendant.

16. HC3 est présenté en annexe G
17. Forward Evaluation
18. Backward Propagation

Algorithm 4.1 HC4

Function : HC4($\{c_1, ..., c_m\}$, B=$I_1 \times ... \times I_n$)
▷ c_i : contraintes, B : un vecteur d'intervalles

1: $S \leftarrow \{c_1, ..., c_m\}$
2: **while** ($S \neq \emptyset$ et $B \neq \emptyset$) **do**
3: $c \leftarrow$ choisir une contrainte c_i dans S
4: $B' \leftarrow$ HC4Revise(c,B)
5: **if** $B' \neq B$ **then**
6: $S \leftarrow S \cup \{c_j \mid \exists x_k \in \text{Var}(c_j) \wedge I'_k \neq I_k\}$
7: $B \leftarrow B'$
8: **else**
9: $S \leftarrow S \setminus \{c\}$
10: **end if**
11: **end while**
12: **return** B

Tel que présenté dans l'algorithme 4.1, HC4 prend en entrée un ICSP, ensemble de contraintes et une boite (vecteur d'intervalles). Il réduit les domaines des variables avec la sous-routine nommée "HC4revise".

A la ligne 3 de l'algorithme 4.1, c est une contrainte représentée sous la forme d'un arbre binaire et $(t_1, ..., t_p)$ sont les nœuds de cet arbre (variables, constantes ou opérations).

Après l'appel de HC4Revise (ligne 4), on vérifie s'il y a eu contraction du domaine initiale (ligne 5). Dans le cas où il y a eu réduction du domaine, on ajoute à la liste des contraintes toutes les contraintes c_j dont le domaine d'une variable vient d'être réduit ; ceci dans le but de les revisiter après car HC4Revise n'est pas idempotent c'est-à-dire qu'une contrainte qui a réduit peut encore réduire. Rappelons que la fonction Var (ligne 6) retourne les variables de la contrainte fournie en paramètre.

Et s'il n'y a pas eu contraction (ligne 9), alors on retire la contrainte c de la liste S des contraintes. Le retrait de c ne signifie pas qu'une contrainte inutile à un moment sera toujours inutile. La contrainte c est retirée de la liste S en sachant qu'on peut la rajouter à la liste chaque fois qu'elle sera concernée par un domaine qui vient de subir une réduction. L'algorithme s'arrête lorsqu'il n'y a plus de réduction sur aucun domaine ou si l'on détecte une absence de solution.

Remarquons que l'algorithme 4.2 comporte 3 fonctions distinctes :

- HC4revise (lignes 1 à 7),
- ForwardEvaluation (lignes 8 à 19) et
- BackwardPropagation (lignes 20 à 38).

Détaillons, à présent, HC4revise. HC4revise prend en entrée, une contrainte c et une boite B. La contrainte $c = r(t_1, ..., t_p)$ est représentée par un arbre d'attributs et son nœud racine contient les relations de type p-aire[19]. Le symbole r[20], et les termes t_i sont des nœuds qui contiennent soit un symbole d'opérateur, une

19. des relations impliquant p variables
20. r représente la relation entre les termes t_i

Algorithm 4.2 HC4 Sous-programmes

Function : HC4revise($c = r(t_1, ..., t_p)$, $B = I_1 \times ... \times I_n$)
▷ c : une contrainte de réels, B : un vecteur d'intervalles

1: $D_A \leftarrow B$
2: **for each** $i \in \{1, ..., p\}$ **do**
3: ForwardEvaluation(t_i, D_A)
4: **end for**
5: BackwardPropagation(c, D_A)
6: $B \leftarrow \text{Hull}_{\square}(D_A)$
7: **return** B

Function : ForwardEvaluation(t, $B = I_1 \times .... \times I_n$)
▷ t : un arbre, B : un vecteur d'intervalles

8: **switch** t **do**
9: **case** $\Diamond(t_1, ..., t_j)$ ▷ Le nœud est une expression
10: **for each** $i \in \{1, ..., j\}$ **do**
11: ForwardEvaluation(t_i,B)
12: **end for**
13: $t.fwd \leftarrow \blacklozenge(t_1.fwd, ..., t_j.fwd)$
14: **case** a ▷ Le nœud est une valeur constante
15: $t.fwd \leftarrow apx_A(\{a\})$
16: **case** x_k ▷ Le nœud est une variable
17: $t.fwd \leftarrow I_k$
18: **end switch**
19: **return** t

Function : BackwardPropagation(t, $B = I_1 \times .... \times I_n$)
▷ t : un arbre, B : un vecteur d'intervalles

20: **switch** t **do**
21: **case** $r(t_1, ..., t_m)$ ▷ Le nœud racine
22: $c \leftarrow r(x_1, ..., x_m)$
23: $B' \leftarrow (t_1.fwd, ..., t_m.fwd)$
24: **for each** $i \in \{1, ..., m\}$ **do**
25: $t_i.bwd \leftarrow \pi_i(apx_A(p_c \cap B'))$
26: BackwardPropagation(t_i, B)
27: **end for**
28: **case** $\Diamond(t_1, ..., t_h)$ ▷ Un terme sous la racine
29: $c \leftarrow (\Diamond(x_1, ..., x_h) = x_{h+1})$
30: $B' \leftarrow (t_1.fwd, ..., t_h.fwd, t.bwd)$
31: **for each** $i \in \{1, ..., h\}$ **do**
32: $t_i.bwd \leftarrow \pi_i(apx_A(p_c \cap B'))$
33: BackwardPropagation(t_i, B)
34: **end for**
35: **case** x_j ▷ Une variable
36: $I_j \leftarrow I_j \cap t.bwd$ ▷ Les domaines intersectés pour prendre en compte les multiples occurrences de x_j
37: **end switch**
38: **return** (t, B)

variable ou une constante. A part la racine, chaque nœud t_i inclut deux attributs intervalles nommés $t_i.fwd$ et $t_i.bwd$.

Pour chaque terme t_i, un appel est fait à la fonction ForwardEvaluation (ligne 3), afin de calculer les $t_i.fwd$. A l'issue de cette étape, HC4revise dispose des intervalles de propagation-avant pour chaque terme de la contrainte. Afin d'obtenir les intervalles $t_i.bwd$ pour chaque nœud, la fonction BackwardPropagation est appelée (ligne 5). On obtient ainsi le domaine final en calculant pour chaque domaine le plus petit intervalle pouvant le contenir (ligne 6). Ceci permet, entre autre, d'éviter des unions d'intervalles.

La fonction ForwardEvaluation permet de calculer les intervalles $t_i.fwd$ de chaque nœud [21]. Elle permet de monter dans l'arborescence des feuilles vers la racine. Suivant que le nœud représente une expression, une variable ou une constante (intervalle dégénéré), une action spécifique est réalisée. Ainsi, si un nœud est une expression [22], on évalue d'abord les intervalles pour chacun de ses nœuds enfants. Ensuite avec une fonction d'inclusion naturelle [23] de l'expression, on calcule son intervalle $t_i.fwd$ (ligne 13). $\blacklozenge$ représente la fonction d'inclusion naturelle [24] de la fonction de réels $\Diamond$ et $apx_A(\{a\})$ représente une approximation intervalle du nombre réel qui correspond au nœud de constante a.

La fonction BackwardPropagation permet de calculer, pour chaque nœud t_i, la valeur de l'intervalle $t_i.bwd$. Elle permet de traverser l'arborescence de la racine vers des feuilles. Elle évalue pour chaque nœud la projection associée à son nœud parent. Plus précisément, 2 types de nœuds sont considérés :

- Pour le nœud racine $r(t_1, ..., t_m)$, les attributs $t_i.bwd$ $(i \in \{1, ..., m\})$ sont calculés par la i^{eme} projection de la contrainte $r(x_1, ..., x_m)$, avec $x_1 \in t_1.fwd, ..., x_m \in t_m.fwd$;

- Pour le nœud $t_{h+1} = \blacklozenge(t_1, ..., t_h)$, les attributs $t_i.bwd$ $(i \in \{1, ..., h\})$ sont calculés par la i^{eme} projection de la contrainte $x_{h+1} = \blacklozenge(x_1, ..., x_h)$, où $x_i \in t_i.fwd$ pour tout $i \in \{1, ..., h\}$, et $x_{h+1} \in t_{h+1}.bwd$; π_i représente la i^{eme} projection. Si le calcul est correctement effectué jusqu'au bout, le domaine de chaque variable x_j dans la contrainte est intersecté avec celui de $x_j.bwd$ (ligne 36).

Une caractéristique intéressante de HC4 est l'indépendance des résultats par rapport à l'ordre dans lequel les contraintes sont appelées.

Les figures 4.3 et 4.4 présentent le calcul effectué par HC4 sur la contrainte $(x + y)^2 = z \times 2$, avec les domaines $I_x = [2, 5]$, $I_y = [0, 5]$, $I_z = [0, 8]$, tels que $x \in I_x$, $y \in I_y$, $z \in I_z$.

Au cours de la phase de propagation-avant, l'algorithme calcule :
$(+(x, y)).fwd = [2, 5] + [0, 5] = [2, 10]$
$(\hat{}(+(x, y)), 2).fwd = [2, 10]\hat{}2 = [4, 100]$
$(\times(z, 2)).fwd = [0, 8] \times 2 = [0, 16]$

21. Rappelons que chaque nœud t_i dispose de 2 intervalles : $t_i.fwd$ et $t_i.bwd$
22. Une expression représente une relation entre plusieurs nœuds. Exemple : (a+b)^2
23. aussi appelée "extension naturelle"
24. C'est une écriture au sens intervalle d'une expression en y remplaçant les variables réelles ($\in \mathbb{R}$) par leurs équivalents intervalles.

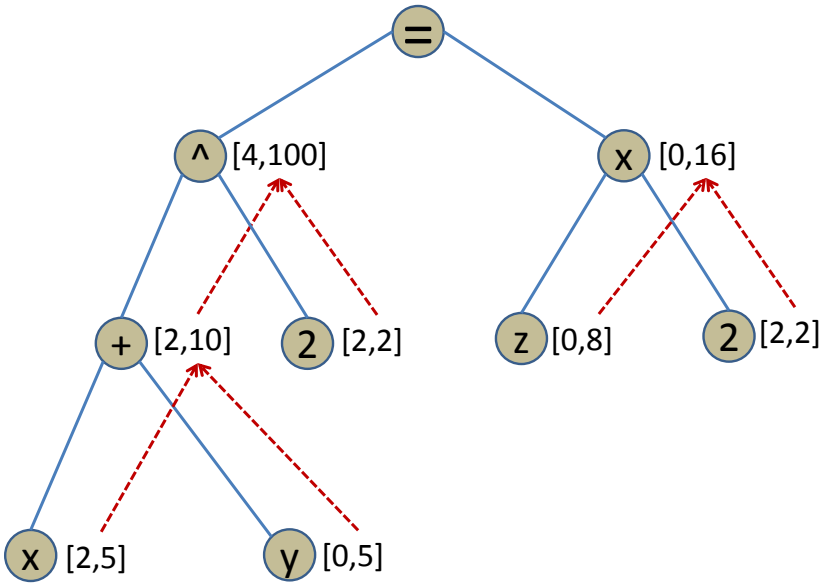


FIGURE 4.3 – Application de HC4 (propagation-avant)

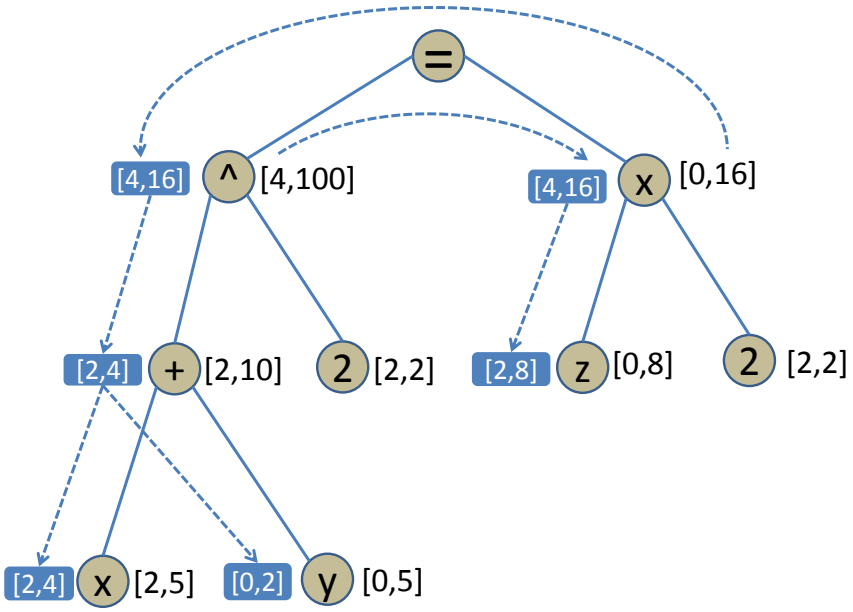


FIGURE 4.4 – Application de HC4 (Propagation-arrière)

Au cours de la phase de propagation-arrière, on calcule :
$(\hat{}(+(x,y)),2).bwd \cap (\times(z,2)).bwd = [4,100] \cap [0,16] = [4,16]$
$(\hat{}(+(x,y)),2).bwd = [4,16]$
$(\times(z,2)).bwd = [4,16]$
$(+(x,y)).bwd = \left(\sqrt{}(\hat{}(+(x,y)),2).bwd\right) \cap ((+(x,y)).fwd) = [2,4] \cap [2,10] = [2,4]$
$x.bwd = ((+(x,y)).bwd - y.fwd) \cap x.fwd = ([2,4] - [0,5]) \cap [2,5] = [2,4]$
$y.bwd = ((+(x,y)).bwd - x.fwd) \cap y.fwd = ([2,4] - [2,5]) \cap [0,5] = [0,2]$
$z.bwd = (\div(\times(z,2)).bwd, 2) \cap z.fwd = ([4,16] \div 2) \cap [0,8] = [2,8]$

Finalement les nouveaux domaines obtenus sont : $I_x = [2,4]$, $I_y = [0,2]$ et $I_z = [2,8]$. Le calcul $(\hat{}(+(x,y)),2).bwd \cap (\times(z,2)).bwd$ au nœud racine, correspond à l'interprétation de l'égalité entre les termes $(\hat{}(+(x,y)),2).bwd$ et $(\times(z,2)).bwd$. Il faut noter que dans la figure les termes à droite des nœuds représentent leur intervalle de propagation noté "fwd" et les termes à gauche en bleu représentent leur intervalle de rétro-propagation noté "bwd".

4.5.3 La consistance de boite

La consistance de boîte[25] (Benhamou et al., 1994) a été proposée afin de remédier aux problèmes[26] liés à la décomposition des contraintes complexes en contraintes primitives lors des calculs de la consistance d'enveloppe avec l'algorithme HC3[27]. Considérons l'équation non linéaire suivante :

$$f(x_1, ..., x_n) = 0 \tag{4.7}$$

Cette équation peut faire partie d'un système à résoudre ou d'un problème d'optimisation à satisfaire. Dans tous les cas, on peut supposer que nous cherchons une solution dans une boîte $[x]$. On souhaite utiliser la consistance de boîte pour éliminer les sous-boîtes non solutions. Pour cela, on remplace toutes les variables par les bornes de leurs intervalles, exceptée la $i^{ème}$ variable. On obtient l'équation 4.8 suivante :

$$q([x_i]) = f([\underline{x}_1, \overline{x}_1], ..., [\underline{x}_{i-1}, \overline{x}_{i-1}], [x_i], [\underline{x}_{i+1}, \overline{x}_{i+1}], ..., [\underline{x}_n, \overline{x}_n]) = 0 \tag{4.8}$$

Dans cette équation 4.8, on peut noter que seule $[x_i]$ est une variable. Toutes les autres variables doivent être représentées par les bornes (des nombres réels) $\underline{x}$, $\overline{x}$ de leurs intervalles. On dira de cette équation 4.8 qu'elle est univariable[28]. Si $0 \notin q([x_i])$ pour x_i dans un certain sous-intervalle $[x'_i]$ de $[x_i]$, alors il n'y a pas consistance pour $x_i \in [x'_i]$ et la sous-boîte $([\underline{x}_1, \overline{x}_1]$, ..., $[\underline{x}_{i-1}, \overline{x}_{i-1}]$, $[x'_i]$, $[\underline{x}_{i+1}, \overline{x}_{i+1}]$, ..., $[\underline{x}_n, \overline{x}_n])$ de $[x]$ doit être éliminée. Cette opération est réalisée pour chacune des variables afin de maintenir la consistance de boîte dans une contrainte d'un

25. Aussi appelée Box-consistance
26. Erreurs d'écriture, introduction de nouvelles variables, augmentation de l'espace de recherche, augmentation du temps de calcul, etc...
27. Premier algorithme de consistance d'enveloppe. HC3 est détaillé en annexe G
28. Il s'agit donc d'une équation à une variable

ICSP. Plusieurs algorithmes implémentent ce type de consistance : BC3, BC4 (notons que BC4 est une approche hybride).

4.5.3.1 BC3

BC3 (Benhamou et al., 1994) contracte les domaines via une procédure de recherche basée sur la bissection avec les extensions naturelles [29] des contraintes. Il consiste en un remplacement du test de satisfaction de contraintes sur un domaine réel par une procédure de réfutation sur les domaines. Si $[I_k]$ est le domaine de la variable x_k, alors $[I_k]$ est une boîte consistante si on a :

$$[I_k] = \{x_k \in [I_k] : 0 \in F([I_1], ..., [I_{k-1}], \{x_k\}, [I_{k+1}], ..., [I_{n-1}])\} \tag{4.9}$$

où F est une extension intervalle naturelle de la contrainte. Il utilise la méthode de Newton basée sur les intervalles (Alefeld and Herzberger, 1983). Si $[I_k]$ inclut la valeur 0, alors une inclusion améliorée $[I_{k+1}]$ peut être calculée de la manière suivante :

$$[I_{k+1}] \leftarrow [I_k] \cap (m([I_k]) - \frac{F(m([I_k]))}{F'([I_k])}) \tag{4.10}$$

$m([I_k])$ représente le milieu de $[I_k]$ et F' est la dérivée de F. L'ensemble solution est encadré par le résultat de cette itération qui renvoie une séquence d'intervalles imbriqués. Il est important que $0 \notin F'([I_k])$ pour chaque $k \geq 0$. Dans le cas où $0 \in F'([I_k])$ la méthode de Newton par intervalles n'est pas utilisée et l'exemple suivant montre le principe utilisé. Soit la contrainte $y - 2x = 0$ et $(x, y) \in [0, 1] \times [0, 8]$. La borne inférieure de $[I_y]$ est boite-consistante puisque $0 \in 0 - 2 \times [I_x] = [-2, 0]$. Mais la borne supérieure de $[I_y]$ ne l'est pas parce que $0 \notin 8 - 2 \times I_x = [6, 8]$. Le domaine de y est alors divisé en 2 pour pouvoir trouver la valeur la plus consistante de la manière suivante :

$[0, 8]$ est divisé en $[0, 4]$ et $[4, 8]$
$[0, 4] - 2 \times I_x = [-2, 4]$
$[4, 8] - 2 \times I_x = [2, 8] \Longrightarrow [4, 8]$ est éliminé, car il ne contient pas 0
$[0, 4]$ est divisé en $[0, 2]$ et $[2, 4]$ car la borne 4 n'est pas boîte consistante
$[0, 2] - 2 \times I_x = [-2, 2]$
$[2, 4] - 2 \times I_x = [0, 4]$
$[2, 4]$ est divisé en $[2, 3]$ et $[3, 4]$
$[2, 3] - 2 \times I_x = [0, 3]$
$[3, 4] - 2 \times I_x = [1, 4] \Longrightarrow [3, 4]$ est éliminé
...

Le domaine final de y est $[I_y] = [0, 2]$. La même technique est appliquée pour déterminer $[I_x] = [0, 1]$.

L'algorithme BC3 est détaillé dans (Benhamou et al., 1994). Donnons à présent, une description succincte de l'algorithme de BC3. Pour toute contrainte $c(x_1, ..., x_n)$ du type $f(x_1, ..., x_n) = 0$, on génère les fonctions d'inclusion univariables $F(I_{x_1}, ..., I_{x_{i-1}}, I_{x_i}, I_{x_{i+1}}, ..., I_{x_n})$ associées aux variables x_i. Le processus

29. Les fonctions d'inclusion naturelles

consiste alors à trouver le zéro le plus à gauche et le plus à droite dans toutes les fonctions univariables sur intervalles générées. Il s'agit de résoudre les équations sous la forme :

$$F(I_{x_1}, ..., I_{x_{i-1}}, I_{x_i}, I_{x_{i+1}}, ..., I_{x_n}) = 0$$

Algorithm 4.3 LNAR

Function : LNAR(D, F_{xi})

```
1:  L ← {D}
2:  right ← right(D)
3:  while (L ≠ ∅) do
4:      D' ← first_from_left(L)
5:      L ← L/D'
6:      if 0 ∈ F_xi(D') then
7:          D' ← NEWTON(D', F_xi)
8:          if (D' ≠ ∅) then
9:              if (0 ∈ F_xi([left(D'), ⌜left(D')⌝])) then
10:                 return [left(D'), right]
11:             else
12:                 SPLIT(D', D_left, D_right)
13:                 L ← L ∩ {D_left, D_right}
14:             end if
15:         end if
16:     end if
17: end while
18: return ∅
```

Ensuite, on calcule pour chaque variable x_i de domaine D_i les zéros [30] les plus à gauche (z_g) et à droite (z_d) appartenant à D_i de la fonction F_{x_i} et réduit le domaine de $x_i \in [z_g, z_d]$. BC3 utilise les fonctions LNAR, RNAR et NEWTON. Les deux fonctions LNAR et RNAR [31] permettent de calculer les zéros les plus à gauche et à droite d'une fonction univariable sur un domaine D (cf. algorithme 4.3). LNAR contracte dans un premier temps l'intervalle D en utilisant l'extension aux intervalles de la méthode de Newton (voir l'algorithme 4.4 et l'équation 4.10).

Il est possible que la fonction NEWTON ne réduise pas suffisamment le domaine D pour le rendre boite-consistant. Pour cette raison, on complète ce traitement par un processus de division de domaines [32]. On effectue alors la recherche d'un zéro dans la partie gauche et en cas d'échec, on le recherche dans la partie droite.

30. En mathématiques, un zéro ou point d'annulation d'une fonction est une valeur en laquelle cette fonction s'annule. Autrement dit, il s'agit d'un antécédent de la valeur zéro. (cf. wikipédia)

31. La fonction RNAR est définie de la même façon que la fonction LNAR, en remplaçant les éléments à gauche par les éléments à droite

32. Ce processus permet de trouver le zéro le plus à gauche même dans le cas d'une fonction non dérivable

Algorithm 4.4 NEWTON

Function : NEWTON(D, F_{x_i})

1: $DF \leftarrow DERIVE(F_{x_i})$
2: $D_2 \leftarrow D$
3: **repeat**
4: $\quad D_1 \leftarrow D_2$
5: $\quad D_2 \leftarrow D_2\ (center(D_2) - F_{x_i}(center(D_2))/DF(D_2))$
6: **until** $D_1 = D_2$
7: **return** D_2

Le rôle de la fonction SPLIT [33] de l'algorithme 4.3 est de calculer les deux plus grands sous-domaines de l'intervalle D (fourni en arguments) inférieur et supérieur au milieu de D alors que les fonctions $left$ et $right$ renvoient respectivement la borne gauche et la borne droite de l'intervalle (fourni en paramètre). La fonction "$DERIVE$" retourne la dérivée de la fonction prise en paramètre. La fonction "center" retourne le milieu de l'intervalle fourni en argument. Notons que l'algorithme BC3 associe à chaque contrainte utilisateur autant de fonctions univariables qu'il y a de variables dans cette contrainte.

4.5.3.2 BC4

HC4 et BC3 assurent deux types de consistances différents. HC4 est efficace sur les contraintes dont les variables sont mono-occurrentes. Il utilise des fonctions de projections ; ce qui est peu coûteux (de simples évaluations d'opération et des intersections sont réalisées). L'algorithme BC3, de son côté, est réduit les domaines même sur les contraintes avec des occurrences multiples. Cependant, chaque réduction est coûteuse à cause de la nécessité de rechercher les zéros les plus à gauche et à droite ; ce qui peut demander de nombreuses itérations de la fonction de Newton [34]. Ces remarques ont conduit à définir l'algorithme BC4 (Benhamou et al., 1999) qui combine BC3 et HC4. BC4 (voir l'algorithme 4.5) corrige les défauts de HC4 [35] et de BC3 [36]. Il adapte le calcul en fonction du nombre d'occurrences de variables dans la contrainte en cours de traitement. Il fait appel à HC4 pour réduire les domaines des variables qui apparaissent une seule fois dans une contrainte. Pour les variables multi-occurrentes, on utilise BC3.

L'algorithme 4.5 prend en entrée un ICSP sous la forme d'un ensemble de contraintes et d'une boîte contenant l'ensemble des domaines des variables. Cet algorithme réduit le domaine initiale jusqu'à ce qu'il ne soit plus possible de réduire. Expliquons quelques lignes importantes pour la compréhension :

La ligne 6 : $\mathcal{V}_c^1$ contient la liste des variables mono-occurrentes de la contrainte c

33. SPLIT divise le domaine D en deux intervalles D_{left} et D_{right}
34. Rappelons que la fonction de Newton est elle-même une méthode itérative.
35. problème de multiples occurrences de variables
36. le grand temps d'exécution

Algorithm 4.5 BC4

Function : BC4(C=$\{c_1, ..., c_m\}$, B=$I_1 \times ... \times I_n$)

▷ c_i : contraintes, B : une boîte

1: **repeat**
2: $B' \leftarrow B$
3: **repeat**
4: $NotFinished \leftarrow false$
5: **for each** $c \in C$ **do**
6: $\mathcal{V}_c^1 \leftarrow \{Var(c) | Multiplicity(x, c) = 1\}$
7: $B'' \leftarrow B$
8: $HC4revise(c,B)$
9: $NotFinished \leftarrow (B \neq \emptyset \wedge ((\exists I_k \neq I_k'' \wedge x_k \in \mathcal{V}_c^1) \vee NotFinished))$
10: **end for**
11: **until** $(NotFinished = false)$
12: **if** $B \neq \emptyset$ **then**
13: $BC3(C_p^+, B)$
14: **end if**
15: **until** $B' = B$ or $B = \emptyset$
16: **return** B

La ligne 8 : Appel de la méthode $HC4revise$ pour réduire les domaines des variables de la contrainte c

La ligne 9 : La valeur de $NotFinished$[37] est définie à VRAI si le domaine de B est vide (pas de solution) ou si le domaine I_k de la variable x_k mono-occurrente a été réduit par HC4revise.

Ligne 12 : A la fin de la réduction par $HC4Revise$, si B n'est pas vide[38]

Ligne 13 : Appel de l'algorithme BC3 sur C_p^+ et la boîte courante. Notons que C_p^+ contient la liste des contraintes multi-occurrentes[39] de C. Les contraintes de la liste C_p^+ sont écrites sous la forme de contraintes univariables[40].

4.5.4 kB-Consistance

Les techniques de consistance comme la consistance d'enveloppe ou la consistance de boîte sont souvent trop faibles pour atteindre une approximation fine des domaines des variables constituant la solution du problème. Notons que, pour les domaines finis, la faiblesse de la consistance d'arc (cf. Annexe E) avait conduit à proposer l'algorithme de consistance de chemin (cf. Annexe E). De même, les consistances plus fortes ont aussi été proposées dans les CSP à intervalles. (Lhomme, 1993) a ainsi proposé la notion de 3B-consistance généralisée à kB-consistance. Étant donné que 3B s'appuie sur un algorithme de 2B, kB-

37. *NotFinished* indique la fin de la réduction par HC4
38. donc il existe des solutions à l'ICSP
39. Contraintes contenant des variables à occurrences multiples
40. Dans ces contraintes, toutes les variables (mono-occurrentes ou multi-occurrentes) sont remplacées par leur valeur intervalle exceptée une. Exemple : La contrainte $[-1; 2]+x-2\times[0, 3] = 0$ est une contrainte univariable de $z + x - 2 \times y = 0$

P'	$y = 1$	$y = 1.5$
$(x + y = 2) \wedge (x = 0.5)$	faux	vrai
$(y \leq x + 1) \wedge (x = 0.5)$	vrai	vrai
$(y \geq x) \wedge (x = 0.5)$	vrai	vrai

P''	$y = 1$	$y = 1.5$
$(x + y = 2) \wedge (x = 1)$	vrai	faux
$(y \leq x + 1) \wedge (x = 1)$	vrai	vrai
$(y \geq x) \wedge (x = 1)$	vrai	vrai

Tableau 4.1 – Exemple de 3B-consistance

consistance s'appuie sur (k-1)B-consistance.

4.5.4.1 3B-Consistance

L'algorithme 3B (Lhomme, 1993) calcule la projection des ensembles de contraintes sur les variables et associe les contraintes en vue d'améliorer la précision de la réduction de domaines. Considérons P un ICSP, soit x une variable de P telle que $D_x = [a, b]$. Soient :

$P_{D_x^1 \leftarrow [a,a^+)}$, le CSP dérivé de P en remplaçant D_x par $D_x^1 = [a, a^+)$

$P_{D_x^2 \leftarrow (b^-,b]}$, le CSP dérivé de P en remplaçant D_x par $D_x^2 = (b^-, b]$

avec a^+étant la plus petite valeur réelle supérieur à a et b^- le plus grand nombre réel inférieur à b.

D_x est 3B-consistant si et seulement si $\Phi_{2B}(P_{D_x^1 \leftarrow [a,a^+)})$ et $\Phi_{2B}(P_{D_x^2 \leftarrow (b^-,b]})$ sont non vides avec $\Phi_{2B}(P)$ l'application d'une consistance de type 2B sur P. En d'autres termes, D_x est 3B-consistant si en appliquant la 2B-consistance (une consistance d'enveloppe) sur $P_{D_x^1 \leftarrow [a,a^+)}$ et $P_{D_x^2 \leftarrow (b^-,b]}$, on obtient un domaine de variable non vide. L'ICSP P est 3B-consistant si tous les domaines sont 3B-consistants. Par définition, lorsqu'un problème est 3B-consistant, il est aussi 2B-consistant. En guise d'exemple, considérons l'ICSP suivant :

$$\begin{cases} x + y = 2 \\ y \leq x + 1 \\ y \geq x \end{cases} \tag{4.11}$$

Avec $D_x = [0.5, 1]$, $D_y = [1, 1.5]$, le tableau 4.1 montre que le problème est 3B-consistant. Pour chaque instance x, nous avons un sous-problème 2B-consistant à résoudre. Sur chaque ligne du tableau 4.1, nous avons au moins une cellule à "vrai". Cela signifie que le domaine n'est pas vide. La même méthode est utilisée pour la variable y. Il en résulte que L'ICSP de l'équation 4.11 est aussi enveloppe-consistant. Pour décrire la 3B-consistance en d'autres termes, on peut dire qu'au lieu de projeter les contraintes une par une, elle projette l'ensemble des contraintes de l'ICSP. Par exemple, considérons deux variables x_k et y_k telles que $(x_k, y_k) \in [-2, 2] \times [-2, 2]$ avec les contraintes suivantes :

Algorithm 4.6 3B

Function : 3B(P)

```
1:  PF ← false
2:  while ¬PF do
3:      for i = 1..n do
4:          if ¬PF_{x_i} then
5:              S_{x_i} ← min(S_{x_i}, size(D_{x_i}))/2        ▷ Valeur initiale des S_{x_i}
6:              if S_{x_i} ≤ w then
7:                  PF_{x_i} ← true
8:                  S_{x_i} ← w
9:              end if
10:         end if
11:     end for
12:     D ← 3B–fixpoint(P, S)
13: end while
14: return P
```

$$\begin{cases} x_k + y_k &= 0 \\ x_k - y_k &= 0 \end{cases} \tag{4.12}$$

Les domaines $[-2, 2]$ sont consistants si on prend les contraintes une par une. On peut noter que pour $x_k = -2$, la première contrainte donne $y_k = 2$ et la seconde $y_k = -2$ qui sont des solutions encore pessimistes. 3B prend en compte les deux contraintes et finit par trouver la solution $[0, 0]$ qui est la solution la plus exacte. On notera que 3B se base sur un algorithme de faible niveau de consistance comme HC4, BC3 ou BC4 pour réduire les intervalles.

4.5.4.2 Description de l'algorithme 3B-Consistance

Notons que l'algorithme 3B repose sur une suite de tentatives de réfutation de certaines portions des domaines des variables. L'algorithme 3B (voir l'algorithme 4.6) prend en entrée un ICSP $P = (V, D, C)$ où V représente l'ensemble des variables, D, l'ensemble des domaines et C l'ensemble des contraintes. Il retourne un ICSP qui est 3B-consistant.

Dans les deux algorithmes 4.6 et 4.7, S est le vecteur des Δ utilisés pour la réduction des domaines. n est le nombre des variables du problème. PF est un vecteur de booléens qui indique si le point fixe est atteint pour chaque domaine. PF contient pour chaque variable x_i un membre PF_{x_i}. Nous présentons, ici, quelques lignes de l'algorithme 4.6 :

Ligne 5 : on calcule les Δ de chaque variable x_i en divisant soit la valeur courante par eux ou la taille du domaine initiale D_{x_i} par deux.

Ligne 7 : L'opération étant itérative, on compare chaque Δ c'est-à-dire chaque S_{x_i} à w (la taille minimale définie par l'utilisateur). w influence aussi la précision de la réduction.

Ligne 12 : Appel de 3B–fixpoint sur P et S.

Algorithm 4.7 3B–fixpoint

Function : 3B–fixpoint(P, S)

1: $PF \leftarrow false$
2: **while** $\neg PF$ **do**
3: $PF \leftarrow true$
4: **for** $i = 1..n$ **do**
5: $P_{borne_inf} = (V, \{D_{x_1}, ..., D_{x_{i-1}}, [a, a + S_{x_i}], D_{x_{i+1}}, ..., D_{x_n}\}, C)$
6: **if** $2B\text{–}filtering(P_{borne_inf}) = P_\emptyset$ **then**
7: $PF \leftarrow false$
8: $D_{x_i} \leftarrow D_{x_i} \backslash [a, a + S_{x_i}]$
9: $D \leftarrow 2B\text{–}filtering(P)$
10: **end if**
11: $P_{borne_sup} = (V, \{D_{x_1}, ..., D_{x_{i-1}}, [b-S_{x_i}, b], D_{x_{i+1}}, ..., D_{x_n}\}, C)$
12: **if** $2B\text{–}filtering(P_{borne_sup}) = P_\emptyset$ **then**
13: $PF \leftarrow false$
14: $D_{x_i} \leftarrow D_{x_i} \setminus [b - S_{x_i}, b]$
15: $D \leftarrow 2B\text{–}filtering(P)$
16: **end if**
17: **end for**
18: **end while**
19: **return** D

Le rôle de la fonction "3B–fixpoint" (algorithme 4.7) est de réduire les domaines des variables. Il prend en paramètre l'ICSP et le vecteur S des Δ à utiliser pour la réduction de chaque domaine du problème. Les Δ représentent les tailles à considérer pour des intervalles. Nous allons présenter, ici, quelques lignes de l'algorithme 4.7 :

Ligne 5 : P_{borne_inf} représente $P_{D_x^1}$ vu plus haut. Ici D_x^1 est remplacé par $[a, a + S_{x_i}]$

Ligne 6 : On teste si P_{borne_inf} est 2B-consistant. Sinon le domaine $[a, a + S_{x_i}]$ est éliminé (ligne 8) du domaine de la variable x_i

Ligne 11 : P_{borne_sup} représente $P_{D_x^2}$ vu plus haut. Ici D_x^2 est remplacé par $[b-S_{x_i}, b]$

Ligne 12 : On teste si P_{borne_sup} est 2B-cosnsitant. Sinon le domaine $[b-S_{x_i}, b]$ est éliminé (ligne 14) du domaine de la variable x_i Soit $D_x = [a, b]$ dans un ICSP P, si $\Phi_{3B}(P_{D_x \leftarrow [a, \frac{a+b}{2}]}) = \emptyset$, alors la portion $[a, \frac{a+b}{2})$ de D_x est éliminée et le filtrage se poursuit sur l'intervalle $[\frac{a+b}{2}, b]$; sinon le filtrage se poursuit sur l'intervalle $[a, \frac{a+b}{4}]$.

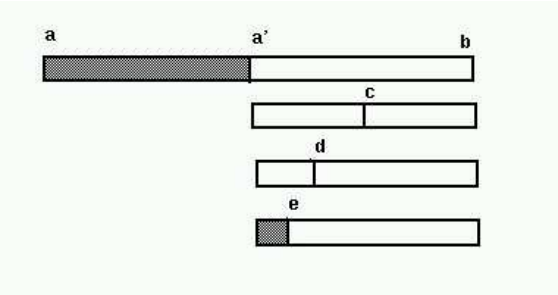


FIGURE 4.5 – Exemple de réduction de la borne gauche (Rueher, 2005)

La figure 4.5 présente les premières étapes d'une réduction d'un intervalle $[a, b]$ à partir de la borne de gauche :
$\Phi_{3B}(P_{D_x \leftarrow [a,a']}) = \emptyset$
$\Phi_{3B}(P_{D_x \leftarrow [a',a]}) \neq \emptyset$
$\Phi_{3B}(P_{D_x \leftarrow [a',d]}) \neq \emptyset$
$\Phi_{3B}(P_{D_x \leftarrow [a',e]}) = \emptyset$...
L'algorithme s'arrête lorsque le point fixe est atteint ou lorsque l'intervalle qu'on essaie de réfuter devient plus petit qu'une valeur absolue (ou relative) w fixée. Dans ce cas, on parle de 3B(w)-consistance. Il s'agit d'une approche dichotomique. La 3B ($w1$,$w2$)-consistance utilise $w1$ comme critère d'arrêt pour la division des domaines et $w2$ comme critère d'arrêt pour l'algorithme de 2B-consistance utilisé.

4.6 Les solveurs à intervalles

Pour simplifier la résolution des CSP, plusieurs outils logiciels ont été développés. Ces outils sont soit sous la forme de librairies à intégrer dans un programme soit des logiciels complets et autonomes. Ces outils sont souvent destinés à un certain type donné de CSP. Pour les CSP à intervalles (ICSP), des outils spécifiques ont été proposés. On notera qu'un solveur résout un ICSP de manière déclarative, l'utilisateur ne fournissant que le modèle. Idéalement, la modélisation est donc à la charge de l'utilisateur et la résolution est à la charge du solveur. Dans cette section, nous présenterons quelques solveurs à intervalles et nous détaillerons les raisons de notre choix.

4.6.1 Numerica

Proposé par Van Hentenryck et al. (Van Hentenryck et al., 1997), ce solveur permet de minimiser les fonctions non linéaires avec ou sans contraintes. Ce solveur est disponible en librairie C++ dans Ilog Solver (IlcNum).

4.6.2 RealPaver

Il a été conçu dans les années 90 par Granvilliers et Benhamou (Granvilliers and Benhamou, 2006) et est spécialisé dans la résolution de systèmes d'équations non linéaires utilisant des méthodes à intervalles.

4.6.3 ALIAS

ALIAS[41] (Merlet, 2000) est le solveur utilisé par le projet COPRIN[42](Merlet, 2001) pour les applications robotiques. Il est dédié à la résolution de systèmes d'équations utilisant des opérateurs issus de l'analyse par intervalles et la propagation de contraintes. La version Maple d'Alias est le premier solveur intégrant les méthodes intervalles et les techniques de calcul symbolique.

4.6.4 ICOS

Icos[43] (Lebbah, 2009) est spécialisé dans l'optimisation globale robuste et sous contraintes. Cette librairie contient certains algorithmes issus de la programmation par contraintes, l'analyse par intervalles et les techniques de relaxation linéaire.

Il contient aussi une interface pour certains solveurs de programmation linéaire et d'optimisation locale tels que Coin/Clp, Cplex et IpOpt.

4.6.5 GloptLab

Développé sous Matlab par Domes, GloptLab[44] (Domes, 2009) (Domes and Neumaier, 2007) est spécialisé dans la résolution de systèmes quadratiques. Il implémente plusieurs algorithmes de résolution de CSP : propagation de contraintes, relaxation linéaire, les encadrements convexes et les méthodes coniques.

4.6.6 Ibex

Ibex[45] (Chabert, 2007) a été développé par Gilles Chabert et se limite à la résolution de système d'équations. Depuis 2008, le langage Quimper(Jaulin, 2009) a été implémenté sur Ibex, permettant ainsi aux ingénieurs non-spécialistes d'utiliser facilement la librairie. Quimper est un ensemble de règles pour la rédaction d'un CSP.

4.6.7 Choix de solveur

RealPaver permet de modéliser et résoudre les ICSP non linéaires. Il réalise des approximations fiables des ensembles solutions continus ou discrets en utilisant les produits cartésiens des intervalles.

Les ICSP sont fournis sous forme d'ensembles d'équations ou d'inéquations sur des variables entières et réelles. En outre, ils peuvent être de nature différente, étant carrés ou non, linéaires, polynomiales ou faisant intervenir des fonctions trigonométriques. Des exemples de comparaison sur des problèmes de programmation linéaire ont été réalisés par (Keil, 2007) et (Keil, 2008). Le langage de

41. ALIAS : Algorithms Library of Interval Analysis for Systems
42. COPRIN : Contraintes, OPtimisation, Résolution par INtervalles
43. ICOS : Interval COnstraint Solver
44. GloptLab : Global Optimization Laboratory
45. Ibex est mis pour Interval-Based EXplorer

modélisation permet d'indiquer des modèles de contraintes et des paramètres de configuration des algorithmes de résolution qui combinent les méthodes intervalles pour résoudre les ICSP. La plupart des techniques de consistance (box, hull, 3B) sont implémentées dans RealPaver.

De plus, Realpaver a déjà été utilisé pour résoudre rigoureusement des problèmes continus et non linéaires dans les domaines du contrôle automatique, d'aide à la conception et de robotique. Ainsi (Berger et al., 2010) ont utilisé Realpaver, pour déterminer les erreurs en position et rotation des effecteurs d'un système robotique (un bras manipulateur). Ils ont modélisé les erreurs de prédiction en formulant deux ICSP qu'ils ont résolus en utilisant des algorithmes fournis dans Realpaver. Nicolas Berger dans (Berger et al., 2009) a aussi utilisé Realpaver pour résoudre des problèmes spécifiques de type MINLP (Mixed Integer Nonlinear Programming) de l'optimisation globale.

Luc Jaulin dans (Luc Jaulin, 2002) s'est aussi servi de RealPaver pour montrer la possibilité d'utiliser les techniques de consistance pour estimer la position d'un satellite en orbite autour de la terre (un unique CSP a été formalisé et résolu). L'approche a été utilisée pour la résolution de problèmes d'estimation ensembliste et de commande robuste. Ce solveur fournit des algorithmes de consistance locale tels que :

- HC3 et HC4 pour la consistance d'enveloppe
- BC3 pour la consistance de boite
- BC4 pour la combinaison HC4 et BC3

Pour la consistance forte, l'algorithme 3B ainsi qu'une version plus légère (weak3B) sont fournis. En natif, l'utilisation de ce solveur nécessite de définir les ICSP à résoudre sous le format suivant :

```
Variables
x1 in [-2,5.57] , x2 in [-6.25,1.3], x3 in [-5.39,0.7] ,
y1 in [-5.57,2.7], y2 in [-6.25,2.7], y3 in [-5.39,3.11],
z1 in [0,5.57] , z2 in [-2,6.25] , z3 in [-3.61,5.39] ;
Constraints
x1^2 + y1^2 + z1^2 = 31,
x2^2 + y2^2 + z2^2 = 39,
x3^2 + y3^2 + z3^2 = 29,
x1*x2 + y1*y2 + z1*z2 + 6*x1 - 6*x2 = 51,
x1*x3 + y1*y3 + z1*z3 + 7*x1 - 2*y1 - 7*x3 + 2*y3 = 50,
x2*x3 + y2*y3 + z1*z3 + x2 - 2*y2 - x3 + 2*y3 = 34,
-12*x1 + 15*y1 - 10*x2 - 25*y2 + 18*x3 + 18*y3 = -32,
-14*x1 + 35*y1 - 36*x2 - 45*y2 + 30*x3 + 18*y3 = 8,
2*x1 + 2*y1 - 14*x2 - 2*y2 + 8*x3 - y3 = 20 ;
```

4.7 Conclusion

Dans le cadre de cette thèse, nous considérons le problème de la localisation de véhicule comme un problème de satisfaction de contraintes à intervalles étant

donné que les domaines des variables prises en compte sont continus et représentés comme des intervalles de $\mathbb{R}$.

Pour faciliter la compréhension du reste du document, ce chapitre introduit la notion de CSP dans un le contexte général puis dans le contexte particulier des CSP à intervalles. Les algorithmes de résolution de CSP les plus connus sont aussi abordés. Ces algorithmes peuvent être soit reprogrammés, soit utilisés directement en faisant appel à des solveurs à intervalles qui les intègrent. Nous avons présenté les solveurs à intervalles. Plusieurs solveurs ont fait l'objet de notre attention. Le souci de comparer plusieurs algorithmes nous a conduit à en retenir un : RealPaver.

Chapitre 5

Localisation multi-sensorielle garantie

Sommaire

5.1 Introduction 84
5.2 Les différents types de localisation 85
5.2.1 Suivi de position 85
5.2.2 Localisation globale 86
5.3 Caractéristiques des environnements 86
5.3.1 Environnement d'intérieur ou d'extérieur 86
5.3.2 Environnement statique ou dynamique 87
5.4 Autres aspects du problème de localisation 87
5.5 Les approches de localisation en robotique mobile . 89
5.5.1 Les approches probabilistes 89
5.5.2 Les approches ensemblistes 92
5.6 Application des algorithmes ICP à la localisation . . 94
5.6.1 Problématique 94
5.6.2 Approches possibles 96
5.6.3 Processus de localisation 97
5.6.4 Résultats 100
5.6.5 Conclusion 105
5.7 Proposition de correction du cap du véhicule 106
5.7.1 Présentation du problème 107
5.7.2 Formulation du problème de localisation sous forme de νDICSP 110
5.7.3 Résultats 114
5.7.4 Comparaison d'ICPS avec l'EKF 118
5.7.5 Conclusion 120
5.8 Conclusion 120

5.1 Introduction

Un véhicule autonome doit pouvoir naviguer dans son environnement de façon autonome. La réussite du processus de navigation nécessite celle de quatre autres tâches :

- la perception, le véhicule doit pouvoir extraire des informations pertinentes de ses capteurs et les interpréter
- la localisation, il doit pouvoir calculer instantanément sa pose (position et orientation) dans son environnement
- la cognition, il doit pouvoir choisir les actions à entreprendre en vue d'atteindre ses objectifs
- le contrôle, il doit être en mesure de gérer ses effecteurs afin de suivre une trajectoire souhaitée par exemple.

La recherche s'est beaucoup intéressée ces dernières décennies à la tâche de localisation. Et plusieurs avancées ont été réalisées dans ce sens.

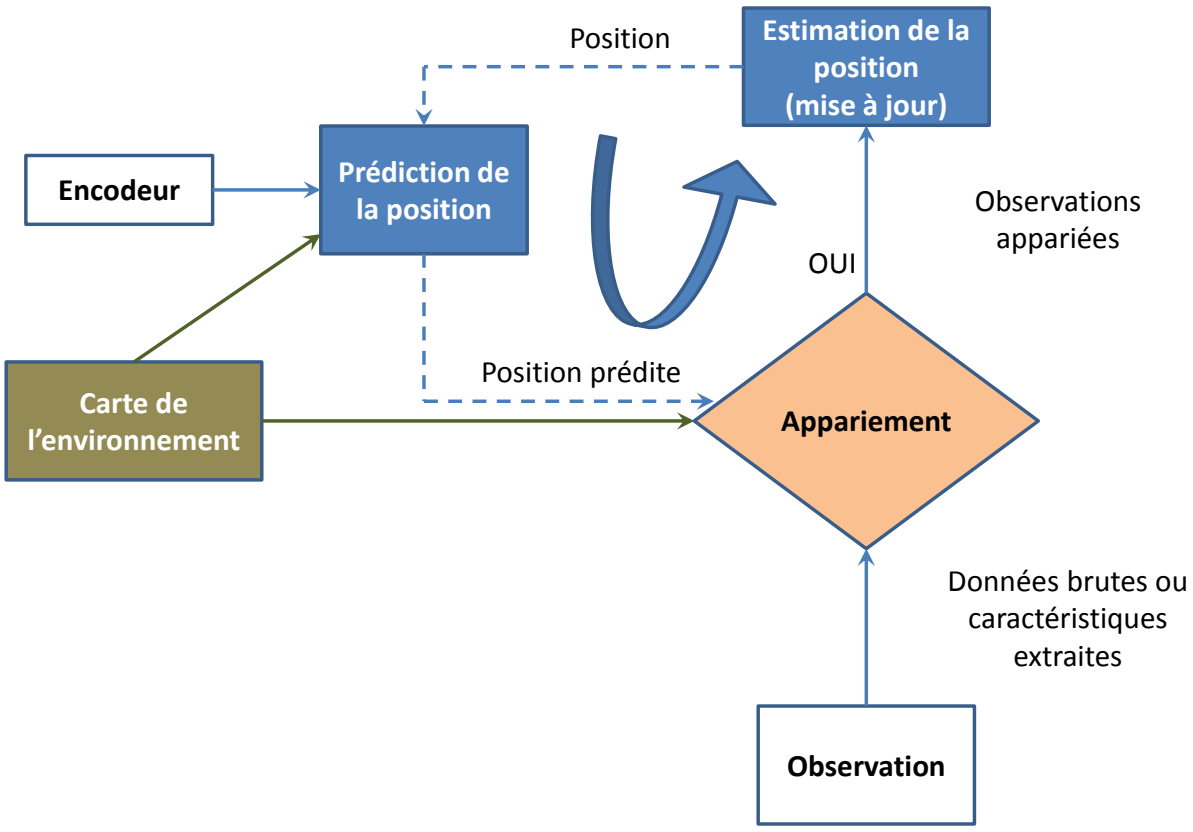


FIGURE 5.1 – Processus classique de localisation (par carte)(Siegwart et al., 2011)

Hélas, à ce stade de la technologie, le GPS n'est précis qu'au mètre voire à une dizaine de mètres près. De plus, il ne peut pas fonctionner dans certains endroits (zones de masquage du GPS, par exemple).

La figure 5.1 présente un processus classiquement utilisé pour la localisation basée sur la connaissance a priori du modèle de l'environnement. L'étape de prédiction prend en compte des données proprioceptives pour définir les positions possibles du véhicule dans la carte. L'étape d'estimation calcule la position la plus probable du véhicule en se basant sur des données provenant d'un capteur extéroceptif, la position prédite et la carte de l'environnement. Une description détaillée des différentes techniques de localisation par carte a été publié par Filliat (Filliat and Meyer, 2003). (Gruyer et al., 2014) propose des méthodes de résolution de conflits dans le cadre de l'utilisation d'une carte.

Dans ce chapitre, diverses situations de localisation seront présentées dans la section 2. Dans la troisième section, nous aborderons, les différents types d'environnement parcourus par les véhicules autonomes. La quatrième section sera consacrée à la présentation des différentes approches de localisation notamment, les approches probabilistes et les approches ensemblistes. Dans la section 5, nous présenterons la première contribution de cette thèse. Et dans la section 6 la deuxième contribution sera abordée. Elle concerne la correction du cap du véhicule par les méthodes ensemblistes.

5.2 Les différents types de localisation

Estimer la position d'un système mobile autonome est d'une importance capitale dans bon nombre d'applications automobiles. Il s'agit d'un problème d'estimation d'état d'un système dynamique non linéaire.

Plusieurs méthodes existent pour résoudre le problème de localisation. Elles varient suivant le type des capteurs utilisés, le type d'environnement ou des considérations d'ordre purement technique comme les ressources systèmes disponibles, par exemple. Le terme de localisation regroupe plusieurs capacités en robotique mobile notamment, le suivi de position, la localisation globale ou encore la détection d'une situation de kidnapping.

5.2.1 Suivi de position

Dans le cadre du suivi de pose, on considère connue une estimation de la position du véhicule. Il s'agit de mettre à jour cette position en prenant en compte les nouvelles perceptions de l'environnement ou les données proprioceptives captées au cours du déplacement. On procède donc de manière itérative en partant de la position précédente du véhicule et en calculant sa position courante. On parle aussi de localisation locale étant donné que la position courante est estimée localement par rapport à la position précédemment évaluée.

Le suivi de position souffre, en général, des imprécisions des mesures capteurs (essentiellement, les mesures du déplacement). Ce problème conduit à des dérives du véhicule. Ces imprécisions s'amplifient et s'accumulent tout au long du trajet du véhicule. Pour contourner ce problème, on a généralement recours à des données de capteurs extéroceptifs ou à des modèles (cartes) de l'environnement pour corriger et affiner l'estimation de la position du véhicule.

En pratique, le fait que le suivi de position dépende d'une estimation initiale de la pose réalisée, est problématique. En effet, si cette position est trop différente de la position réelle, on peut se trouver dans l'incapacité de localiser le véhicule.

5.2.2 Localisation globale

Contrairement au suivi de position, aucune position initiale n'est nécessaire pour la localisation globale. En effet, le calcul de la position courante ne nécessite aucunement la connaissance de la position précédente. Le véhicule peut se localiser de façon autonome sans intervention extérieure préalable et ainsi définir sa position en toute situation. Le véhicule peut être mis hors tension et remis à un emplacement quelconque de son environnement sans qu'il soit nécessaire de lui indiquer sa position lors de la remise sous tension. La position est définie de façon globale dans l'environnement.

On se base sur l'utilisation de capteurs extéroceptifs. Et, on utilise soit des points de repère naturels, soit des points de repère artificiels. Notons que la localisation globale nécessite dans certains cas un modèle de l'environnement (un ensemble d'informations *a priori* regroupant les caractéristiques sur l'environnement).

On peut facilement imaginer que le véhicule soit kidnappé alors qu'il se déplace dans son environnement et être positionné à un emplacement quelconque de l'environnement, sans pour autant l'indiquer à son système de positionnement. Une des difficultés de cette situation est que le véhicule doit détecter l'erreur de position (reconnaître la différence entre la position courante et celle dans laquelle il estime être).

Dans des travaux comme (Filliat et al., 2002), on vise la localisation globale. En effet, (Filliat et al., 2002) utilisent un sonar et des capteurs visuels (embarqués sur un robot) pour construire une carte de l'environnement. A chaque instant, le système de navigation est capable de localiser globalement le robot c'est-à-dire d'estimer sa position même s'il a été déplacé d'un endroit à l'autre de la carte.

5.3 Caractéristiques des environnements

Les environnements considérés dans le cadre de la localisation de véhicule sont de différents types. Ils peuvent être structurés ou non, d'intérieur ou d'extérieur, statiques ou dynamiques, symétriques ou asymétriques, etc.

5.3.1 Environnement d'intérieur ou d'extérieur

Environnement d'intérieur Nous faisons, ici, référence à tout espace situé à l'intérieur d'un bâtiment. On peut considérer dans cette catégorie l'intérieur d'un immeuble ou appartement. Une usine dans laquelle est à l'œuvre un robot comme LittleHelper (Hvilshj et al., 2009; Hvilshøj and Bøgh, 2011) est aussi un exemple d'environnement d'intérieur. Les odomètres, les télémètres laser, les sonars et les caméras sont souvent utilisés dans ce genre d'environnement. D'autres capteurs ne sont pas adaptés. Par exemple, les récepteurs GPS ne sont pas d'une grande

utilité dans ce genre d'environnement étant donné la difficulté à capter les signaux d'une part et la précision nécessaire d'autre part.

Environnement d'extérieur Il s'agit de tout milieu extérieur à des bâtiments. On peut ainsi penser à la localisation des véhicules aériens (drone) ou celle des robots sous-marins ou encore des voitures en pleine circulation. Ce type d'environnement est de taille plus grande que l'environnement d'intérieur. On y utilise des capteurs GNSS comme le GPS. D'autres capteurs comme les télémètres laser ou les caméras (mono-vision ou stéréo-vision) sont aussi mis à profit pour localiser un véhicule dans ce type d'environnement. Cependant, des capteurs de type sonars ne sont pas très adaptés. Nos travaux concernent un environnement d'extérieur, même s'ils peuvent être adaptés à un environnement d'intérieur.

5.3.2 Environnement statique ou dynamique

Environnement statique Un environnement est considéré comme statique s'il ne subit aucun changement au cours du temps. Même s'ils ne sont réalistes que dans des cas rares, ce genre d'environnement est souvent considéré comme hypothèse de base et permet de faire les premiers tests des algorithmes destinés à évoluer afin de supporter d'autres types d'environnement. (Leonard and Durrant-Whyte, 1991) travaillent dans un environnement statique.

Environnement dynamique Les environnements qui changent au cours du temps sont dits "dynamiques". Ainsi un supermarché avec des clients allant et venant dans tous les sens ou une route aux heures de pointe en sont des exemples.

Le tableau 5.1 de (Aldon, 2001) fait le lien entre les différentes approches de localisation, les différents types d'environnements, localisation relative ou absolue ainsi que les capteurs utilisés pour les mettre en œuvre.

5.4 Autres aspects du problème de localisation

On peut aussi voir le problème de localisation sous d'autres angles. Par exemple, la localisation peut être active ou passive, multi-véhicules ou mono-véhicule.

Localisation passive

Dans le cas de la localisation passive, le mobile ne fait qu'observer et produire des informations sur le positionnement. Le module de localisation ne contrôle pas le véhicule.

Localisation active

Dans ce cas, le module de localisation contrôle le véhicule. Il a ainsi la capacité de réduire les erreurs de localisation.

Technique	**Type de localisation**	**Environnement - Mise en œuvre**	**Performances**
Odométrie (sur roues motrices)	*Relative* : mesure de position et de cap	Intérieur et extérieur sur terrain plat - Simple	Forte dérive. Erreurs importantes sur sol irrégulier
Gyroscope à fibre optique	*Relative* : mesure de cap	Indifférent - Simple	Dérive faible (quelques degrés par heure)
Inclinomètres	*Absolue* : mesure de tangage et de roulis	Sol lisse Mouvement uniforme - Simple	Bonnes en l'absence d'accélérations parasites Erreur $< 0{,}1°$
Compas magnétique	*Absolue* : mesure de cap	Extérieur. Milieu dépourvu de perturbations magnétiques - Simple	Erreur : quelques dixièmes de degré
Centrale inertielle d'attitude	*Relative* : mesure des trois angles d'attitude	Indifférent - Simple	Précision : quelques dixièmes de degré Faible dérive : quelques degrés par heure
GPS	*Absolue* : mesure de position (x,y,z)	Extérieur - Simple	Précision : quelques cm à quelques mètres en mode différentiel
Localisation 2D sur balises optiques artificielles	*Absolue 2D* : mesure de position et de cap	Intérieur - Installation et modélisation du champ de balises	Erreur : 1 à 5 cm en position et $< 1°$ en orientation
Localisation 3D sur balises optiques artificielles	*Absolue 3D* : mesure de position (x,y,z) et d'attitude (ϕ,ψ,θ)	Extérieur - Installation et modélisation du champ de balises	Précision : fonction de la distance des balises
Mise en correspondance de cartes	*Absolue* : mesure de position et de cap	Intérieur Milieu relativement structuré - Acquisition préalable d'une carte de référence	Erreur : $<$ 20 cm en position et $< 5°$ en orientation
Vision dynamique	*Absolue ou relative 3D* : mesure à un facteur d'échelle près	Indifférent - Algorithmes complexes. Temps de calcul élevé	
Stéréo-vision	*Absolue ou relative 3D*	Indifférent - Algorithmes complexes. Temps de calcul élevé	

Tableau 5.1 – Caractéristiques des différentes techniques de localisation (Aldon, 2001)

Algorithm 5.1 Localisation Markovienne

Function : Localisation_Markov($bel(\mathcal{X}_{t-1})$, u_t, z_t, C)

1: **for each** $\mathcal{X}_t$ **do**
2: $\quad \overline{bel}(\mathcal{X}_t) = \int p(\mathcal{X}_t|u_t, \mathcal{X}_{t-1}, C)bel(\mathcal{X}_{t-1})d\mathcal{X}_{t-1}$; ▷ Prédiction
3: $\quad bel(\mathcal{X}_t) = \eta p(z_t|\, \mathcal{X}_t, C)\overline{bel}(\mathcal{X}_t)$; ▷ Estimation
4: **end for**
5: **return** $bel(\mathcal{X}_t)$

Localisation multi-véhicules

On se réfère à un groupe de véhicules qui communiquent entre eux. Dans un premier temps, chaque véhicule se localise. Ensuite, lorsque deux véhicules se détectent l'un et l'autre, ils peuvent échanger leurs informations de localisation. Par exemple, (Rohani et al., 2013) traitent de la localisation collaborative entre véhicules.

Localisation mono-véhicule

Un seul véhicule est considéré. Il collecte les données pour se localiser à l'aide des capteurs embarqués. C'est le cas le plus fréquent dans la littérature. Il n'y a aucune communication. Dans notre travail, nous nous intéressons à ce cas.

5.5 Les approches de localisation en robotique mobile

Plusieurs approches sont utilisées pour localiser un véhicule dans les différentes situations présentées dans la section précédente ainsi que dans les différents types d'environnement. Le suivi de position et la localisation globale sont les problèmes classiquement traités. L'environnement dynamique est souvent difficile à gérer, le but étant toujours de répondre à la question "Où suis-je ?". Dans cette section, différentes approches seront abordées : probabilistes et ensemblistes. Notons que beaucoup d'autres techniques ont été utilisées avec plus ou moins de succès notamment la théorie des possibilités (Maaref et al., 1999).

5.5.1 Les approches probabilistes

La plupart des méthodes de localisation sont basées sur des approches probabilistes et utilisent la loi de Bayes. Nous abordons, dans cette section, les plus connues de ces méthodes, sans toutefois chercher à être exhaustif.

5.5.1.1 Localisation Markovienne

Lorsque le filtre Bayésien est appliqué directement à un problème de localisation, on parle de localisation Markovienne. L'algorithme 5.1 présente cette méthode. On peut remarquer que cet algorithme prend en entrée une croyance

sur l'état précédent du véhicule en plus d'une carte (et d'autres paramètres) et retourne en sortie de traitement la croyance courante $bel(\mathcal{X}_t)$. $bel(\mathcal{X}_t)$ représente la densité de probabilité de la position courante $(\mathcal{X}_t)$ du véhicule à l'instant t.

La plupart des implémentations de l'approche bayésienne sont basées sur le filtre de Kalman et son extension appelée EKF [46] ou filtre de Kalman étendu. Le principe de cette méthode est détaillé en annexe C. Les approches probabilistes ont été utilisées pour résoudre les différents types de problèmes de localisation précités.

Dans la plupart des cas, on connaît la position initiale du véhicule et cette position est mise à jour tout au long de son déplacement à l'aide des données provenant des capteurs proprioceptifs. Et les situations de dérive sont tout naturellement observées étant donnés que les erreurs liées à l'odométrie par exemple s'accumulent au cours du déplacement.

Pour compenser ces erreurs qui sont sources de dérive du véhicule, on fait intervenir des capteurs extéroceptifs et on a recours à une technique de fusion de données pour avoir des informations cohérentes et exploitables.

Dans une expérience grandeur nature, on ne connaît jamais complètement la position initiale du véhicule. Cette position fait l'objet d'une approximation. Pour représenter cette approximation, les approches probabilistes fournissent une Gaussienne centrée sur la position estimée $\overline{\mathcal{X}_0}$. On obtient ainsi la croyance de la position initiale sous la forme de l'équation 5.1 :

$$bel(\mathcal{X}_0) \sim \mathcal{N}\left(\overline{\mathcal{X}_0}, \sum{}_0\right) \tag{5.1}$$

$bel(\mathcal{X}_0)$ est la densité de probabilité correspondant à la connaissance de la position initiale $\mathcal{X}_0$ du véhicule. $\sum_0$ la matrice de covariance de l'incertitude sur la pose initiale. (Jensfelt and Christensen, 2001), (Ivanjko and Petrovic, 2004) et (Lingemann et al., 2005) ont utilisé l'EKF pour traiter le problème de suivi de position.

Lorsque le problème de localisation se pose en termes de localisation globale (ici, on part du principe que la position initiale n'est pas connue), deux autres approches probabilistes sont souvent utilisées. Il s'agit de la localisation Monte Carlo (MCL [47]) et de la localisation par grille.

5.5.1.2 La localisation par grille

La localisation par grille (utilisée par (Burgard et al., 1999)) consiste à décomposer l'espace d'états en un ensemble de cellules constituant une grille, puis à attribuer à chaque cellule la probabilité que le véhicule soit dans la configuration correspondante. On définit donc la croyance $bel(\mathcal{X}_t)$ par :

$$bel(\mathcal{X}_t) = \{P_{k,t}\}$$

où $P_{k,t}$ représente la probabilité qu'à l'instant t le véhicule soit dans la configuration qui correspond à la $k^{ième}$ cellule de la grille. Dans certains cas, la grille

46. EKF : Extended Kalman Filter
47. Monte Carlo Localization

Algorithm 5.2 Localisation Monte Carlo

Function : Localisation_Monte_Carlo($\mathcal{X}_{t-1}$, u_t, z_t, C)

1: $\overline{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$;
2: **for** $m = 1 : M$ **do**
3: générer $x_t^{[m]} \sim p(\mathcal{X}_t | u_t, x_{t-1}^{[m]})$;
4: $w_t^{[m]} = p(z_t | x_t^{[m]}, C)$;
5: $\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + \prec x_t^{[m]}, w_t^{[m]} \succ$;
6: **end for**
7: **for** $i = 1 : M$ **do**
8: générer $x_t^{[i]}$ en fonction de $\overline{\mathcal{X}}_t$
9: $w_t^{[i]} = p(z_t | x_t^{[i]})$;
10: ajouter $x_t^{[i]}$ à $\mathcal{X}_t$
11: **end for**
12: **return** $\mathcal{X}_t$

est définie de façon statique et les cellules gardent la même taille tout au long du traitement. La taille des cellules définit la précision de la grille et donc le résultat de la localisation. Cependant, la recherche de la plus grande précision conduit à un temps de calcul exorbitant. Pour cette raison, l'usage des grilles adaptatives est souvent proposé.

On initialise la croyance $bel(\mathcal{X}_0)$ à une probabilité identique ($\frac{1}{N}$, N étant le nombre de cellules) pour toutes les cellules. On peut noter que la connaissance d'un modèle de l'environnement peut être utile pour améliorer l'initialisation des valeurs des cellules.

5.5.1.3 La méthode Monte Carlo

La méthode Monte Carlo utilise un ensemble de particules pour représenter la position du véhicule. Un ensemble de M particules définit la croyance $bel(\mathcal{X}_t)$ avec :

$$\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]}, x_t^{[3]}, ..., x_t^{[M]}\}$$

Chacune des positions possibles du véhicule est représentée par une particule.

On attribue à chaque particule une valeur qui représente le poids correspondant à la probabilité qu'elle (cette particule) contienne la position du véhicule. Cette méthode est présentée dans l'algorithme 5.2. L'algorithme du filtre particulaire constitue la base de la méthode Monte Carlo (Fox et al., 1999), (Thrun et al., 2001), (Fu et al., 2011).

Pour définir la croyance initiale, un jeu de particules est généré de façon aléatoire. Et le poids est identique pour chaque particule et de valeur $\frac{1}{M}$, M étant le nombre de particules.

5.5.2 Les approches ensemblistes

Parmi les alternatives aux approches probabilistes, figurent les approches ensemblistes. Et contrairement aux approches probabilistes qui représentent la position du véhicule à l'aide des densités de probabilité, les approches ensemblistes quant à elles utilisent les vecteurs d'intervalles pour y arriver. Elles ont en plus l'avantage d'être robustes et garanties. La boîte formée par le vecteur intervalle contient de façon certaine la position du véhicule. Cependant, on ne peut privilégier une partie de la boîte au détriment d'une autre. Cela n'a aucun sens pour ces approches contrairement aux approches probabilistes qui privilégient le maximum de la densité de probabilité de la croyance comme étant l'état le plus probable correspondant à la position du véhicule. Le but des approches ensemblistes est de calculer la plus petite boîte (vecteur intervalle) $[\mathcal{X}_t]$ qui contient de façon certaine à l'instant t, la position $\mathcal{X}_t$ du véhicule.

L'hypothèse est que les erreurs sont bornées et les bornes sont connues. Cette hypothèse nous conduit à attribuer aux vecteurs de mesures z_t le vecteur d'intervalles $[z_t]$ et au vecteur de commande u_t le vecteur d'intervalles $[u_t]$. En considérant $[f]$, la fonction d'inclusion de la fonction réelle f qui définit le modèle d'évolution du véhicule, on peut écrire :

$$[\mathcal{X}_{t+1}] = [f]([\mathcal{X}_t], [u_t]) \tag{5.2}$$

avec $\mathcal{X}_{t+1}$ représentant la position du véhicule à l'instant $t+1$ alors que $\mathcal{X}_t$ représente la position à l'instant t.

Pour la localisation de véhicules, deux approches basées sur l'analyse par intervalles sont souvent utilisées dans la littérature. Il s'agit de l'approche basée sur SIVIA et l'approche basée sur la propagation de contraintes sur les intervalles. Ces deux approches font l'objet des sous-sections qui suivent.

5.5.2.1 Approche basée sur SIVIA

Cette approche utilise l'algorithme SIVIA[48] qui est un algorithme proposé par (Jaulin and Walter, 1993; Jaulin, 2001). SIVIA est une forme particulière des algorithmes dits de "Interval Branch And Bound". Les approches "Interval Branch And Bound" sont décrites dans l'annexe B de ce document. Le problème de localisation de véhicule, dans ce contexte, est traité comme un problème d'inversion ensembliste (détaillé en annexe A).

Il s'agit de définir d'abord, à chaque instant t, un pavé initial contenant toutes les poses possibles du véhicule. Ensuite, on procède par bissections successives. On obtient ainsi un sous-pavage (la notion de sous-pavage est présentée dans la section 3.5). Chaque pavé du sous-pavage fait l'objet d'un test d'inclusion. Le résultat obtenu est ensuite regroupé avec l'algorithme "ImageSP"[49](Jaulin, 2001). Le but d'ImageSP est de calculer un sous-pavage régulier à partir d'un sous-pavage

48. SIVIA : Set Inverter Via Interval Analysis (en français : Inversion ensembliste avec de l'analyse par intervalles)

49. ImageSP signifie "Image de sous-pavage"

qui lui est fourni en entrée. (Seignez et al., 2005) utilise ces algorithmes dans le cadre de la localisation de véhicule (la technique se nomme BESE [50]).

5.5.2.2 Approche basée sur la propagation de contraintes

Dans le domaine de la localisation, la propagation de contraintes sur les intervalles a été introduite par (Gning and Bonnifait, 2004) il y a une dizaine d'années. Il s'agit de formuler le problème de localisation comme un problème de satisfaction de contraintes. Le but est de contracter une boîte contenant la position du véhicule de manière à obtenir la plus petite boite cohérente avec les mesures et la position prédite. Plus la boîte est petite et plus précise sera la connaissance de la position du véhicule. Plusieurs travaux ont suivi cette approche notamment en robotique d'intérieur (Gning and Bonnifait, 2004), en environnement d'extérieur (Vincke et al., 2010; Vincke and Lambert, 2009) et pour la robotique sous-marine (Jaulin, 2006).

L'objectif de (Gning and Bonnifait, 2004) a consisté à déterminer comment les techniques de propagation sur les intervalles peuvent être utilisées pour réaliser la fusion de données. Pour cela, ils ont développé une approche pour la fusion de données basée sur :

- 4 encodeurs de roue
- une mesure d'angle de la roue motrice
- un récepteur différentiel GPS

L'algorithme utilisé est de type Forward-Backward. Deux niveaux de fusion sont utilisés : la fusion statique et la fusion dynamique.

- La fusion statique utilise le modèle d'Ackerman (Ackermann and Bünte, 1997) et implémente la propagation Forward-Backward. On arrive à obtenir l'estimation des déplacements élémentaires δs, $\delta\theta$.
- La fusion dynamique est basée sur la propagation Forward-Backward et fournit la position du mobile.

L'implémentation a été réalisée sous MATLAB. Les résultats ont permis de démontrer que la propagation par contraintes sur les intervalles peut être vue comme une alternative aux approches probabilistes afin de localiser un véhicule. La fonction réalisée est capable de contracter les boîtes issues de l'observation de l'environnement. En d'autres termes, cette fonction fournit des quantités non directement observées et qui contiennent de façon certaine la position du véhicule.

(Vincke et al., 2010) ont apporté des améliorations au travail précédent. Ils ont utilisé :

- 2 odomètres
- un gyromètre
- un récepteur GPS

L'algorithme utilisé est aussi le Forward-Backward avec deux niveaux de fusion : la fusion statique et la fusion dynamique. La différence avec le travail précédent (Gning and Bonnifait, 2004) se trouve au niveau du modèle utilisé pour la phase de fusion statique. L'algorithme a été nommé SDF pour "Static and Dynamic Fusion". Cet algorithme a été implémenté en C++ avec Profil/Bias. Les résultats

50. BESE pour "Bounded Error Set Estimation"

ont permis de comparer d'une part, les boîtes résultant de SDF et de CP[51] (algorithme de base) et d'autre part les intervalles d'erreurs en x et en y de SDF, CP et des données GPS.

(Delafosse et al., 2004) ont aussi présenté le problème de la localisation sous forme de CSP. Et ils ont utilisé la propagation de contraintes sur les intervalles pour résoudre ce CSP et donc localiser le robot. La carte 2D de l'environnement était connue. Les valeurs imprécises ont été représentées sous la forme d'intervalles. Les données sont fusionnées par propagation de contraintes sur les intervalles. Ils ont utilisé le même modèle d'odométrie que (Gning and Bonnifait, 2004), pour la prédiction et la correction des états estimés. Ils ont utilisé :

- 2 odomètres et
- un capteur extéroceptif composé de
  - 1 télémètre laser et
  - 1 miroir conique SYCLOP qui est un capteur de vision omnidirectionnelle

Les coordonnées polaires des primitives perçues sont exprimées sous forme de 2 intervalles $[d]$ et $[theta]$. En outre, 2 odomètres donnent les informations sur la position du robot.

5.6 Application des algorithmes ICP à la localisation

Dans cette section, la décomposition des contraintes en contraintes binaires sera présentée. Puis seront testés plusieurs algorithmes qui feront l'objet d'une comparaison sur le plan de la précision des résultats ainsi que celui du temps de calcul pour la localisation d'un véhicule.

5.6.1 Problématique

Les travaux cités dans la sous-section 5.5.2.2 sont basés sur la technique de propagation Forward-Backward et suivent le principe de l'algorithme de Waltz (Waltz, 1972). Cette technique nécessite de décomposer les contraintes du problème en contraintes primitives (binaires).

Ainsi, chaque équation est décomposée en une série d'équations binaires pour la phase de propagation et une autre série d'équations binaires pour la phase de rétro-propagation. Il en résulte la saisie manuelle et l'écriture de plusieurs équations. Pour illustrer, prenons un exemple simple. Considérons les trois contraintes ci-dessous. On notera que ces contraintes ne sont pas prises au hasard. Elles ont un lien avec la localisation de véhicule. Nous en reparlerons dans les prochaines sections.

$$\begin{cases} [x_k] &= [x_{k-1}] + [\delta s_k]\, cos([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ [y_k] &= [y_{k-1}] + [\delta s_k]\, sin([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ [\theta_k] &= [\theta_{k-1}] + [\delta\theta_k] \end{cases} \tag{5.3}$$

51. Constraint propagation

Pour utiliser la méthode de Waltz, il faut nécessairement décomposer les contraintes en contraintes binaires (encore appelées contraintes primitives). Il faut créer :

– un groupe de contraintes pour la phase de Forward
– un groupe de contraintes pour la phase de Backward.

Pour la phase de Forward, on obtient les contraintes binaires suivantes :

$$\begin{cases} [a_1] & = [\delta\theta_k]\,/2 \\ [a_2] & = [\theta_{k-1}] + [a_1] \\ [a_3] & = cos\,([a_2]) \\ [a_4] & = [\delta s_k] \times [a_3] \\ [x_k] & = [x_{k-1}] + [a_4] \\ [a_5] & = sin\,([a_2]) \\ [a_6] & = [\delta s_k] \times [a_5] \\ [y_k] & = [y_{k-1}] + [a_6] \\ [\theta_k] & = [\theta_{k-1}] + [\delta\theta_k] \end{cases} \tag{5.4}$$

Pour la phase de Backward, on réécrit les contraintes binaires de manière à calculer les résultats des membres à droite de la formule (eq. 5.3). On obtient les contraintes binaires suivantes :

$$\begin{cases} [x_{k-1}] & = [x_k] - [a_4] \\ [a_7] & = [x_k] - [x_{k-1}] \\ [\delta s_k] & = [a_7]\,/\,[a_3] \\ [a_8] & = [a_7]\,/\,[\delta s_k] \\ [a_9] & = arccos\,([a_8]) \\ [\theta_{k-1}] & = [a_9] - [a_1] \\ [a_{10}] & = [a_9] - [\theta_{k-1}] \\ [\delta\theta_k] & = [a_{10}] \times 2 \\ [y_{k-1}] & = [y_k] - [a_5] \\ [a_{11}] & = [y_k] - [y_{k-1}] \\ [\delta s_k] & = [a_{11}]\,/\,[a_5] \\ [a_{12}] & = [a_{11}]\,/\,[\delta s_k] \\ [a_{13}] & = arcsin([a_{12}] \\ [\theta_{k-1}] & = [a_{13}] - [a_1] \\ [a_{14}] & = [a_{13}] - [\theta_{k-1}] \\ [\delta\theta_k] & = [a_{14}] \times 2 \end{cases} \tag{5.5}$$

Finalement, on a 9 équations pour le forward et 16 équations pour le backward. On rencontre souvent des systèmes d'équations qui sont beaucoup plus dense en termes de nombre de contraintes et de nombre d'opérations. Pour se faire une idée, nous traiterons dans ce document des systèmes comportant 40 fois le

nombre des contraintes de la même forme que dans l'équation 5.3. Cette tâche de décomposition implique des difficultés fréquentes :

- d'une part, on ne peut garantir le risque d'erreur lors de la saisie manuelle (programme difficilement lisible)
- d'autre part, lorsque les contraintes du problème sont nombreuses, la décomposition devient une tâche fastidieuse. Et le débogage du programme s'en trouve compliqué.

5.6.2 Approches possibles

Nous avons donc cherché à éviter l'étape de décomposition. Pour y arriver, nous avons deux possibilités :

5.6.2.1 Approche par binarisation

La première possibilité consiste à développer un algorithme qui génère un CSP binaire à partir d'un CSP n-aire comme dans (Dechter and Pearl, 1989; Rossi et al., 1990). En d'autres termes, réaliser un outil qui décompose automatiquement les CSP de façon à n'avoir que des contraintes binaires et unaires. Cette solution ajoute une étape supplémentaire dite de "binarisation" au processus de localisation.

5.6.2.2 Approche sans binarisation

Notons que le problème de décomposition s'est déjà posé aux mathématiciens (surtout dans le domaine de l'optimisation globale). Il est fréquent de traiter des CSP contenant plusieurs dizaines de contraintes. Et la décomposition en contraintes primitives conduit à des nombres de contraintes assez élevés. Des approches de solution ont été proposées. Ainsi, dans l'état de l'art sur la propagation de contraintes sur les intervalles, et essentiellement dans le domaine de l'optimisation globale, nous trouvons les algorithmes comme HC4 (Benhamou et al., 1999), BC3 (Benhamou et al., 1994), BC4 (Benhamou et al., 1999), 3B (Lhomme, 1993), présentés dans la section 4.5. Ces algorithmes prennent en entrée, des ICSP et fournissent en sortie, des domaines contractés associés à chacune des variables contenues dans les contraintes.

5.6.2.3 Choix de l'approche

Nous avons opté pour cette deuxième solution afin de comparer différents algorithmes sur le problème de la localisation de véhicule. Notons que d'autres travaux comme (Granvilliers, 2000; Benhamou et al., 1999; Chabert and Jaulin, 2009) ont effectué des comparaisons sur ces algorithmes d'ICP en les appliquant à des problèmes pris dans des domaines tels que la chimie, l'astrophysique, etc. Pour chaque problème, un CSP est formalisé. Puis, ce CSP est résolu à l'aide d'un solveur à intervalles. Le temps de calcul ainsi que la précision sont ensuite analysés. Aucune étude de ce genre n'a été faite sur la problématique de la localisation.

Nous avons effectué cette étude appliquée au problème de la localisation de véhicule où le CSP évolue avec une fenêtre temporelle glissante. Ces algorithmes sont disponibles sous certains solveurs à intervalles. Notons aussi qu'en général, ces solveurs ne sont pas utilisés pour résoudre ce genre de problème.

Notre objectif est donc double. Il s'agit dans un premier temps de tester la possibilité d'utiliser un solveur à intervalles dans le cadre de la localisation en contexte dynamique. Dans un deuxième temps, il s'agit de s'assurer que ces algorithmes résolvent les ICSP typiquement liés au domaine véhiculaire et ensuite comparer leurs performances en vue de déduire le plus adéquat. Nous allons présenter dans cette section la comparaison effectuée sur ces différents algorithmes ICP en les appliquant à un problème de localisation dans un environnement d'extérieur.

5.6.3 Processus de localisation

La figure 5.2 présente le processus de localisation utilisé. Les données proprioceptives sont utilisées pour réaliser la phase de prédiction. A l'issue de cette phase, des boîtes $[X_k^*][Y_k^*][\theta_k^*]$ constituent la pose prédite du véhicule. Lorsqu'une donnée GPS est disponible, on procède à la création (le cas initial) ou à la mise à jour d'un DICSP (Dynamic Interval Constraint Satisfaction Problem). La solution de ce DICSP à l'instant k constitue la pose corrigée $[X_k][Y_k][\theta_k]$ du véhicule. Le cycle qui se répète tant que le véhicule se déplace.

On remarquera que dans ce processus, il n'y a pas de carte de l'environnement contrairement au schéma générique présenté par la figure 5.1.L'utilisation d'une carte nécessite, en général, une connaissance a priori de l'environnement. Ceci peut diminuer l'autonomie du véhicule.

5.6.3.1 Le modèle de déplacement dans un contexte à erreurs bornées

La configuration imprécise initiale du véhicule est représentée par une boîte (un vecteur d'intervalles) :

$$[\mathcal{X}] = \begin{pmatrix} [x] \\ [y] \\ [\theta] \end{pmatrix}$$

(x, y) sont les coordonnées du centre de l'essieu arrière et θ est l'orientation du repère local attaché au véhicule. La notion de "vecteur d'intervalles" est présentée dans la section 3.3 de ce document. Un déplacement du véhicule les instants $k-1$ et k est représentée par :

$$[\mathcal{X}_k] = \begin{cases} [x_{k-1}] + [\delta s_k] \cos([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ [y_{k-1}] + [\delta s_k] \sin([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ [\theta_{k-1}] + [\delta\theta_k] \end{cases} \tag{5.6}$$

où [ ] représentent les intervalles encadrant les valeurs et :

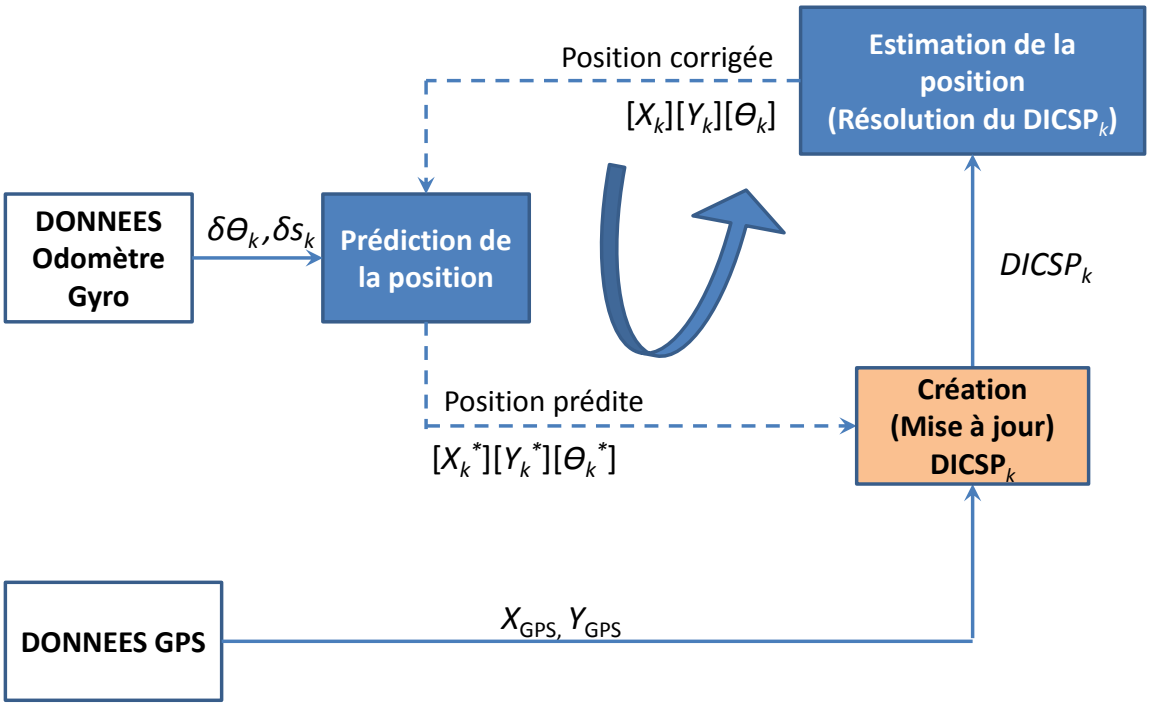


FIGURE 5.2 – Principe de la localisation ensembliste

– $[\delta s_k]$ est fourni par les odomètres. Les odomètres sont placés sur les deux roues arrière du véhicule. Ils fournissent la distance parcourue par chaque roue prise indépendamment. $[\delta s_k]$ est calculé de la manière suivante :

$$[\delta s_k] = \frac{[\pi]\left([w_l]\,[\delta p_l] + [w_r]\,[\delta p_r]\right)}{P} \tag{5.7}$$

w_r est le rayon de la roue droite, w_l est le rayon de la roue gauche et P est la résolution de l'odomètre. $[\pi]$ est l'intervalle représentant le nombre π.

– $[\delta\theta_k]$ est obtenu à partir du gyromètre. Un gyromètre est un capteur d'orientation qui mesure la vitesse d'orientation dans un système de référence inertiel. Notre gyromètre est basé sur la vibration d'une structure de silicone qui utilise la force de Coriolis pour calculer la vitesse angulaire indépendamment de l'accélération. Il mesure la vitesse de lacet à partir de laquelle nous déduisons la rotation élémentaire $[\delta\theta_k]$:

$$[\delta\theta_k] = [\delta\theta_{gyro} - \varepsilon_{gyro}, \delta\theta_{gyro} + \varepsilon_{gyro}] \tag{5.8}$$

– $[\mathcal{X}_{k-1}] = \begin{pmatrix} [x_{k-1}] \\ [y_{k-1}] \\ [\theta_{k-1}] \end{pmatrix}$ la boîte de localisation à l'étape précédente

– $[x_k]$ et $[y_k]$ sont initialisés avec les mesures GPS (voir Eq. 5.9). Notre récepteur GPS fournit les coordonnées de longitude et de latitude qui sont ensuite converties en une position $[\mathcal{Y}] = \begin{pmatrix} [x] \\ [y] \end{pmatrix}$ dans un référentiel cartésien local. Il calcule les imprécisions de mesure ε_{gps} en ligne et les renvoie dans le référentiel GST NMEA.

$$[\mathcal{Y}] = \begin{pmatrix} [x_{gps} - \varepsilon_{x,gps}, x_{gps} + \varepsilon_{x,gps}] \\ [y_{gps} - \varepsilon_{y,gps}, y_{gps} + \varepsilon_{y,gps}] \end{pmatrix} \tag{5.9}$$

- $[\theta_k]$ est initialisé à $]-\pi, \pi]$.
- $[x_{k-1}]$, $[y_{k-1}]$ et $[\theta_{k-1}]$ sont obtenus à partir de la résolution de l'ICSP précédent (correspondant à l'instant $k-1$).

5.6.3.2 Présentation du problème de localisation sous forme de DICSP

Les contraintes au temps k sont données par le système d'équations suivant :

$$\left\{ \begin{array}{lcc} [x_k] & = & [x_{k-1}] + [\delta s_k] \cos([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ [y_k] & = & [y_{k-1}] + [\delta s_k] \sin([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ [\theta_k] & = & [\theta_{k-1}] + [\delta\theta_k] \end{array} \right\} \tag{5.10}$$

$[\mathcal{X}_k] = \begin{pmatrix} [x_k] \\ [y_k] \\ [\theta_k] \end{pmatrix}$ la boîte de localisation courante.

Un nouveau ICSP$_k$ (V_k, D_k, C_k) est généré à chaque temps k. L'ICSP$_k$ est défini de la manière suivante :

- un ensemble de variables
$$V_k = \{x_k, y_k, \theta_k, x_{k-1}, y_{k-1}, \theta_{k-1}, \delta s_k, \delta\theta_k\}$$
- un ensemble de domaines
$$D_k = \{[x_k], [y_k], [\theta_k], [x_{k-1}], [y_{k-1}], [\theta_{k-1}], [\delta s_k], [\delta\theta_k]\}$$
- un ensemble de contraintes
$$C_k = (C_k^1, C_k^2, C_k^3)^T$$

$$\left\{ \begin{array}{lcc} C_k^1 \Leftrightarrow [x_k] & = & [x_{k-1}] + [\delta s_k] \cos([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ C_k^2 \Leftrightarrow [y_k] & = & [y_{k-1}] + [\delta s_k] \sin([\theta_{k-1}] + \frac{[\delta\theta_k]}{2}) \\ C_k^3 \Leftrightarrow [\theta_k] & = & [\theta_{k-1}] + [\delta\theta_k] \end{array} \right. \tag{5.11}$$

C_k correspond au modèle de déplacement du véhicule. On peut remarquer que l'ICSP traité dans le cadre de la localisation est dynamique (Dechter and Dechter, 1988). En effet à chaque instant k, nous ajoutons de nouvelles contraintes au système et si $k > \Xi$ (où Ξ représente la taille de fenêtre), nous supprimons les plus anciennes contraintes d'un système. L'opération d'addition (resp. de suppression) est aussi nommée "restriction" (resp. "relaxation"). Le DICSP (Dynamic ICSP) de localisation garde la même taille mais change temporairement. Notre DICSP est défini par $3 \times \Xi$ contraintes. Par conséquent, il est considéré à partir de l'instant $k - \Xi + 1$ jusqu'à l'instant k (l'instant présent). Le DICSP de localisation est défini à l'instant k par :

$$DICSP_k = \begin{pmatrix} ICSP_k \\ ICSP_{k-1} \\ ICSP_{k-2} \\ ... \\ ICSP_{k-\Xi+1} \end{pmatrix} \tag{5.12}$$

Les algorithmes ICP (Interval Constraint Propagation) détaillés dans la section 4.5 sont utilisés pour résoudre chaque $DICSP_k$ à l'instant k. Les résultats constituent la pose courante du véhicule.

5.6.4 Résultats

Pour valider les algorithmes de localisation, les données ont été collectées sur la piste de Satory (voir fig. 5.3 et 5.4) avec le véhicule prototype CARLLA [52] du LIVIC [53]. CARLLA est présenté dans la section 2.3 de ce document. Pendant la durée de l'expérimentation, le véhicule roulait à une vitesse moyenne de 50 km/h (pendant 7 min environ). Un gyromètre fournit les informations de vitesse de lacet et la localisation globale est réalisée par un récepteur GPS (AgGPS132). Les mesures du GPS ainsi que l'acquisition des données sur le gyromètre (VG400CC) et l'odomètre sont réalisées à une fréquence de 5Hz. Les données de positionnement fournies par un GPS-RTK (à précision centimétrique) sont utilisées comme vérité terrain. La piste de test de Satory a deux parties. La première partie (la partie en haut dand fig. 5.3 et fig. 5.4) est sous la forme d'une route péri-urbaine et la seconde partie peut être perçue comme une route rurale. Cette deuxième partie (en bas : peu après l'instant $t = 100$ jusqu'à $t = 250$ sur la figure 5.4) est très intéressante parce qu'elle est en partie recouverte par une forêt. On observe d'importantes pertes du signal GPS (avec AgGPS132) dans cette région. Ce problème a donc lieu sur une longue durée du trajet pendant l'expérimentation. Cependant, cette situation est utile puisqu'elle nous permet de tester la robustesse de nos algorithmes.

52. Contrôleur d'Assistance Routière Longitudinale et LAtérale
53. Laboratoire sur les Interactions Véhicules-Infrastructure-Conducteurs

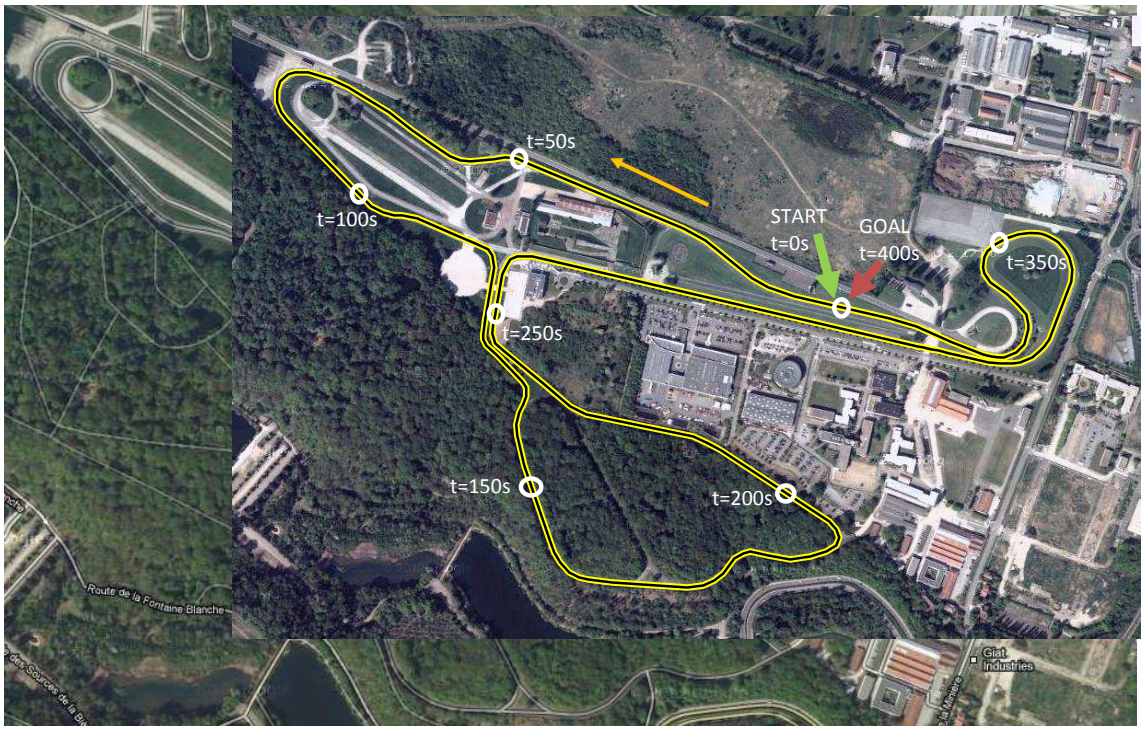


FIGURE 5.3 – Piste de Satory

FIGURE 5.4 – Piste de Satory avec la trajectoire du véhicule

Nous avons développé un outil logiciel basé sur le solveur Realpaver (Granvilliers and Benhamou, 2006). Les tests ont été effectués sur un PC équipé d'un processeur Intel Core i7 CPU 960 @ 3.20GHz tournant sur un système Ubuntu 12.04 LTS.

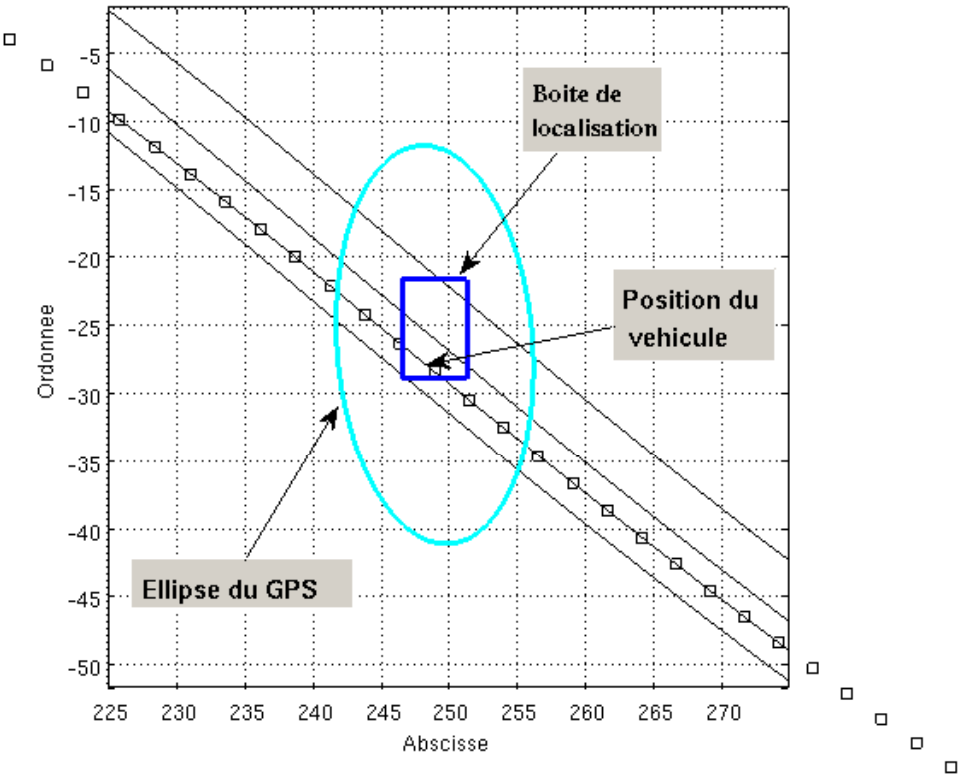


FIGURE 5.5 – Exemple de boîte résultat de localisation ICP

HC4, BC3, BC4 et 3B-BC4[54] ont été utilisés pour résoudre des DICSP de la forme (5.10). Ces algorithmes ont été retenus pour les expérimentions parce qu'ils nous semblent représentatifs de tous les autres méthodes ICP disponibles sous le solveur RealPaver et dans l'état de l'art. D'autres algorithmes comme BC5 et Weak3B ont aussi été étudiés. Mais les résultats de BC5 et ceux de BC4 d'une part sont quasiment identiques de même que les résultats de Weak3B et ceux de 3B, d'autre part. La résolution des DICSP a été effectuée pour chaque algorithme sur les mêmes données. La figure 5.5 montre que la boîte de localisation obtenue à chaque instant (tout au long du parcours du véhicule) contient bien la position réelle du véhicule indiquée par le GPS-RTK.

5.6.4.1 Influence de la fenêtre

Pour l'algorithme HC4, nous avons fait varier la valeur de w entre 1 et 200 (voir la partie gauche de la figure 5.6) afin de déterminer la meilleure taille de fenêtre w. Nous avons calculé la surface moyenne de la boite de localisation pour un tour de piste. Le graphe à gauche dans la figure 5.6 correspond à une fonction hyperbolique. Plus la fenêtre de localisation est grande et plus petite est l'erreur de localisation. La valeur de la surface de localisation baisse de 15% (de 760 à $650m^2$) pour $w = 20$. En faisant augmenter la valeur de la fenêtre w, on améliore la taille de la surface d'environ 4%.

54. Rappelons que 3B-BC4 correspond à l'utilisation de 3B avec BC4 comme un algorithme de faible consistance.

FIGURE 5.6 – Surface moyenne de localisation pour l'algorithme HC4 et le temps de calcul pour les différentes tailles de fenêtre

Algorithmes	HC4	BC3	BC4	3B-BC4
Durée (ms)	0.542	1.02	0.58	5257
Surface moyenne de localisation (m^2)	658.3	658.3	656.9	633.5
Imprécision moyenne sur θ (°)	33.9	33.9	33.9	23.1

Tableau 5.2 – Comparaison des algorithmes

5.6.4.2 Temps de calcul

Pour étudier l'influence de la fenêtre de localisation sur le temps de calcul, nous avons mesuré le temps moyen mis par chaque algorithme pour résoudre un DICSP et ce pendant un tour complet de la piste (environ 2000 CSPs ont été considérés le long de chaque tour). La partie droite de la figure 5.6 montre (pour HC4) que le temps de calcul s'accroît linéairement avec w. Les résultats obtenus pour BC3 et BC4 sont assez proches.

Le tableau 5.2 montre le temps moyen mis par l'algorithme pour résoudre un DICSP à chaque pas de temps. Chaque DICSP contient $3 \times w$ équations (nous avons choisi $w = 20$). HC4, comme déjà mentionné, est le plus rapide des algorithmes (chaque CSP est résolu en $0.52ms$). BC3 est plus consommateur en temps de calcul que HC4.

BC4 contourne la lenteur de BC3 en utilisant HC4 lorsqu'aucune variable n'apparaît plus d'une fois dans une contrainte. Dans l'équation 5.10, on peut noter que chaque variable apparaît seulement une fois sur chaque ligne du CSP. Par conséquent, BC4 utilise seulement HC4 et les temps de calcul sont assez proches. La petite différence entre les résultats obtenus pour ces deux algorithmes est due au test effectué par BC4 pour choisir à chaque fois quel algorithme à utiliser (HC4 ou BC3) en fonction des contraintes à traiter.

3B-BC4 [55] est le meilleur pour la réduction des domaines des variables. Mal-

55. Il s'agit de l'algorithme 3B utilisant BC4 comme algorithme de bas niveau

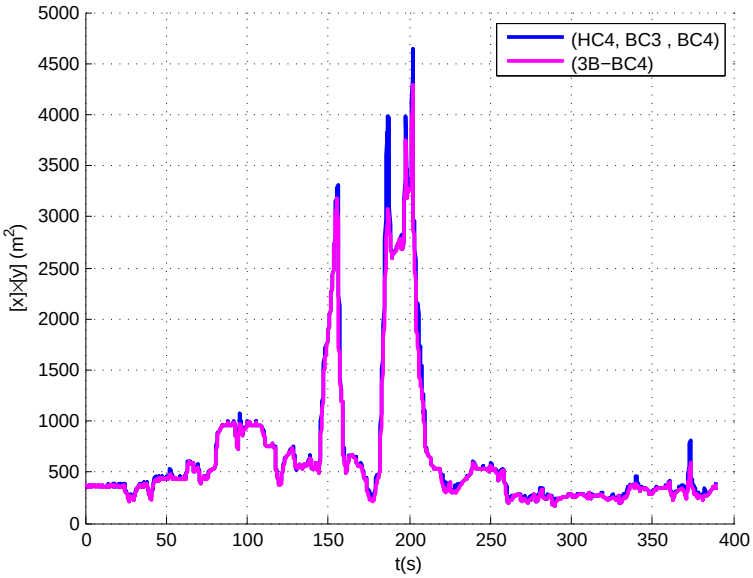


FIGURE 5.7 – Surface de localisation ICP par étape

heureusement, il souffre d'un temps de calcul prohibitif et ne peut donc fonctionner en temps réel (il met en moyenne 5.2s pour résoudre le CSP à chaque étape de l'expérimentation). On peut déduire de la surface moyenne de localisation (voir tableau 5.2) que 3B fournit un meilleur résultat en moyenne que les autres algorithmes : 4% de moins que les algorithmes de consistance locale (HC4, BC3 et BC4). La surface des boîtes de localisation par étape (figure 5.7) est calculée à l'instant k avec la formule suivante :

$$S_localisation_k = (\overline{x_k} - \underline{x_k}) \times (\overline{y_k} - \underline{y_k}) \tag{5.13}$$

avec $[x_k] \times [y_k]$ étant la boîte de localisation courante englobante. La surface des boîtes de localisation nous donne une vue d'ensemble sur les différentes incertitudes en x et y.

5.6.4.3 Imprécision en orientation

Le calcul de l'imprécision en orientation (fig. 5.8) est fait de manière suivante à l'instant k :

$$Inc_\theta_k = w([\theta_k]) = \overline{\theta_k} - \underline{\theta_k}$$

On peut remarquer que l'imprécision moyenne en orientation (voir tableau 5.2) est plus faible avec 3B (37%) qu'avec les autres algorithmes. La figure 5.8 aussi confirme ces résultats et montre que les algorithmes de consistance locale ne réduisent pas l'imprécision en orientation qui s'accumule avec le temps.

Contrairement aux autres algorithmes, 3B réduit l'imprécision en orientation. Par conséquent, il réduit mieux les erreurs en $[x]$ et $[y]$ que HC4, BC4 et BC3.

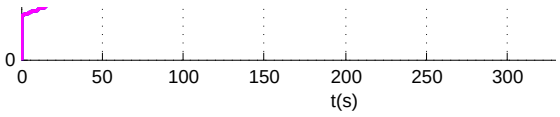


FIGURE 5.8 – Incertitude d'orientation

5.6.4.4 Consistance des résultats de localisation

L'intervalle de l'erreur d'un état a est défini par $[\underline{a} - a_{ref}, \overline{a} - a_{ref}]$ où a_{ref} est l'état de référence. Par conséquent, un algorithme de localisation fournit de bons résultats si son couloir est petit et s'il inclut toujours la valeur zéro (cela veut dire que le résultat de l'algorithme encadre la référence). Etant donné que le GPS-RTK nous fournit la vraie position (x_{vrai}, y_{vrai}) du véhicule, nous pouvons calculer les couloirs d'incertitude en x et y de manière suivante :

$$\begin{aligned} [clr_x] &= [x] - x_{vrai} \\ [clr_y] &= [y] - y_{vrai} \end{aligned} \tag{5.14}$$

On peut vérifier la consistance de l'algorithme à l'aide de ce couloir. Si la valeur zéro est incluse dans le couloir $[clr_x]$, alors le résultat est consistant en x. Cela veut dire que $[x]$ contient la vraie position du véhicule. Nous avons donc comparé les couloirs formés par chaque algorithme. Le GPS fournit des mesures consistantes (mesures qui encadrent la référence) avec d'importantes variations (voir fig. 5.9). HC4, BC3 et BC4 donnent des résultats de localisation consistants et similaires entre eux. On peut remarquer que, de manière générale, ces algorithmes fournissent des résultats consistants. Cela n'est pas toujours le cas des filtres bayésiens qui peuvent parfois perdre leur consistance (Lambert et al., 2009).

5.6.5 Conclusion

Dans cette section, nous avons utilisé les algorithmes de propagation de contraintes basés sur des intervalles pour résoudre des CSP (avec une fenêtre temporelle) en

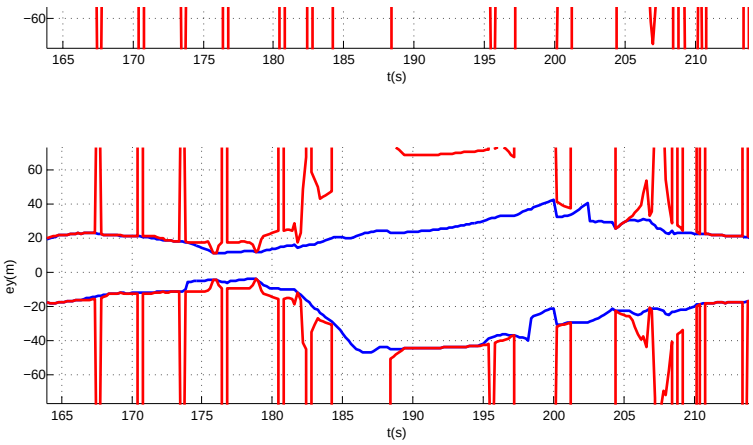


FIGURE 5.9 – Intervalle d'erreur (de la 163s à 212s)

vue de traiter un problème de localisation. Quatre algorithmes de propagation de contraintes (HC4, BC3, BC4 et 3B-BC4 [56]) ont été testés et comparés sur les mêmes données expérimentales. Nous avons montré que les algorithmes de consistance locale ne corrigent pas l'imprécision en orientation du véhicule pendant le déplacement.

Contrairement aux travaux précédents, nous avons démontré qu'il n'est pas nécessaire de décomposer les contraintes du problème en contraintes élémentaires afin de localiser un véhicule équipé d'odomètres, de gyromètre et d'un récepteur GPS. A chaque pas de temps, le déplacement imprécis du véhicule est mesuré et la position du véhicule est corrigée avec les mesures GPS. Les algorithmes testés fournissent des résultats similaires en termes de localisation excepté 3B-BC4 qui offre de meilleures réductions. De plus, nous avons vu que seul 3B réduit l'imprécision en orientation. BC3 est deux fois plus lent que HC4 et BC4. Il convient de noter que 3B est trop lent et n'est donc pas adéquat pour des applications en temps réel.

5.7 Proposition de correction du cap du véhicule

Comme nous l'avons vu dans la section précédente, les algorithmes de consistance locale ne permettent pas de contracter les intervalles représentant le cap du véhicule. Nous présenterons dans cette section, dans un premier temps, l'explication du problème, ensuite les solutions possibles, et la formulation de la solution proposée.

56. Il s'agit de l'algorithme 3B utilisant BC4 comme algorithme de bas niveau

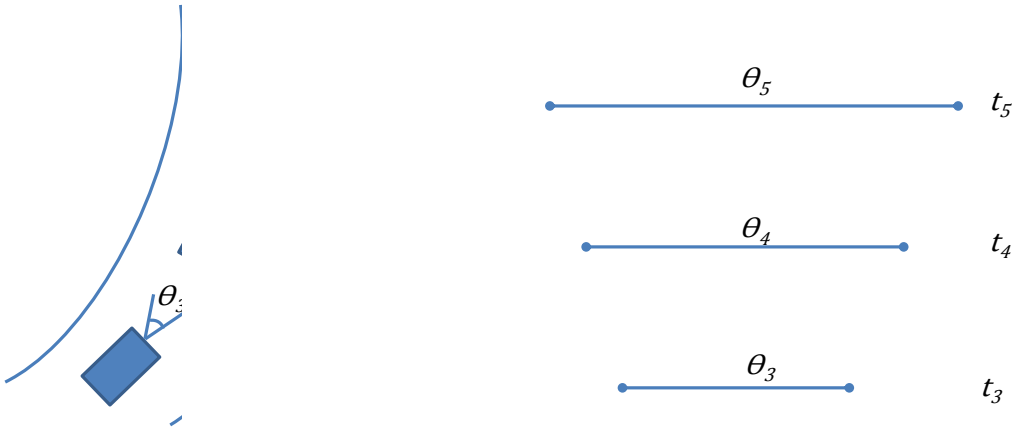


FIGURE 5.10 – Heading uncertainty

5.7.1 Présentation du problème

A mesure que le véhicule se déplace, l'incertitude en orientation s'accumule malgré la réduction des autres paramètres de la pose. La figure 5.10 est une illustration à titre indicatif du phénomène. $w()$ la taille de l'intervalle[57] fourni en argument, on obtient :

$$w([\theta_3]) \leq w([\theta_4]) \leq w([\theta_5]) \leq ... \leq w([\theta_k]) \tag{5.15}$$

Les $[\theta_i]$ avec $i \in \{3, .., k\}$ représentent les intervalles correspondant à la valeur de l'angle de la pose du véhicule aux instants i. On remarque que $w()$ est une fonction croissante du temps. Ceci est problématique puisque la connaissance du cap du véhicule est un élément important pour une bonne estimation de la position en x et y. On peut montrer que les algorithmes classiques de propagation rétro-propagation ICP ne permettent pas de réduire les incertitudes sur le cap pour un véhicule équipé d'un récepteur GPS, d'un gyromètre et d'odomètres.

(Vincke and Lambert, 2011) ont expliqué le problème. Ils ont utilisé un algorithme qui nécessite la décomposition de toutes les contraintes du problème en contraintes binaires. Les algorithmes de consistance locale utilisent une seule boîte pour représenter $[\theta_{k-1}]$. L'équation 5.16 est celle qui modifie la valeur des variables $[\theta_{k-1}]$.

$$\begin{array}{rcl} [\theta_{k-1}] & = & [\theta_{k-1}] \bigcap (\arccos(\frac{[x_k]-[x_{k-1}]}{[\delta s_k]}) - \frac{[\delta\theta_k]}{2}) \\ [\theta_{k-1}] & = & [\theta_{k-1}] \bigcap (\arcsin(\frac{[y_k]-[y_{k-1}]}{[\delta s_k]}) - \frac{[\delta\theta_k]}{2}) \end{array} \tag{5.16}$$

Considérons un cas avec un déplacement le long de l'axe y. Il s'ensuit que la variation de l'orientation est nulle ($\frac{[\delta\theta_k]}{2} = 0$) et x reste constant. La correction majeure sur θ sera réalisée par :

$$\arcsin(([y_k] - [y_{k-1}])/[\delta s_k]) \tag{5.17}$$

57. Dans cette section du document, w sera utilisé pour désigner la taille de l'intervalle pris en paramètre, et Ξ pour désigner la taille de la fenêtre glissante de localisation

La largeur de $[y_{k-1}]$ avoisine en général, $10m$ dans les expériences. En considérant en plus, par souci de simplicité, un déplacement centré sur l'axe des abscisses de sorte qu'on ait par exemple $[y_{k-1}] = [-5; 5]$.

Alors on peut avoir :

$$[y_k] - [y_{k-1}] = [-5 + \epsilon_y; 5 - \epsilon_y] - [-5; 5] = [-10 + \epsilon_y; 10 - \epsilon_y] \tag{5.18}$$

où ϵ_y est une petite valeur de l'imprécision due à une nouvelle mesure du GPS. $[\delta s_k]$ a une valeur autour de $1m$: $[\delta s_k] = [1 - \epsilon_s, 1 + \epsilon_s]$. Finalement, on obtient,

$$\begin{array}{ll} \arcsin(\frac{[y_k]-[y_{k-1}]}{[\delta s_k]}) & = \arcsin([-10 + \epsilon_y; 10 - \epsilon_y]/[1 - \epsilon_s, 1 + \epsilon_s]) \\ & =]-\pi; \pi] \end{array} \tag{5.19}$$

La valeur de l'intervalle en paramètre de la fonction *arcsin* est au delà[58] de l'intervalle $[-1, 1]$; de fait, nous n'avons aucune connaissance de la valeur de l'orientation. Les équations 5.16 ne permettent donc pas de corriger l'orientation du véhicule.

Dans le cadre d'algorithmes de propagation de contraintes, seul l'algorithme de consistance 3B (Lhomme, 1993) (basé sur une technique de consistance forte) permet de réduire l'incertitude du cap du véhicule pendant son déplacement. Cependant, cet algorithme est très gourmand en ressources systèmes et en temps de calcul. Plusieurs possibilités peuvent être envisagées pour accélérer la méthode 3B.

Possibilité 1 : Réécriture des équations La théorie intervalle montre aussi que l'expression analytique d'une fonction ou d'une équation peut influer sur la précision du résultat du calcul. Dans les articles (Tóth and Csendes, 2005), (Tóth et al., 2007) Toth a conduit des études expérimentales en utilisant plusieurs formes des fonctions d'inclusion pour améliorer la précision des boîtes résultats. Nous allons présenter ici la forme centrée de Moore (Moore, 1966) et la forme de la valeur moyenne.

- La forme centrée de Moore : Pour obtenir une fonction d'inclusion en forme centrée de Moore $F_c(X_1, ..., X_n)$, nous réécrivons la fonction $f(x_1, ..., x_n)$ de manière suivante[59] :

$$f(x_1, ..., x_n) = f(c_1, ..., c_n) + g(y_1, ..., y_n) \tag{5.20}$$

où $y_i = x_i - c_i$ et g est définie par :

$$g(y_1, ..., y_n) = f(y_1 + c_1, ..., y_n + c_n) - f(c_1, ..., c_n) \tag{5.21}$$

58. $[-1, 1] \subset [-10 + \epsilon_y; 10 - \epsilon_y]/[1 - \epsilon_s, 1 + \epsilon_s]$

59. Dans un souci de lisibilité, nous représentons, ici, les variables intervalles et les fonctions d'inclusion par des lettres majuscules d'une part et les variables réelles et les fonctions de réels par des lettres minuscules d'autre part.

où $c_i = m(X_i)$. Si f est une fonction rationnelle, nous pouvons exprimer g sous la forme :

$$g(y_1, ..., y_n) = y_1 h_1(y_1, ..., y_n) + \cdots + y_n h_n(y_1, ..., y_n) \tag{5.22}$$

Nous définissons F_c, la fonction d'inclusion en forme centrée de Moore, ainsi :

$$F_c(X_1, ..., X_n) = f(c_1, ..., c_n) + \sum_{i=1}^{n} Y_i H_i(Y_1, ..., Y_n)$$

$Y_i = X_i - c_i$ et H_i est la fonction d'inclusion naturelle de h_i
Prenons un exemple (Moore et al., 2009) :
Considérons $p(x) = 1-5x + \frac{1}{3}x^3$. Nous réécrivons $p(x)$ sous la forme :

$$p(x) = 1-5c + \frac{1}{3}c^3 + g(y)$$

où $y = x-c$ et

$$g(y) = p(x) - p(c) = y(-5 + 13((y + c)^2 + (y + c)c + c^2)) \tag{5.23}$$

On peut définir la fonction d'inclusion centrée P_c comme suit :

$$P_c(X) = p(c) + Y H(Y) \tag{5.24}$$

où $c = m(X)$, $Y = X - m(X)$, et

$$H(Y) = -5 + \frac{1}{3}((Y + c)^2 + (Y + c)c + c^2) \tag{5.25}$$

Un autre exemple :
La fonction d'inclusion pour la fonction $f(x) = \frac{1}{3}x^3 - 2x$ s'écrit :

$$F_c(X) = f(c) + Y H(Y) \tag{5.26}$$

où $c = m(X)$, $Y = X - m(X)$, et $H(Y) = -2 + \frac{1}{3}(Y + c)^2 + (Y + c)c + c^2)$
Pour l'intervalle $[1.2, 1.6]$, nous obtenons $F_c([1.2, 1.6]) \subseteq [-1.95, -1.83]$
Un autre exemple :
Pour le polynome $p(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$, la fonction d'inclusion (forme centrée) s'écrit :
$P_c(x) = Y^4 - 2Y^3 - Y^2 + 2Y + p(c)$ où $c = m(X)$, $Y = X - m(X)$

- La forme de la valeur moyenne : Soit X un vecteur d'intervalles et $m = m(X)$. Soit $F^{(i)}$ la fonction d'inclusion de la dérivé i^{eme} de f.

$$F_{mv}(X) = f(m) + \sum_{i=1}^{n} F^{(i)}(X)(X_i - m_i)$$

$F_{mv}(X)$ est la fonction d'inclusion sous la forme de la valeur moyenne de f sur X.
Exemple :
Considérons la fonction $f(x) = \frac{1}{3}x^3 - 2x$

$f'(x) = x^2 - 2$, donc nous avons $F^{(1)}(X) = X^2{-}2$.

$$F_{mv}(X) = f(m) + (X^2{-}2)(X - m)$$

Pour l'intervalle $[1.2, 1.6]$, on obtient $F^{(1)}([1.2, 1.6]) = [-0.56, 0.56]$ et $m = \frac{1.6+1.2}{2} = 1.4$

$F_{mv}([1.2, 1.6]) = f(1.4) + [-0.56, 0.56] \times [-0.2, 0.2]$
$F_{mv}([1.2, 1.6]) = -1.885333 + [-0.112, 0.112]$
$F_{mv}([1.2, 1.6]) = [-2.01, -1.77]$

Le lecteur intéressé par ce sujet peut se référer à (Moore et al., 2009) pour avoir plus de détails sur ces deux types de fonctions d'inclusion. Il existe d'autres méthodes d'écriture des fonctions d'inclusion notamment sous la forme optimale centrée de Baumann (Baumann, 1988), la forme inclinée (Ratz, 1998), la forme affine (Comba and Stol, 1993; Figueiredo and Stolfi, 1996; Stol and De Figueiredo, 1997). Cette option est intéressante puisque les techniques de consistance utilisent les fonctions d'inclusion naturelles déduites des contraintes de l'ICSP. Une contrainte réécrite sous une autre forme peut conduire à des résultats meilleurs ou pires que ceux obtenus avec la forme originelle.

Dans tous les cas, il s'agit de réécrire les contraintes sous des formes particulières avant de passer à l'étape de résolution des ICSP. Cette approche ne nous convient pas puisque le problème de réécriture est toujours présente et peut induire des erreurs. De plus, plusieurs études comme (Alander, 1985) et (Tóth et al., 2007) montrent qu'il n'existe pas de méthode qui soit toujours la meilleure. L'efficacité de la méthode choisie dépend des propriétés de la fonction et des intervalles des arguments.

Possibilité 2 : Bissecter une variable Cette option consiste à découper une variable du problème afin d'obtenir des sous-problèmes que l'on traite ensuite séparément.

Nous avons retenu cette option étant donné que le problème de dérive en terme d'orientation du véhicule est dû au fait que les contraintes sont traitées séparément ; ce qui est tout à fait normal pour les algorithmes de consistance locale. Nous proposons donc dans cette section, une autre méthode pour la localisation de véhicule pouvant réduire l'incertitude du cap et fonctionner en temps réel. L'algorithme proposé utilise HC4 comme algorithme de bas niveau. Les résultats de cet algorithme sont comparés à ceux des algorithmes comme HC4 et 3B.

5.7.2 Formulation du problème de localisation sous forme de νDICSP

En vue de corriger l'incertitude en orientation et maintenir un temps de calcul raisonnable, pendant le processus de localisation, nous proposons de découper la variable $\theta_{k-\Xi}$ en ν variables $\theta_{(k-\Xi),i}$, $i \in \{1, 2, ..., \nu\}$ où ν est le "facteur de découpage". Chaque domaine $[\theta_{(k-\Xi),i}]$ est alors défini par :

$$[\theta_{(k-\Xi),i}] = [\underline{\theta_{k-\Xi}} + (i-1)\mathbf{S}, \overline{\theta_{k-\Xi}} + i\mathbf{S}] \tag{5.27}$$

où $k \geq \Xi$, $\mathbf{S} = \frac{w([\theta_{k-\Xi}])}{\nu}$ est la largeur de l'intervalle de chaque $[\theta_{(k-\Xi),i}]$, et $w([\theta_{k-\Xi}])$ représente la largeur de l'intervalle $[\theta_{k-\Xi}]$. Lorsque k est inférieur à Ξ, $[\theta_0]$ est découpé. $[\theta_{k-\Xi}]$ constitue l'union de tous les $[\theta_{(k-\Xi),i}]$:

$$[\theta_{k-\Xi}] = \bigcup_{i=1}^{\nu} [\theta_{(k-\Xi),i}] \tag{5.28}$$

Pour illustrer, considérons $[\theta_{k-\Xi}] = [6, 7]$. Si le "facteur de découpage" vaut ν=10 alors les intervalles suivants seront générés :

$$\begin{array}{rl} [\theta_{(k-\Xi),1}] & = [\underline{6}, \overline{6.1}] \\ [\theta_{(k-\Xi),2}] & = [\underline{6.1}, \overline{6.2}] \\ [\theta_{(k-\Xi),3}] & = [\underline{6.2}, \overline{6.3}] \\ [\theta_{(k-\Xi),4}] & = [\underline{6.3}, \overline{6.4}] \\ \ldots & \ldots \\ [\theta_{(k-\Xi),10}] & = [\underline{6.9}, \overline{7}] \end{array}$$

Le processus de découpage conduit à avoir :

$$\nu DICSP_k = \left(DICSP_k^1,\, DICSP_k^2, ...,\, DICSP_k^\nu\right) \tag{5.29}$$

avec $DICSP_k^i = \left(ICSP_k^i,\, ICSP_{k-1}^i,\, ...,\, ICSP_{k-\Xi+1}^i\right)^T$

L'ensemble des pavés $\begin{pmatrix} [x^{init}] \\ [y^{init}] \\ [\theta^{init}] \end{pmatrix}$, avec $i \in [1, \nu]$, représente l'état initial de $\begin{pmatrix} [x_{(k-\Xi),i}] \\ [y_{(k-\Xi),i}] \\ [\theta_{(k-\Xi),i}] \end{pmatrix}$. Chaque $[\theta_i^{init}]$ est défini par l'équation 5.11.

La figure 5.11 présente le principe de la méthode. Contrairement, au principe présenté par la figure 5.2, on peut remarquer ici des étapes supplémentaires comme le découpage d'une variable et le regroupement des solutions. Lorsqu'une donnée GPS est disponible, on procède à la création (le cas initial) ou à la mise à jour d'un DICSP qui sera ensuite décomposé en ν parties. Un exemple de DICSP est donné en annexe F. La décomposition se base sur le découpage de la variable $[\theta_{k-w}]$ (il s'agit de la variable $[\theta_{k-1}]$ à l'instant $t = k - w + 1$). Une fois les DICSP résolus, il s'ensuit une opération d'enveloppe intervalle qui fournit la pose corrigée $[X_k][Y_k][\theta_k]$ du véhicule. Et le cycle se poursuit.

Chaque DICSP est résolu séparément avec un algorithme de consistance locale (HC4 dans notre cas). Ensuite, la solution au problème de localisation est donnée par la réunion des résultats obtenus. Il définit la pose courante du véhicule. Lorsque de nouvelles mesures sont disponibles, la procédure recommence. Le résultat précédemment obtenu définit le nouveau pavé initial $\begin{pmatrix} [x^{init}] \\ [y^{init}] \\ [\theta^{init}] \end{pmatrix}$. Il arrive qu'un ou plusieurs DICSP n'aient pas de solution (un de ses domaines est vide). Ceci est un comportement attendu qui permet une correction rapide en éliminant

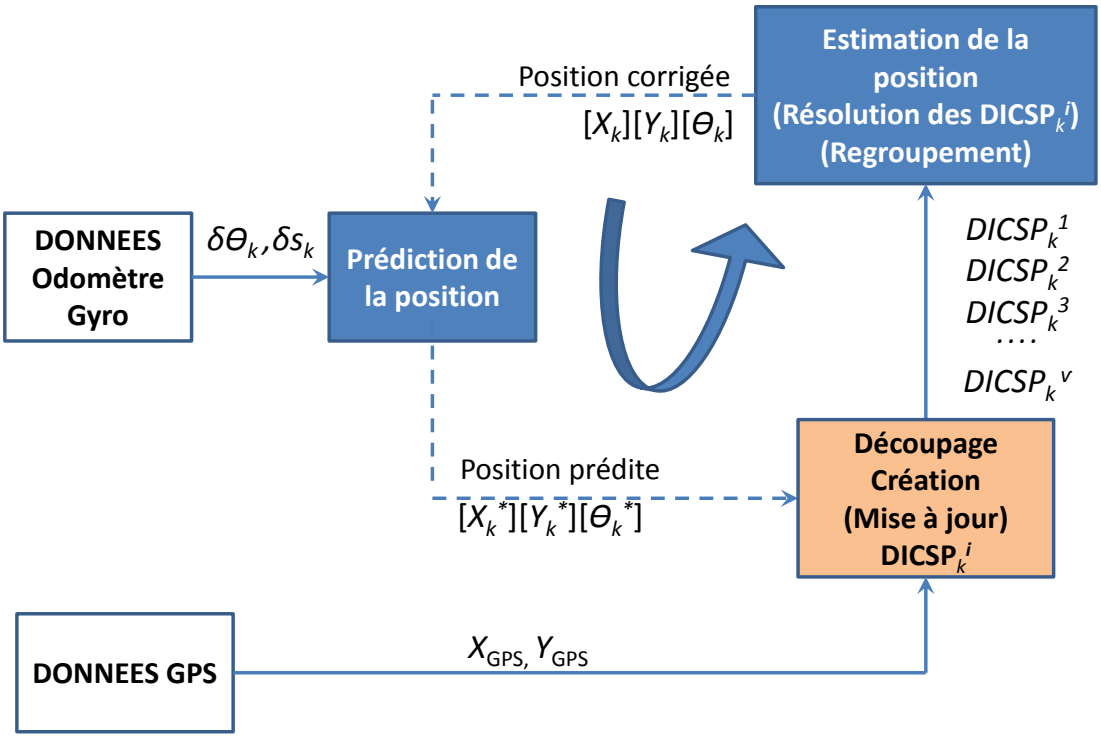


FIGURE 5.11 – Principe de la localisation avec découpage

un ensemble de solutions erronées. L'algorithme 5.3 donne une description succincte de la méthode et une description plus détaillée est fournie par l'algorithme 5.4. L'algorithme est une version simplifiée de la méthode.

- Ligne 1 : On procède à un découpage de la variable $[\theta_{k-\Xi}]$ (il s'agit de la plus ancienne variable représentant le cap dans le DICSP initiale) en ν intervalles
- Ligne 2 : Pour chaque nouveau domaine de $[\theta_{k-\Xi}]$, un DICSP est créé.
- Ligne 3 : Chaque DICSP est soumis à un processus de résolution par un algorithme ICP. Il peut utiliser HC4, BC3, ...
- Ligne 6 : Les nouveaux domaines, résultats de processus de l'étape précédente, sont regroupés avec la fonction $Hull$, une enveloppe intervalle.
- Ligne 7 : L'enveloppe intervalle constitue un vecteur d'intervalles et est renvoyé comme position du véhicule.

Algorithm 5.3 ICP with Splitting

Function : ICPsplitting($ICSP_{init}$)

1: $[\theta_{(k-\Xi),i}]$←Split($[\theta_{k-\Xi}]$,ν)
2: csp_i←Build ($ICSP_{init}$,$[\theta_{(k-\Xi),i}]$)
3: **for** $i \leftarrow 1 : \nu$ **do**
4: $\quad [\mathcal{X}_i] \leftarrow$ ICP(csp_i)
5: **end for**
6: $[\mathcal{X}_{res}] \leftarrow$ Hull($[\mathcal{X}_i]$)
7: **return** $[\mathcal{X}_{res}]$

L'algorithme 5.4 donne une version plus détaillée de la méthode. Il correspond à la description de la figure 5.11. Il est utile de comprendre certaines lignes :

- Ligne 1 : Calcul de S, la taille du domaine des $[\theta]$ après le découpage.
- Ligne 3 : Création des DICSP. Cette étape peut être perçue comme une mise à jour des DICSP.
- Ligne 7 : On peut remarquer que c'est seulement le dernier $[\theta]$ qui est découpé. Les autres variables sont gardées identiques et conservent leurs domaines initiaux dans les nouveaux DICSP.
- Ligne 14 : Appel à HC4 pour résoudre le DICSP courant.
- Ligne 15 : Après l'étape de résolution, on enlève les domaines de $[\theta]$ dont la résolution des DICSP a conduit à des ensembles vides.

Algorithm 5.4 ICP with Splitting

Function : ICPsplitting($ICSP_{init}$, ν)

1: $\mathbf{S} \leftarrow \frac{w([\theta_{k-\Xi}])}{\nu}$ ▷ **S** est la largeur intervalle des $[\theta]$ à l'instant $k-\Xi$ des ICSP dérivés
2: **for** $i \leftarrow 1 : \nu$ **do**
3: **for** $j \leftarrow (k-\Xi) : k$ **do** ▷ Création du i^{eme} ICSP
4: $[x_j] \leftarrow [x_j^{init}]$
5: $[y_j] \leftarrow [y_j^{init}]$
6: **if** $(j{=}k-\Xi)$ **then**
7: $[\theta_j] \leftarrow [\underline{\theta_{k-\Xi}} + (i-1)\underline{\mathbf{S}}, \overline{\theta_{k-\Xi}} + i\overline{\mathbf{S}}]$
8: **else**
9: $[\theta_j] \leftarrow [\theta_j^{init}]$
10: **end if**
11: $[B_i] \leftarrow [B_{i-1}] \times [x_j] \times [y_j] \times [\theta_j]$
▷ Nous avons trois contraintes pour chaque étape
12: $C_i \leftarrow C_{i-1} \cup \{c_{1,j}, c_{2,j}, c_{3,j}\}$
13: **end for**
14: $[B'_i] \leftarrow$ HC4(C_i,$[B_i]$) ▷ Appel de HC4
15: **if** $[B'_i] \neq \emptyset$ **then**
16: **for** $j \leftarrow (k-\Xi) : k$ **do** ▷ La phase du regroupement
17: $[x_j^{res}] \leftarrow [x_j^{res}] \cup [x'_j]$
18: $[y_j^{res}] \leftarrow [y_j^{res}] \cup [y'_j]$
19: $[\theta_j^{res}] \leftarrow [\theta_j^{res}] \cup [\theta'_j]$
20: **end for**
21: **end if**
22: **end for**
23: **return** $[x_j^{res}] \times [y_j^{res}] \times [\theta_j^{res}]$

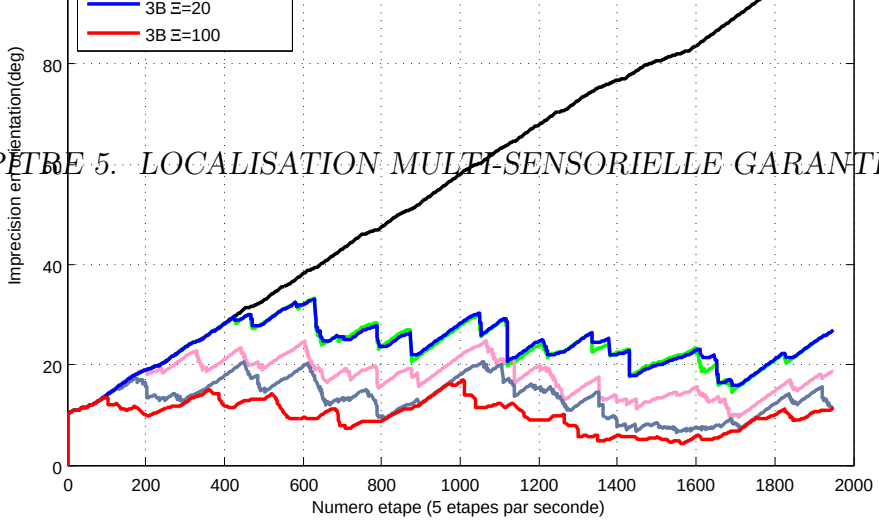


FIGURE 5.12 – Incertitude d'orientation

Algorithmes Paramètres	ICPS Ξ=20 ν=20	ICPS (Ξ =40 ν=20)	HC4 (Ξ=20)	3B (Ξ=20)
Durée (ms)	13.32	172	0.542	5257

Tableau 5.3 – Temps de calcul moyen de l'algorithme pour un pas de temps de localisation

5.7.3 Résultats

Nous présentons, dans cette section, les résultats obtenus avec notre approche. Ces résultats sont comparés avec ceux des autres méthodes ICP.

Dans la figure 5.12, on remarque que l'incertitude sur le cap est corrigée avec notre approche, de même qu'avec l'approche 3B. Avec HC4 seul, l'incertitude ne fait que s'accroître. La consistance 3B traite les bornes de chaque domaine des variables. Il calcule la solution du DICSP de localisation pour chaque borne. Et s'il n'y a pas de solution pour une borne donnée, la borne en question est éliminée (supprimée) du domaine, et ainsi de suite. De cette manière, les domaines sont réduits borne après borne.

3B a été utilisé ici avec une fenêtre de taille ($\Xi = 20$ et $\Xi = 100$) dans la figure 5.12. Lorsqu'on l'utilise avec une fenêtre de taille 20, 3B fournit un résultat très proche de celui d'ICPS à (Ξ=20 ν=20). Mais, avec une fenêtre de taille (Ξ=100), 3B est meilleur que notre méthode avec comme paramètres (Ξ=100 ν=20).

Nous avons mesuré le temps de calcul pour un tour complet (400 secondes) : 2000 DICSP de localisation ont donc été résolues avec chacune des méthodes. Le tableau 5.3 montre que HC4 fonctionne plus vite que les autres méthodes même si on a vu son incapacité à corriger l'incertitude sur le cap. 3B est naturellement très consommateur en termes de temps de calcul (voir tableau 5.3). Par contre,

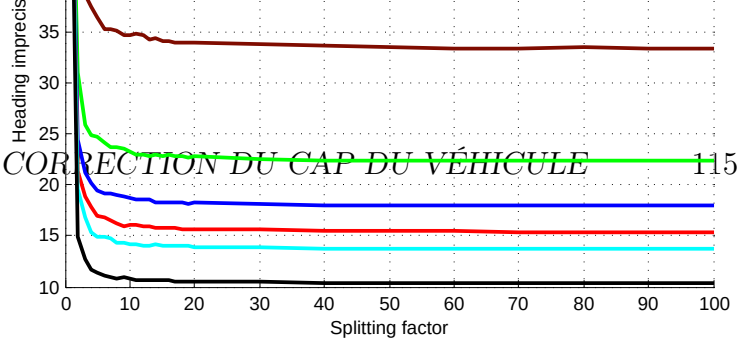


FIGURE 5.13 – Incertitude moyenne en orientation pour différents facteurs de découpage

notre méthode est rapide. 3B élimine les bornes alors que notre méthode élimine de larges intervalles.

L'effet du découpage ($\nu = 20$) peut être observé avec trois tailles de fenêtre Ξ (20, 40 et 100) présentées en fig. 5.12. L'incertitude sur le cap est stabilisée entre 10 et 20 degrés. Cela montre que la taille de fenêtre est un paramètre important dans la correction de l'incertitude.

A partir de la figure 5.13, on peut déduire que la valeur du facteur de découpage peut être définie à 20 : l'efficacité du découpage diminue suivant une fonction hyperbolique. A partir d'un facteur de découpage à 20, on n'observe plus de grands changements au niveau des courbes.

On peut également remarquer à travers la figure 5.14 qu'outre le découpage, la taille de fenêtre joue un rôle non négligeable sur la correction de l'imprécision en orientation. En définissant des valeurs élevées (figure 5.15) pour les paramètres (Ξ,ν), on ne peut pas utiliser la méthode dans le cadre des applications de type temps réel. Le temps de calcul augmente en fonction des deux paramètres. Par exemple, pour des valeurs (Ξ=100 ν=100), nous avoisinons un temps de calcul moyen par DICSP de 6 secondes. Pour pouvoir fonctionner en temps-réel, les résultats de localisation doivent être fournis au système dans un délai de 200 ms.

Néanmoins, plus les valeurs des variables (Ξ et ν) sont grandes, et moins il y a d'erreurs de localisation. Si ν=20 et $\Xi = 40$, alors le temps de calcul de notre algorithme est de 172ms. Ainsi, l'algorithme proposé peut fonctionner en temps-réel assurant en moyenne 16° d'imprécision ($\pm 8°$). Cette incertitude est moins élevée que les valeurs obtenues par les approches intervalles classiques (CP et BESE (Lambert et al., 2009)) sur les mêmes données.

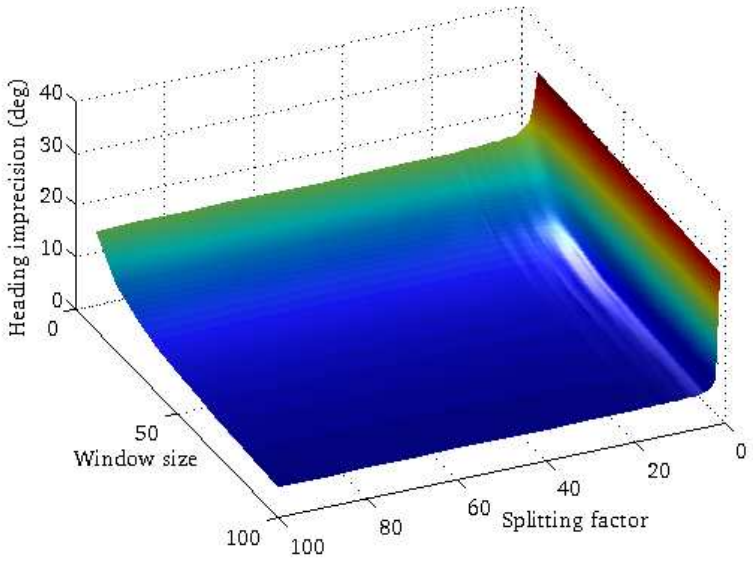


FIGURE 5.14 – Incertitude moyenne en orientation pour différents facteurs de découpage en fonction de la taille de la fenêtre

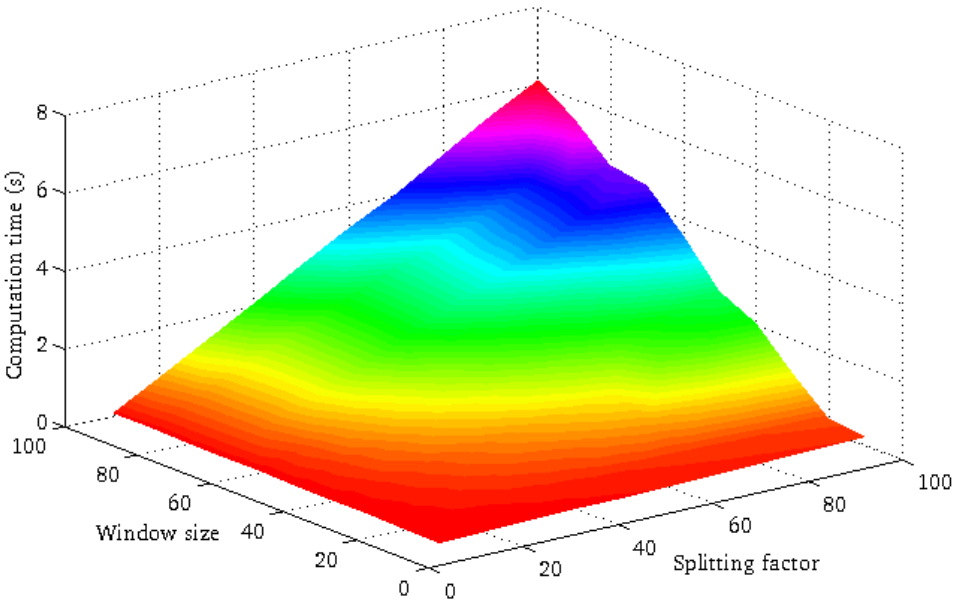


FIGURE 5.15 – Temps de calcul moyen pour différents facteurs de découpage en fonction de la taille de la fenêtre

5.7.3.1 Consistance en orientation

La figure 5.16 montre que le résultat en orientation est consistant avec la référence. Elle représente les intervalles en orientation fournis par ICPS. Ils sont affichés en coordonnées polaires. Le rayon du cercle (en noir) représente la durée du parcours soit $400s$ (environ $7mn$). Chaque point (en bleu ou en rouge) représente un instant t et une valeur θ (en degré) de l'orientation calculée. Chaque instant t est représenté par un cercle. En bleu, sont affichées, les bornes minimum et maximum de l'intervalle calculé à chaque instant tout au long du déplacement. En rouge, on peut observer l'orientation de référence calculée à partir des positions de référence fournies par le GPS-RTK. Ces valeurs angulaires sont affichées en degrés. Elles sont en coordonnées polaires pour simplifier l'analyse. L'orientation de référence est, à tout instant, incluse dans l'intervalle calculé avec notre approche. Ici, ICPS est utilisé avec les paramètres ($\nu = 20$ et $\Xi = 20$). La consistance peut être vérifiée pour l'ICPS avec d'autres paramètres aussi. On peut aussi observer que le véhicule démarre avec une orientation de 165 degrés (orientation initiale) et il termine son parcours dans une orientation assez proche de 150 degrés. Ceci indique qu'il se retrouve à peu près dans la même situation qu'au départ.

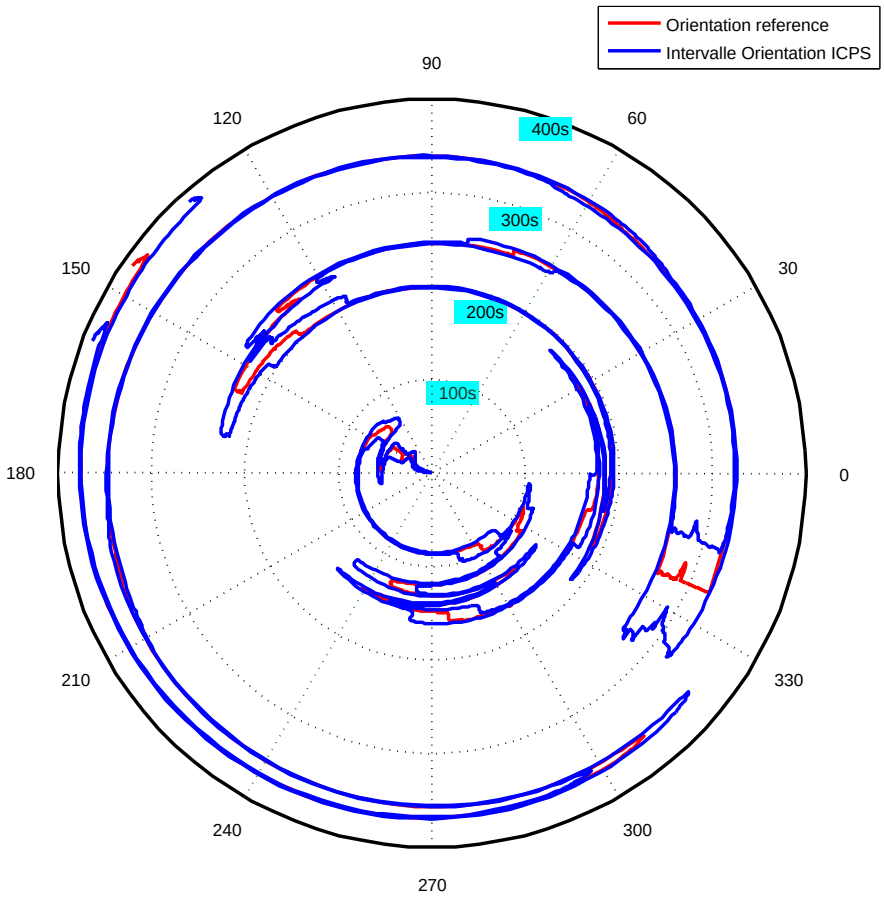


FIGURE 5.16 – Consistance en orientation (les valeurs sont en degrés sur le cercle)

5.7.4 Comparaison d'ICPS [60] avec l'EKF

Nous allons comparer ici la méthode proposée ICPS "Interval Constraint Propagation with Splitting" avec le filtre de Kalman étendu.

5.7.4.1 Comparaison théorique

La gestion des incertitudes : L'approche ICPS proposée considère des erreurs bornées (de bornes connues) alors que l'approches probabilistes traitent les erreurs gaussiennes.

Résultats retournés : ICPS retourne des pavés qui contiennent la pose du véhicule de façon garantie (on recherche toutes les solutions cohérentes avec le modèle et les données du problème formulé). Cette approche est déterministe. Elle représente les résultats sous la forme de vecteurs d'intervalles alors que l'EKF donne une valeur qui a la plus forte probabilité ainsi qu'une matrice de covariance représentant l'incertitude.

5.7.4.2 Comparaison expérimentale

Cette comparaison est réalisée avec le filtre de Kalman étendu (EKF). Notons que dans toute cette comparaison notre algorithme ICPS utilise les paramètres ($\nu = 20$, $\Xi = 40$). Afin de garantir l'équité, les deux algorithmes utilisent les mêmes données. Un écart-type σ des erreurs des capteurs est fourni à l'EKF, tandis que pour l'ICP une borne d'erreur correspond à un niveau de confiance à 0.99 (3σ) est retenue. Rappelons qu'un filtre fournit de bons résultats si son couloir d'incertitude est mince tout en incluant la valeur 0. Le couloir doit ainsi contenir la valeur de référence. La figure 5.17 compare l'EKF et l'ICPS. On peut remarquer que l'EKF produit un couloir très mince ; ce qui laisserait penser à une grande précision. Cependant, ce couloir n'inclut pas la valeur 0 à certains moments du parcours. L'EKF est donc inconsistant par moment. On peut ainsi remarquer une inconstance considérable juste avant l'instant $t = 200s$. Cela est dû au fait que des mesures GPS sont largement biaisées sur cette partie du trajet. L'ICPS fournit un couloir plus large mais consistant.

Pour que l'EKF soit consistant, il faut considérer une erreur de 25σ (voir la figure 5.18). On peut remarquer que le couloir de l'ICPS est globallement plus fin que celui de l'EKF. Nous avons remarqué que l'EKF peut converger vers une mauvaise solution en présence de mesures fortement biaisées. Cela crée une importante inconsistance locale. L'EKF sous-estime sa matrice de covariance en présence de mesures biaisées répétées. Des expériences (Lambert et al., 2009) réalisées avec le filtre particulaire sur les mêmes données conduisent à des résultats assez proches.

60. ICP with Splitting

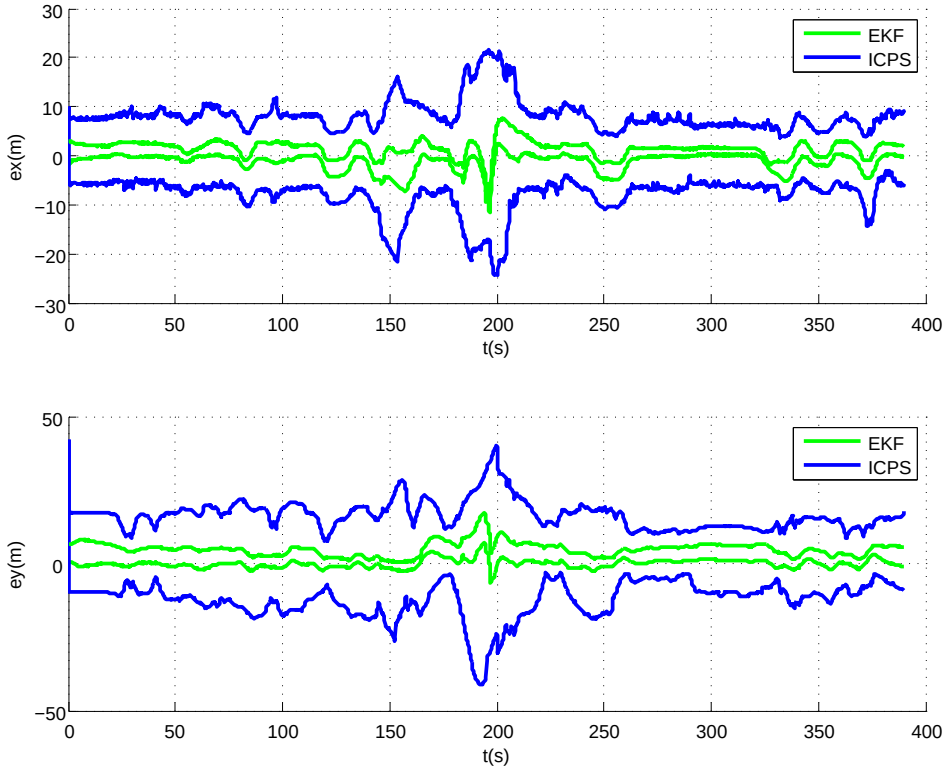


FIGURE 5.17 – Comparaison ICPS (3σ) et l'EKF inconsistant à 3σ d'erreur

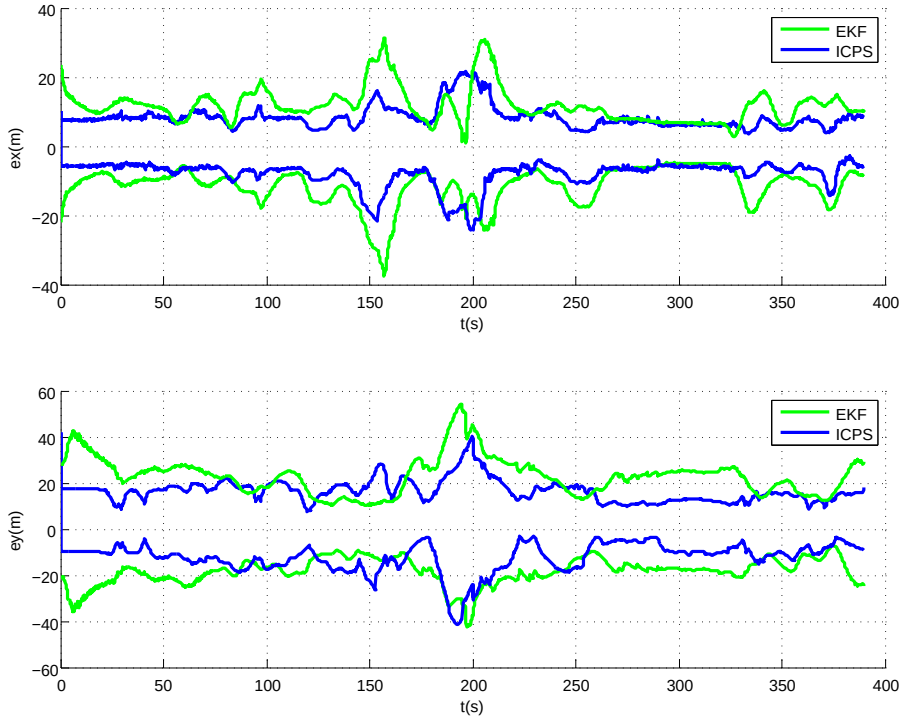


FIGURE 5.18 – Comparaison ICPS (3σ) et l'EKF consistant à 25σ d'erreur

5.7.5 Conclusion

A partir du constat que l'algorithme de consistance 3B est le seul algorithme ICP capable de contracter le domaine intervalle correspondant à l'incertitude en orientation, nous avons proposé dans cette section, un algorithme simple à même de corriger l'incertitude sur le cap lors du déplacement du véhicule (équipé d'un récepteur GPS, un gyromètre et des odomètres).

Il fournit un résultat meilleur que 3B. νICPS a été validé sur des données expérimentales et a montré qu'il pouvait réduire l'incertitude avec une imprécision autour de $\pm 8^\circ$. Le temps de calcul augmente avec le facteur de découpage. Nous recommandons l'utilisation de $\Xi = 20$ et $\nu = 40$ afin de maintenir un temps de calcul raisonnable (172 ms pour chaque pas de temps de localisation) ainsi qu'une bonne localisation.

5.8 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la localisation de véhicule. A cet effet, un bref état de l'art sur la localisation a été présenté. Différentes situations de localisation telles que le suivi de position et la localisation globale ont été abordées pour faciliter la compréhension des différentes approches introduite par la suite. Plusieurs types d'approches sont souvent utilisées pour résoudre le problème de la localisation de véhicule, entre autre, les approches basées sur la théorie des possibilités, les approches bayésiennes, etc. Les approches ensemblistes constituent une alternative à ces méthodes et plusieurs études sont en cours. Elles ont l'avantage de fournir des solutions robustes et garanties. Nous avons étudié une de ces approches qui formule le problème de localisation comme un problème de satisfaction de contraintes sur les intervalles. Cette méthode combine le concept d'analyse par intervalles et la propagation de contraintes.

Nous avons formulé dans ce chapitre, le problème de localisation de véhicule sous forme de DICSP. Une fenêtre temporelle glissante est utilisée. A chaque fois qu'une donnée GPS est disponible, si le nombre d'étapes est inférieur à la taille définie pour la fenêtre temporelle, alors on réalise une restriction, c'est-à-dire qu'on ajoute de nouvelles contraintes et de nouvelles variables avec leur domaine respectif au problème. Et lorsque le nombre d'étapes est supérieur à la taille de fenêtre alors on procède, si une donnée GPS est disponible, à une relaxation suivie d'une restriction. Ceci permet de garder constant le nombre de contraintes exigées par la taille de la fenêtre retenue.

Une difficulté d'ordre méthodologique a été surmontée. En effet, la décomposition des contraintes du problème en contraintes binaires basée sur l'algorithme de Waltz (généralement utilisé pour la localisation) peut induire des erreurs et représente une tâche fastidieuse. Nous proposons d'utiliser des algorithmes de consistance locale qui ne nécessitent pas de décomposition. Nous avons développé un logiciel permettant de tester cette proposition en se basant sur un solveur connu. Nous avons comparé plusieurs algorithmes d'ICP.

En outre, on notera, que les algorithmes de consistance locale ne corrigent pas les imprécisions au niveau du cap du véhicule. Nous avons proposé une solution corrigeant cette incertitude. Cette méthode est robuste et fournit des résultats consistants avec les mesures capteurs. Nous avons comparé notre méthode ICPS au filtre de Kalman étendu. La comparaison a utilisé les mêmes données pour les deux algorithmes et montre l'intérêt de l'ICPS qui fournit un résultat consistant contrairement à l'EKF.

Chapitre 6

Conclusion et perspectives

Sommaire

6.1 Conclusion 124
6.2 Perspectives 125
6.3 Publications scientifiques 126

6.1 Conclusion

Les véhicules deviennent de plus en plus autonomes grâce aux systèmes de navigation qui y sont embarqués. Les techniques de perception et de traitement d'information évoluent vite et sont de plus en plus fiables. L'autonomie grandissante des véhicules est un facteur susceptible de transformer le mode de transport moderne en rendant les trajets plus sûrs, plus efficients, limitant les accidents, les embouteillages ainsi que les émissions de gaz polluants.

Cette thèse s'inscrit dans le cadre générale de la localisation de véhicule, et plus particulièrement dans la problématique de localisation en environnement d'extérieur. Nous nous sommes intéressés à l'utilisation des approches ensemblistes notamment l'analyse par intervalles afin de résoudre ces problèmes. Les approches probabilistes sont classiquement utilisées et ont fait leur preuve dans la résolution de ce genre de problème.

Nous avons présenté notre travail en 6 chapitres. Dans le premier chapitre, nous avons introduit brièvement les problèmes. Dans le deuxième chapitre, nous avons présenté les plate-formes expérimentales utilisées dans cette thèse ainsi que les capteurs embarqués. Les outils mathématiques nécessaires à la compréhension du travail ont été présentés dans les chapitres 3 et 4, notamment l'analyse par intervalles et de la propagation de contraintes sur les intervalles. Le chapitre 5 a été consacré à la problématique de la localisation. Pour ce faire, un état de l'art de la localisation a été présenté. Ensuite, une difficulté d'ordre méthodologique à savoir la décomposition des contraintes d'un problème en contraintes binaires a été présentée et une solution a été proposée. L'étude de plusieurs solveurs nous a conduit à en choisir un qui correspond à nos besoins de comparer plusieurs algorithmes. Nous avons formulé le problème de localisation de véhicule sous forme de DICSP. Une fenêtre temporelle glissante est utilisée. Ayant noté que les algorithmes de consistance locales ne corrigent pas l'imprécision en orientation, nous avons proposé une solution. La méthode proposée ne décompose pas les contraintes et peut fonctionner dans une application respectant les contraintes du temps-réel. Nous proposons donc une méthode pour la localisation de véhicule pouvant réduire l'incertitude du cap. Cette méthode utilise HC4 comme algorithme de bas niveau. Cependant, d'autres algorithmes de consistance locale peuvent aussi être utilisés. Les résultats de cet algorithme sont comparés à ceux des algorithmes comme HC4, 3B et le filtre de Kalman étendu. Un des avantages de la méthode est qu'il est facile de changer les modèles d'évolution de véhicules sans devoir tout réécrire. L'approche proposée est adaptable et utilisable en robotique d'intérieur.

6.2 Perspectives

Aux termes de ce travail, plusieurs pistes de recherche restent intéressantes à explorer. Nous présenterons dans cette section quelques unes.

Améliorer la localisation

Il serait intéressant de permettre de choisir de façon dynamique le facteur de découpage en fonction de la visibilité des satellites. A l'état actuel de l'algorithme, nous fixons une valeur au facteur de découpage. Cette valeur est utilisée tout au long du parcours. Nous pensons qu'on peut l'adapter en fonction de la qualité des données capteurs. En situation de masquage du GPS, par exemple, on pourrait l'augmenter et le réduire sur un trajet normal.

A court terme, nous nous proposons d'appliquer les techniques de correction d'incertitude au monoSLAM ensembliste.

Nous pensons que l'on peut encore améliorer l'implémentation de nos travaux afin d'obtenir des résultats plus compétitifs par rapport aux approches probabilistes.

Application à la localisation collaborative de multi-véhicules

On pourrait étudier la collaboration multi-véhicules avec les approches ICP. En effet, plusieurs véhicules pourraient partager des informations sur leur position représentées par des vecteurs d'intervalles. Ceci permettrait d'augmenter la précision de la localisation.

Ajout d'autres capteurs

Nous pensons qu'il serait intéressant de combiner le GPS avec la vision monoculaire. Ceci permettrait certainement d'améliorer les résultats. On pourrait aussi étudier les techniques de fusion de données de manière à définir des mécanismes de confiance pour le choix les données GPS et les données provenant de la vision dans une situation donnée.

6.3 Publications scientifiques

Revue à comité de lecture

– I. K. Kueviakoe, A. Lambert, and P. Tarroux. "**Comparison of interval constraint propagation algorithms for vehicle localization**". Journal of Software Engineering and Applications, 5 :157–162, 2012.

Colloques internationaux

– I. K. Kueviakoe, A. Lambert, and P. Tarroux. "**Vehicle Localization based on Interval Constraint Propagation Algorithms**", IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, 179-184, 2013

– I. K. Kueviakoe, A. Lambert, P. Tarroux, "**A Real-Time Interval Constraint Propagation Method for Vehicle Localization**", IEEE International Conference on Intelligent Transportation Systems, 1707-1712, 2013

– Kangni Kueviakoe, Alain Lambert, Bastien Vincke, Abdelhafid Elouardi, Emmanuel Seignez, Lu Wu, Philippe Tarroux, "**Interval Based Outdoor Vehicle SLAM**", International Conference on Electrical, Control and Automation Engineering, 387-391, 2013

Annexe A

L'algorithme SIVIA

Nous présentons dans cette annexe, l'algorithme SIVIA [61] de Luc Jaulin (Jaulin and Walter, 1993). Il est basé sur le principe d'inversion ensembliste. L'inversion ensembliste consiste à trouver l'ensemble $\mathbb{X}$

Connaissant un ensemble $\mathbb{Y} \subset \mathbb{R}^m$ et f , une fonction continue de $\mathbb{R}^n$ dans $\mathbb{R}^m$ pour laquelle on peut

$$\mathbb{X} = [f]^{-1}(\mathbb{Y}) = \{x \in \mathbb{R}^n | f(x) \in \mathbb{Y}\} \tag{A.1}$$

La figure A.1 présente le problème et la figure *A.2* indique l'inversion ensembliste comme solution et à la recherche de l'ensemble $\mathbb{X}$.

Cette présentation de SIVIA est inspiré de (Jaulin and Walter, 1993). SIVIA est un algorithme recursif (voir l'algorithme A.1). Les sous pavages $\underline{\mathbb{X}}$ et $\overline{\mathbb{X}}$ sont initialisées comme étant vides.

SIVIA réalise des partitions sur un pavé initial. Et crée ainsi plusieurs autres pavés. L'image de chaque pavé est ensuite évaluée à l'aide de la fonction d'inclusion sur f. Le but est de démontrer que tous les points d'un pavé appartiennent à $\mathbb{X}$. SIVIA utilise un test d'inclusion pour l'étape d'élimination.

61. SIVIA : Set Inverter Via Interval Analysis (en français : Inversion ensembliste avec de l'analyse par intervalles)

Algorithm A.1 SIVIA

Function : SIVIA($[x], [f], \mathbb{Y}, \varepsilon, \underline{\mathbb{X}}, \overline{\mathbb{X}}$)

if then $[f]([x]) \cap \mathbb{Y} = \emptyset$
▷ Le pavé n'est pas solution
    return ;
end if
if then $[f]([x]) \subset \mathbb{Y}$
    $\underline{\mathbb{X}} \leftarrow \underline{\mathbb{X}} \cup [x]$;
    $\overline{\mathbb{X}} \leftarrow \overline{\mathbb{X}} \cup [x]$;
    return ;
end if
if then $w([x]) < \varepsilon$
    $\overline{\mathbb{X}} \leftarrow \overline{\mathbb{X}} \cup [x]$;
    return ;
end if
$(\underline{\mathbb{X}}, \overline{\mathbb{X}}) \leftarrow SIVIA(L[x], [f], \mathbb{Y}, \varepsilon, \underline{\mathbb{X}}, \overline{\mathbb{X}})$;
$(\underline{\mathbb{X}}, \overline{\mathbb{X}}) \leftarrow SIVIA(R[x], [f], \mathbb{Y}, \varepsilon, \underline{\mathbb{X}}, \overline{\mathbb{X}})$;
return $(\underline{\mathbb{X}}, \overline{\mathbb{X}})$;

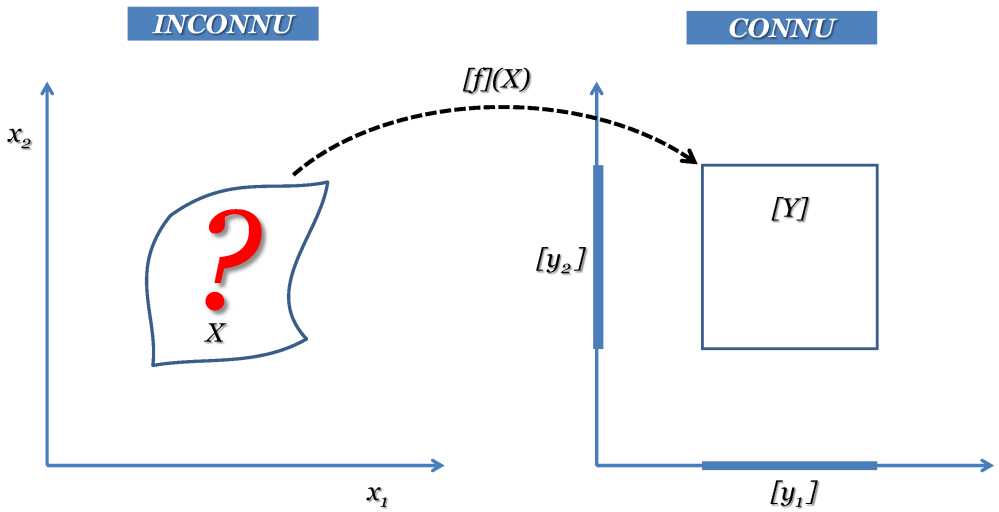


FIGURE A.1 – Inversion ensembliste (le problème)

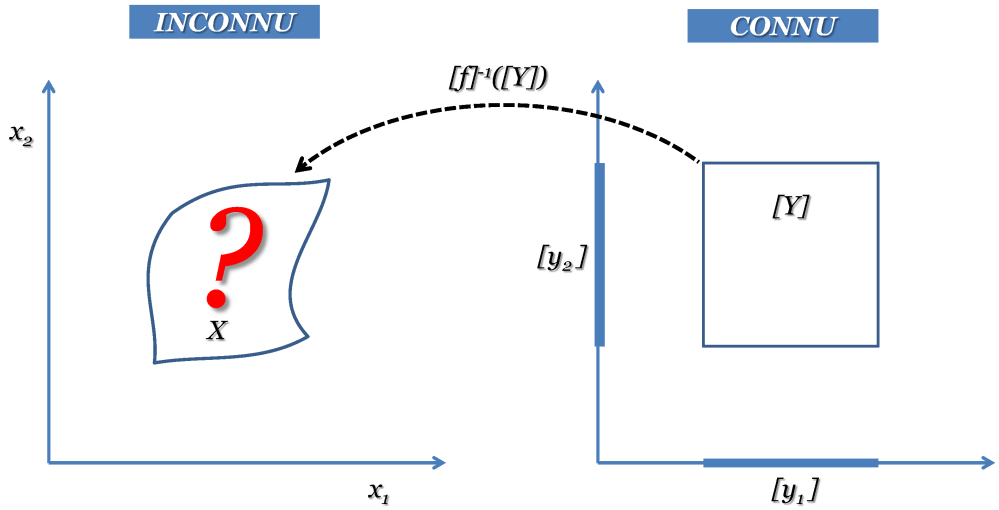


FIGURE A.2 – Inversion ensembliste (la solution)

Annexe B

Méthode intervalle de Branch&Bound

Nous présentons dans cette section un algorithme générique de Branch&Bound basé sur les intervalles. L'algorithme de Branch&Bound Lawler and Wood (1966) est une méthode générique connu en français sous l'appellation "Séparation et Évaluation". Il est utilisé pour résoudre des problèmes d'optimisation. Il permet d'énumérer de façon explicite toutes les solutions possibles d'un problème. On peut éviter d'énumérer un grand nombre de mauvaises solutions en analysant en amont les propriétés du problème. En général, un algorithme de ce type énumère seulement les bonnes solutions. Deux étapes constituent le déroulement de l'algorithme : l'évaluation (Bound) et la séparation (Branch). L'étape de séparation consiste à diviser le problème en plusieurs sous-problèmes ayant chacun ses propres solutions. L'ensemble des solutions permet de construire une hiérarchie sous la forme d'un arbre de recherche. La phase d'évaluation permet de calculer l'optimal de l'ensemble des solutions. Elle permet aussi de prouver qu'un ensemble donné ne contient pas de solution optimale. Cet algorithme a été aussi adapté à l'analyse par intervalles. L'algorithme B.1 présente la description générale de Branch&Bound adapté à l'analyse par intervalle (IBB [62])(Walster et al., 1985) (Kearfott, 1987)(Kearfott, 1996b). On peut remarquer que l'algorithme SIVIA(Jaulin and Walter, 1993) est une forme de IBB.

Les IBB utilisent des pavés (boites) pour représenter l'espace de recherche et une fonction d'inclusion pour réaliser la phase d'évaluation.

Ces algorithmes dépendent de l'existence de la fonction d'inclusion et d'un pavé initiale. Décrivons quelques étapes de cet algorithme :

- Sélection
  Parmi les boites x^j disponibles dans L_w, choisir une boite x telle que $f(x) = min\{f(x^j)|x^j \in L_w\}$
- Evaluation
  La fonction d'inclusion f est utilisé pour évaluer x
- Eliminition
  Plusieurs techniques peuvent être utilisées pour l'élimination des pavés non solutions. Entre autres, on peut trouver le test du point milieu de l'intervalle,

62. IBB pour Interval Branch&Bound

Algorithm B.1 Interval Branch And Bound

Function : IBB(s, f)

▷ On initialise la liste de travail

1: $L_W \leftarrow \{s\}$

▷ On initialise la liste solution

2: $L_S \leftarrow \emptyset$

3: **while** $(L_W \neq \emptyset)$ **do**

▷ Etape de Sélection

4: Sélectionner une boite x de L_w

▷ Etape d'Evaluation ("Bound" en anglais)

5: Calculer $f(x)$

▷ Eliminition

6: **if** (x ne peut pas être éliminé) **then**

▷ Phase de Séparation, bissection ("Branch" en anglais)

7: Diviser x en x^j, $j = 1, ..., p$

8: **for** $j = 1, ..., p$ **do**

▷ Phase de Finition

9: **if** (x^j satisfait le critère de finition) **then**

10: Sauvegarder x^j dans L_S

11: **else**

12: Sauvegarder x^j dans L_W

13: **end if**

14: **end for**

15: **end if**

16: **end while**

17: **return** L_S

le test de monotonicité, le test de non-convexité,...

- Séparation
  Il s'agit de bissecter le pavé et générer deux autres pavés dans la direction k où k est la coordonnée telle que $w(x_k) = max_i =_{1,...,n} w(x_i)$
- Finition
  Un paramètre ε détermine la précision souhaitée à la solution du problème. Ainsi, une boite x avec $w(x) \leq \varepsilon$ et/ou $w(f(x)) \leq \varepsilon$ est placé dans la liste résultat L_S (la liste contenant les solutions).

Annexe C

Les filtres probabilistes

C.1 Le filtre Bayesien basique

Le filtre Bayésien est l'algorithme le plus généralement utilisé pour calculer les croyances. Nous présentons le filtre Bayésien basique dans l'algorithme C.1. Il s'agit d'un algorithme récursif qui calcule la croyance de l'état courant en fonction de la croyance de l'état précédent.

Algorithm C.1 Le filtre bayesien

Function : Filtre_Bayesien ($bel(\mathcal{X}_{t-1})$, u_t, z_t)

1: **for each** $\mathcal{X}_t$ **do**
2: $\quad \overline{bel}(\mathcal{X}_t) = \int p(\mathcal{X}_t|u_t, \mathcal{X}_{t-1})bel(\mathcal{X}_{t-1})d\mathcal{X}_{t-1}$;
3: $\quad bel(\mathcal{X}_t) = \eta p(z_t|\, \mathcal{X}_t, C)\overline{bel}(\mathcal{X}_t)$;
4: **end for**
5: **return** $bel(\mathcal{X}_t)$

C.2 Filtre de Kalman (KF)

Proposé en 1960 par (Kalman, 1960), le filtre de Kalman permet d'estimer l'état d'un système dynamique à partir de mesures imprécises ou incomplètes. C'est un estimateur optimal pour les systèmes linéaires et les distributions gaussiennes. Il permet de mettre à jour toute l'intégralité d'un système en observant une de ses composantes. En effet, les composantes d'un système sont reliées entre elles par une matrice de covariance qui modélise les relations entre les différents paramètres du système.

Le filtre de Kalman prend en entrées : la commande u_k et les observations Y_k. Ce filtre suit trois étapes principales : l'initialisation, la prédiction et la correction. A partir de l'hypothèse que le signal utile X et le signal observé Y peuvent être représentés par un modèle d'état gaussien markovien, alors on peut écrire :

$$\begin{aligned} X_{k|k-1} &= A_{k|k-1}X_{k-1|k-1} + B_k u_k + w_k \qquad w_k \rightsquigarrow \mathcal{N}(0, Q_k) \\ Y_k &= H_k X_{k|k-1} + v_k \qquad\qquad\qquad\qquad v_k \rightsquigarrow \mathcal{N}(0, R_k) \end{aligned}$$

avec :

– A_k, la matrice d'évolution du système de l'instant $k-1$ à l'instant k,

- B_k, la matrice de commande à l'instant k,
- H_k, la matrice de mesure à l'instant k,
- u_k, la commande supposée connue, à l'instant k,
- w_k, le bruit de processus à l'instant k,
- Q_k, la matrice de variance-covariance du bruit de processus à l'instant k,
- v_k, le bruit de mesure à l'instant k,
- R_k, la matrice de variance-covariance du bruit de mesure à l'instant k.

On considère que les bruits sont indépendants entre eux et indépendants de l'état, gaussiens, blancs et centrés. Dans l'algorithme C.2, nous présentons une implantation du filtre de Kalman. On distingue l'étape d'initialisation du filtre, la phase de prédiction ainsi que celle de correction. Les deux dernières phases se répètent de façon récursive. Lors de la phase d'initialisation, on définit les valeurs initiales pour :

- $\hat{X}_0$, le vecteur d'état
- P_0, la matrice de variance-covariance de l'erreur d'estimation
- Q_0, la matrice de variance-covariance du bruit de processus
- R_0, la matrice de variance-covariance du bruit de mesure.

L'étape de prédiction consiste à calculer $\hat{X}_{k|k-1}$ et $P_{k|k-1}$ et la phase de correction permet de calculer les valeurs de K_k, $\hat{X}_{k|k}$ et $P_{k|k}$ où K_k est le gain de Kalman à l'instant k, I représente la matrice identité de même taille que $P_{k|k-1}$.

Algorithm C.2 Algorithme de filtre de Kalman

- Initialisation

$\hat{X}_0 = E[X_0]$
$P_0 = E[(X_0 - \hat{X}_0)(X_0 - \hat{X}_0)^T]$
$Q_0 = E[(w - \bar{w})(w - \bar{w})^T]$
$R_0 = E[(v - \bar{v})(v - \bar{v})^T]$
$\forall k \in [1, ..., \infty[$

- Prédiction

$$\begin{aligned} \hat{X}_{k|k-1} &= A_{k|k-1} X_{k-1|k-1} + B_{k-1} u_{k-1} \\ P_{k|k-1} &= A_{k|k-1} X_{k-1|k-1} A_{k|k-1}^T + Q_{k-1} \end{aligned}$$

- Correction

$$\begin{aligned} K_k &= P_{k|k-1} H_k^T \left[H_k P_{k|k-1} H_k^T + R_k \right]^{-1} \\ \hat{X}_{k|k} &= \hat{X}_{k|k-1} + K_k \left[Y_k - H_k X_{k|k-1} \right] \\ P_{k|k} &= (I - K_k H_k) P_{k|k-1} \end{aligned}$$

C.3 Le filtre de Kalman étendu (EKF)

En 1979, le filtre de Kalman a été étendu aux cas non linéaires sous l'appelation "EKF"[63] par (Maybeck, 1979). La différence entre le filtre de Kalman standard et l'EKF réside dans le calcul de différentes matrices. Dans le filtre de Kalman, on utilise de "vraies" fonctions linéaires. Alors qu'avec l'EKF, on utilise des matrices jacobiennes qui sont composées de fonctions de Taylor linéarisées au

63. Extended Kalman Filter

premier ordre. L'algorithme C.3 présente le principe du filtre. soient f et h deux fonctions non linéaires. Considérons f, la fonction d'évolution et h, la fonction de mesure (ou d'observation), le problème d'estimation se pose ainsi :

$$\begin{aligned} X_{k|k-1} &= f(X_{k-1|k-1},\, u_k,\, w_k) \qquad w_k \rightsquigarrow \mathcal{N}(0, Q_k) \\ Y_k &= h(X_{k|k-1}, v_k) \qquad\qquad\quad v_k \rightsquigarrow \mathcal{N}(0, R_k) \end{aligned}$$

Le vecteur d'état qui est prédit à l'instant k est $X_{k|k-1}$. w_k est le bruit du processus et v_k, le bruit de mesure. Les deux bruits sont supposés indépendants, blancs gaussiens, centrés et avec une matrice de variance-covariance Q_k et R_k, symétriques et définies positives.

L'implantation de l'EKF est assez analogue à celle du filtre de Kalman classique.

Algorithm C.3 Algorithme de filtre de Kalman étendu

- Initialisation

$\hat{X}_0 = E[X_0]$
$P_0 = E[(X_0 - \hat{X}_0)(X_0 - \hat{X}_0)^T]$
$Q_0 = E[(w - \bar{w})(w - \bar{w})^T]$
$R_0 = E[(v - \bar{v})(v - \bar{v})^T]$
$\forall k \in [1, ..., \infty[$

- Calcul des matrices Jacobiennes des fonctions du système

$$\begin{aligned} A_{x_k} &= \nabla_x f(X, u_k, \bar{w})|_{X=\hat{X}_{k-1|k-1}} \\ H_k &= \nabla_x h(X, \bar{v})|_{X=\hat{X}_{k|k}} \end{aligned}$$

- Prédiction

$$\begin{aligned} \hat{X}_{k|k-1} &= f(\hat{X}_{k-1|k-1},\, u_k,\, \bar{w}) \\ P_{k|k-1} &= A_{x_k} P_{k-1|k-1} A_{x_k}^T + Q_{k-1} \end{aligned}$$

- Correction

$$\begin{aligned} K_k &= P_{k|k-1} H_k^T \left[H_k P_{k|k-1} H_k^T + R_k \right]^{-1} \\ \hat{X}_{k|k} &= \hat{X}_{k|k-1} + K_k \left[Y_k - h(\hat{X}_{k|k-1}, v_k) \right] \\ P_{k|k} &= (I - K_k H_k) P_{k|k-1} \end{aligned}$$

Une des limites de l'EKF concerne la condition de dérivabilité des fonctions qui n'est pas toujours satisfaite. Il est donc impossible dans ces cas de calculer les matrices Jacobiennes. En outre, la linéarisation au premier ordre de Taylor peut tronquer des termes significatives.

Annexe D

Algorithme de Waltz

L'algorithme de Waltz proposé en 1972 dans une thèse (Waltz, 1972) au MIT et publié dans une revue en 1975 (Waltz, 1975), a pour but de reconnaître les objets 3D dans une scène en interprétant les lignes en 2D. Il s'agit de donner une interprétation tridimensionnelle à des dessins 2D. L'objectif principal est de reconnaître des figures (triédriques) 3D à partir de contours en 2D.

Cette méthode a été développée pour résoudre le problème d'étiquettage de scène.

Dans un premier temps les lignes et les sommets sont étiquettés. Dans la figure D.1 (image à gauche), on a 4 traits dont un n'est pas visible. Cet algorithme fonctionne sous réserve :

- que l'objet représenté ne coupe pas le cadre du dessin,
- que jamais plus de trois traits ne se rejoignent en un même point (confère figure),
- que les objets représentés soient des polyèdres (donc des objets à faces planes et à arêtes rectilignes).

Si on lui soumet un objet impossible (non reconnaissable), deux cas se présentent :

- l'algorithme n'en trouve aucune interprétation tridimensionnelle, ce qui signifie qu'il a repéré qu'il s'agit d'une figure impossible (sous les hypothèses qui sont les siennes) ;
- l'algorithme propose une interprétation, dont l'observation attentive montre qu'elle n'est pas globalement cohérente.

Les sommets possibles et reconnaissables par l'algorithme de Waltz (voir figureD.2) sont : 6L, 4T, 3 flèches, 3 étoiles.

La figure D.3 présente des exemples de sommets étiquettés.

Dans l'algorithme de Waltz, on retrouve bien REVISE à l'étape où il faut "Eliminer les marques incompatibles deux à deux". Par exemple, si nous considérons le CSP suivant :

Variables : $x_1,\ x_2,\ x_3$

Domaines (les étiquettes initiales) :

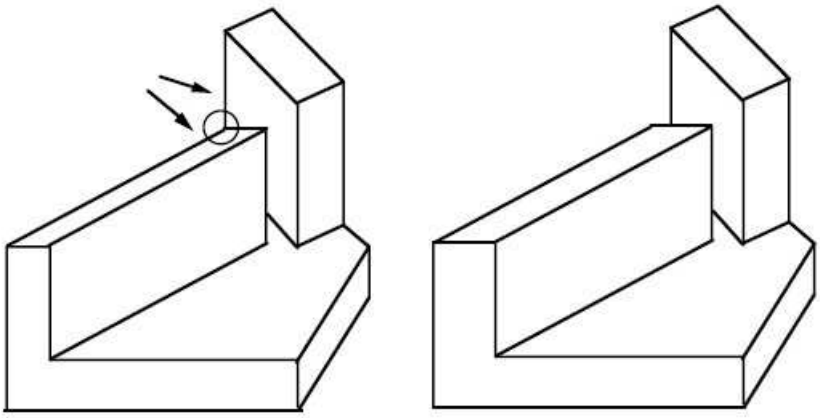
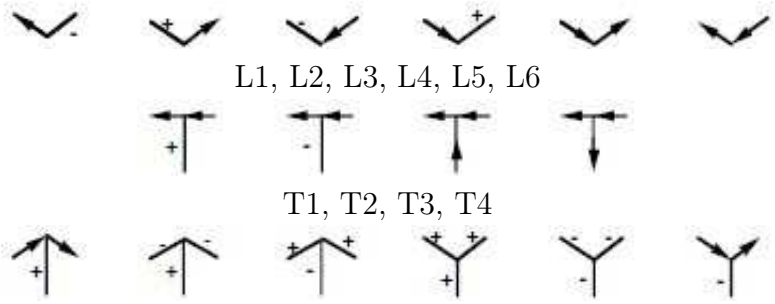


Figure non reconnaissable Figure reconnaissable

FIGURE D.1 – Figure reconnaissable ou non par la méthode Waltz (Waltz, 1975)



L1, L2, L3, L4, L5, L6

T1, T2, T3, T4

F1, F2, F3, E1, E2 et E3

FIGURE D.2 – Les sommets possibles (Waltz, 1975)

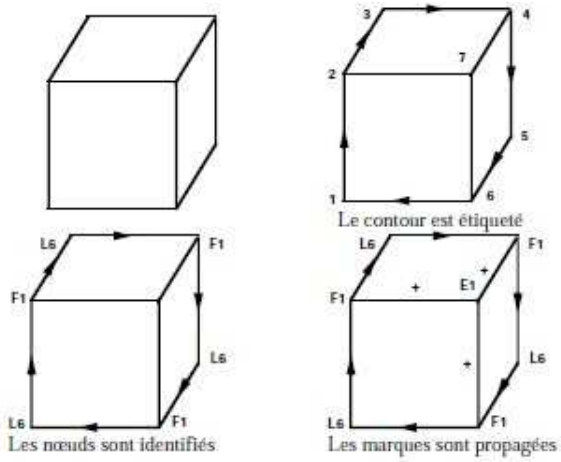


FIGURE D.3 – Propagation des étiquettes (Waltz, 1975)

Algorithm D.1 Waltz

Function : waltz()

```
1: Etiquetter le contour par une flêche afin la matière soit à droite de la flêche
2: while (tous les sommets n'ont pas été examinés) do
3:     Pendre un des sommets les plus contraints        ▷ Plus contraint ou avec moins de choix de d'étiquette
4:     Propager les étiquettes
5:     Vérifier la compatibilité des étiquettes
6:     Eliminer les étiquettes incompatibles deux à deux
7:     if (une arête ne peut être étiquettée) then      ▷ Ensemble des étiquettes vide
8:         sortir avec échec
9:     else
10:        Etiquetter l'arête
11:    end if
12: end while
```

$l_1 = \{a,\ b,\ c,\ d\}$,
$l_2 = \{a,\ b,\ c,\ d\}$,
$l_3 = \{a,\ b,\ c,\ d\}$

Contraintes :
$C_1(x_1, x_2) = \{(a,b),(a,c),(c,d)\}$,
$C_2(x_1, x_3) = \{(a,b),(b,c),(c,d)\}$,
$C_3(x_2, x_3) = \{(a,b),(a,c),(d,d)\}$

La propagation des étiquettes sera effectuée de manière suivante :
1.
$C_1(x_1) \Rightarrow l_1 = \{a,c\}$
$C_1(x_2) \Rightarrow l_2 = \{b,c,d\}$

2.
$C_2(x_1) \Rightarrow l_1 = \{a,c\}$
$C_2(x_3) \Rightarrow l_3 = \{b,d\}$

3.
$C3(x2) \Rightarrow l_2 = \{d\}$
$C3(x3) \Rightarrow l_3 = \{d\}$

4.
$C_1(x_1) \Rightarrow l_1 = \{c\}$

ce qui donne une solution unique :
$x_1 = c$,
$x_2 = d$,
$x_3 = d$

Dans cet exemple, chaque étiquette est finalement réduite à une seule valeur, et on n'a plus besoin de faire une recherche. La complexité de l'algorithme peut être estimée comme suit. Considérons m contraintes, n variables dont les domaines ont la taille maximale d. A chaque itération, REVISE est appliquée au plus $2 \times m$ fois, une fois dans chaque direction. Pour que l'itération ne s'arrête pas, il faut enlever au moins une valeur. Il ne peut pas y avoir plus que $n \times d$ itérations. Par conséquent, la complexité totale ne peut pas dépasser $O(m \times n \times d)$.

La propagation de Waltz peut être facilement généralisée à des contraintes de n variables : REVISE doit alors veiller à ce que pour chaque valeur de l'étiquette L_i, il existe toujours une combinaison de valeurs admises par les étiquettes des autres $(n-1)$ variables pour que la contrainte soit respectée.

Annexe E

Les techniques de consistance sur les domaines finis

Nous donnons ici une description de quelques algorithmes de consistance utilisés pour les domaines finis. Cette présentation des techniques de consistance s'inspire en partie de (Kumar, 1992). Le but des techniques de consistance est d'éliminer les valeurs incohérentes par rapport aux contraintes.

Lorsqu'un CSP est binaire (un CSP binaire est un CSP qui comporte uniquement des contraintes binaires et une contrainte est dite binaire si elle contient uniquement deux variables), il peut être représenté sous la forme de graphe non dirigé, dans lequel les nœuds représentent les variables et les arcs représentent les contraintes. Les contraintes unaires aussi peuvent être représentées dans ces graphes. Les noms donnés à ces techniques de consistance sont donc issus des notions de graphe. On rencontre essentiellement les types de consistances suivants :

- consistance de nœud
- consistance d'arc
- consistance de chemin

E.1 La consistance de nœud

Elle supprime/élimine (sur chaque variable) les valeurs inconsistantes avec les **contraintes unaires.** C'est la technique la plus simple. Le nœud qui représente une variable V dans un graphe de contraintes est nœud-consistant (ou consistant au niveau des nœuds) si pour chaque valeur x du domaine courant de V, chaque contrainte unaire sur V est satisfaite.

Si le domaine D d'une variable V contient une valeur "a" (exemple : 10) qui ne satisfait pas la contrainte unaire sur V, alors l'instanciation de V à "a" va toujours conduire à un échec immédiat. Ainsi, l'inconsistance de nœud peut être éliminée en supprimant simplement cette valeur (exemple : 10) du domaine D. Voir l'algorithme E.1 présente la consistance de nœud.

Algorithm E.1 Consistance de nœuds

Function : NC()

```
1: for each V dans nodes(G) do
2:     for each X dans le domaine D de V do
3:         if Une contrainte unaire sur V est inconsistante avec X then
4:             Supprimer X de D
5:         end if
6:     end for
7: end for
```

E.2 La consistance d'arc

La consistance d'arc est le type de **consistance** est le plus répandu. Il offre un bon rapport simplicité/performance. Il s'attaque aux contraintes binaires individuellement.

Il est important de noter le caractère directionnel de la consistante d'arc. Une contrainte peut être consistante au niveau arc ou pas. Elle est arc-consistante si pour toute valeur de chaque variable dans la contrainte, il existe une valeur pour les autres variables de sorte que la contrainte soit satisfaite. Un CSP peut être arc-consistant si toutes ses contraintes sont arc-consistantes.

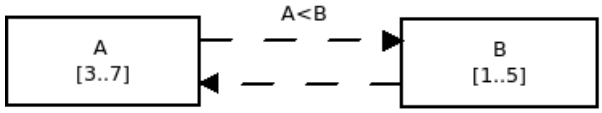


FIGURE E.1 – Aucun arc n'est consistant

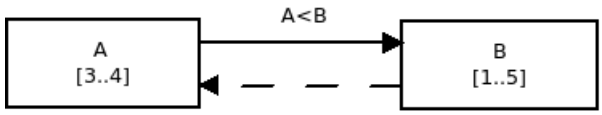


FIGURE E.2 – $A \rightarrow B$ est consistant

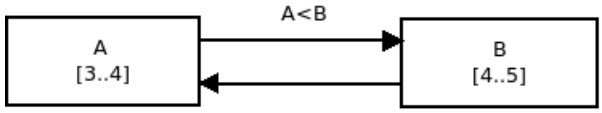


FIGURE E.3 – $A \rightarrow B$ et $B \rightarrow A$ sont consistants

On notera que dans les fugures E.1, E.2 et E.3, on ne compare pas les intervalles, mais plutôt les valeurs à l'intérieur des domaines des variables.

L'arc $A \rightarrow B$ signifie qu'à chaque valeur $a \in A$ correspond une valeur $b \in B$ de sorte que $a < b$.

L'arc $B \rightarrow A$ signifie qu'à chaque valeur $b \in B$ correspond une valeur $a \in A$ de sorte que $a < b$.

Algorithm E.2 REVISE

Function : REVISE(V_i, V_j)

```
1: SUPPRESSION ←FAUX
2: for each X dans D_i do
3:     if (Il n'existe pas de Y dans D_j tel que (X, Y) soit consistant) then
4:         Supprimer X de D_i
5:         SUPPRESSION ←VRAI
6:     end if
7: end for
8: return SUPPRESSION
```

Dans la figure E.1, les arcs $A \rightarrow B$ et $B \rightarrow A$ sont inconsistants. Après avoir éliminé certaines valeurs dans A, on obtient la figure E.2 dans laquelle l'arc $A \rightarrow B$ est consistant mais l'arc $B \rightarrow A$ n'est toujours pas consistant parce qu'il n'existe pas de valeur $a \in A$ telle que $a < 1$ ou $a < 2$ par exemple. Or $\{1\} \in B$ et $\{2\} \in B$. Après l'élimination de 1, 2 et 3 dans B, on obtient $B = [4, 5]$. Dès lors, l'arc $B \rightarrow A$ est aussi consistant.

En général, une contrainte est rendue arc-consistante en propageant le domaine d'une variable à l'autre et vice-versa. On parle ainsi de **révision** de l'arc (orienté) dans le graphe. Un CSP est rendu AC en répétant les révisions sur tous les arcs.

Puisqu'on travaille souvent avec des CSP binaires, il est important d'assurer la consistance des contraintes binaires. Dans le graphe, une contrainte binaire correspond à un arc. C'est pour cette raison que ce type de consistance est appelé consistance d'arc. L'arc (V_i, V_j) est consistant si pour chaque valeur x appartenant au domaine courant de V_i , il existe une valeur y dans le domaine de V_j de sorte que $V_i = x$ et $V_j = y$ soit autorisé par la contrainte binaire entre V_i et V_j. Il est important de noter le caractère directionnel de la consistante d'arc c'est-à-dire, si l'arc (V_i, V_j) est consistant, cela n'implique par automatiquement que (V_j, V_i) soit aussi consistant. En clair, un arc (V_i, V_j) peut être rendu consistant en supprimant simplement les valeurs du domaine de V_i pour lesquelles il n'existe pas de valeur correspondante dans le domaine de D_j de sorte que la contrainte binaire entre V_i et V_j soit satisfaite. Il est important de noter qu'en supprimant ces valeurs on n'élimine aucune solution du CSP original. L'algorithme REVISE (voir algorithme E.2) réalise ce traitement.

E.2.1 Algorithme REVISE

L'algorithme REVISE prend en arguments 2 variables (V_1 et V_2) qui ont une relation entre elles (exemple : $V_1 + V_2 < 0$). Il agit donc sur un arc. Puis il se fixe sur la deuxième variable (V_2). Il parcourt le domaine (D_1) de la première variable à la recherche des valeurs qui ne respectent pas la dite relation. S'il en trouve, il les élimine du domaine. Supposons qu'il trouve les valeurs $10, 15$ et 25 et que le domaine initial $D_1 = \mathbb{R}$, alors on aura $D_1 = \mathbb{R} - \{10, 15, 25\}$. Ainsi cet algorithme supprime uniquement les valeurs du domaine de la première variable. C'est donc un algorithme directionnel. Le terme REVISE suppose une correction d'un do-

Algorithm E.3 AC-1

Function : AC-1()

1: $Q \leftarrow \{(V_i, V_j) | (V_i, V_j) \in arcs(G), i \neq j\}$
2: **repeat**
3: CHANGE ←FAUX
4: **for each** (V_i, V_j) dans Q **do**
5: CHANGE ← REVISE(V_i, V_j) OR CHANGE ;
6: **end for**
7: **until** not(CHANGE)

maine donné en éliminant certaines valeurs qui y sont initialement présentes et qui deviennent inutiles et incohérentes pour des raisons d'intégrité vis-à-vis des relations auxquelles participent les variables censées prendre ces valeurs.

E.2.2 Algorithme AC-1

AC-1 est le plus simple des algorithmes de consistance d'arc et il répète les révisions jusqu'à ce que le domaine de chaque variable ne soit plus modifiable.

Pour faire en sorte que chaque arc du graphe soit consistant, il ne suffit pas d'exécuter REVISE sur chaque arc juste une seule fois. REVISE réduit le domaine d'une variable par rapport à une autre variable en fonction des relations existant entre ces deux variables.

Il faut donc répéter l'appel à cet algorithme REVISE dans l'autre sens, pour corriger le domaine de la seconde variable, vu que REVISE agit de façon directionnelle. Ensuite, il faut lui faire appel plusieurs fois puisque certaines valeurs du domaine de V_1 peuvent ne plus être compatibles avec toutes les valeurs restantes dans le domaine de V_2. C'est ce que fait l'algorithme AC-1.

Chaque fois que le domaine d'une variable a changé (suite à l'appel de REVISE sur un arc impliquant la variable de ce domaine), on refait appel à REVISE sur tous les autres arcs. Ce qui n'est pas efficace puisque le changement d'un domaine n'affecte pas forcément les domaines des autres variables.

Exemple :
$X \in [1, .., 6]$, $Y \in [1, .., 6]$, $Z \in [1, .., 6]$ et
$X < Y$,
$Z < X - 2$

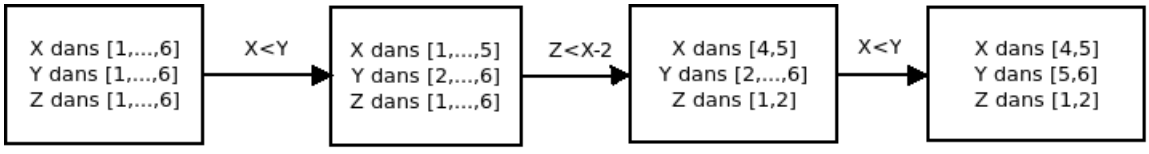


FIGURE E.4 – Exemple pour AC-1

Dans la figure E.4, on peut noter l'application de la contrainte $X < Y$ puis de $Z < X - 2$ et finalement de $X < Y$.

Algorithm E.4 AC-3

Function : AC-3()

1: $Q \leftarrow \{(V_i, V_j) | (V_i, V_j) \in arcs(G), i \neq j\}$
2: **while** non(Q) vide **do**
3: Sélectionner et supprimer un arc (V_k, V_m) de Q ;
4: **if** REVISE (V_k, V_m) **then**
5: $Q \leftarrow Q \cup \{(V_i,V_k) \mid (V_i, V_k) \in arcs(G), i \neq k, i \neq m\}$
6: **end if**
7: **end while**

E.2.3 Algorithme AC-2

AC-2 est une version généralisée de l'algorithme d'étiquetage de Waltz (section D). Il étend le graphe de façon incrémentielle en ajoutant des arcs à chaque étape. A chaque étape, les arcs reliés à une variable donnée sont traités. Ceci est fait afin que si jamais le domaine de la variable en question a subi une modification, alors tous les arcs (et seulement ces arcs) qui lui sont reliés seront revisités en vue de corriger les domaines des autres variables impliquées par ces arcs. Ceci évite de faire appel à REVISE sur tous les arcs non impliqués.

Cependant, il arrive que la réduction du domaine d'une variable n'influence pas forcément tous les arcs associés. AC-2 crée 2 listes, une liste contenant les arcs pour la révision de base et une autre contenant les arcs à re-réviser. On peut se passer de la première liste. C'est ce que propose AC-3.

E.2.4 Algorithme AC-3

AC-3 est une variante des algorithmes de consistance d'arc qui élimine les faiblesses de AC-1 et AC-2 et réalise la re-révision uniquement pour les arcs qui sont affectés par une précédente révision.

Pendant que AC-3 révise l'arc pour la 2ème fois, il re-teste plusieurs couples de valeurs qui sont déjà connus (aux itérations précédentes) pour être soit consistant ou inconsistant et qui ne sont pas affectés pas la réduction du domaine.

Vu que ceci est une source d'inefficacité potentielle, l'algorithme AC-4 a été introduit afin d'affiner le traitement des arcs. AC-3 est le plus utilisé mais il n'est pas très optimal. On teste la consistance de plusieurs couples de valeurs dans chaque révision d'arc. Ces tests sont répétés chaque fois que l'arc est révisé.

Sur la figure E.5, lorsque l'arc (V_2, V_1) est révisé, la valeur "a" est supprimée du domaine V_2. Avec AC-3, le domaine de V_3, devrait être exploré pour trouver si une quelconque valeur "a","b","c","d" a perdu son support [64] dans V_2. On peut observer ceci : les valeurs "a","b" et "c" n'ont pas besoin d'être vérifiées encore une fois puisqu'elles ont toujours un support dans V_2 autre que "a". Une question se pose alors : "Peut-on calculer en une seule fois les ensembles des supports et les utiliser pour les re-révision ?" Oui ; c'est ce qu'effectue AC-4.

64. Le support d'une valeur val_1 du domaine D_1 est toute valeur val_2 du domaine D_2 tel que le couple (val_1, val_2) soit cohérent vis-à-vis de la relation existant entre les variables V_1 et V_2.

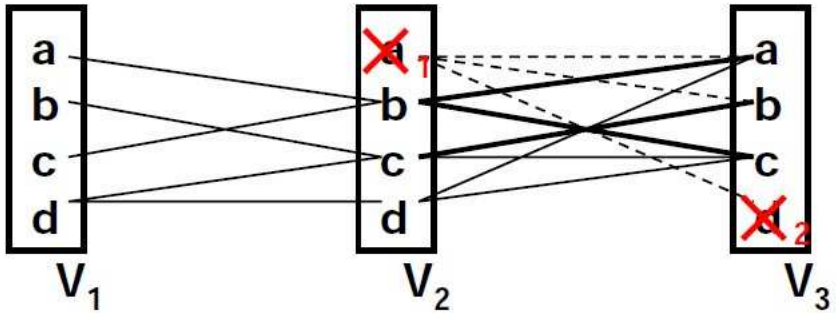


FIGURE E.5 – Exemple pour AC-3

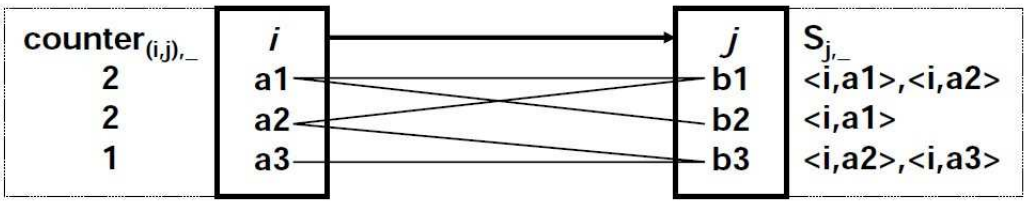


FIGURE E.6 – Exemple AC-4

E.2.5 Algorithme AC-4

Répondre à la question précédente conduit à créer l'algorithme INITIALIZE qui sera utilisé dans AC-4. Pour une valeur donnée, on va conserver l'ensemble des valeurs qu'elle supporte et le nombre de ses supports. Une fois les ensembles des supports calculés avec INITIALIZE, on va les utiliser.

Utilisation des ensembles des supports :

1. Supposons que b_3 soit supprimé du domaine de j (pour certaine raison)
2. On cherche dans S_j, b_3 les valeurs qui étaient supportées par b_3 (i.e.$<i, a_2>$, $<i, a_3>$) comme le montre la figure E.6. Notons que S_j, b_3 représente l'ensemble des supports de $b3$ en sachant que $b3$ appartient au domaine de j.
3. On décrémente le compteur pour ces valeurs (on leur signale qu'ils ont perdu un de leur support)
4. Si un compteur devient nul (a_3, par exemple), alors on supprime la valeur et on répète la procédure avec la valeur suivante (exemple : aller à 1) comme le montre la figure E.7.

AC-4 fonctionne avec des couples de valeurs individuels comme le montre

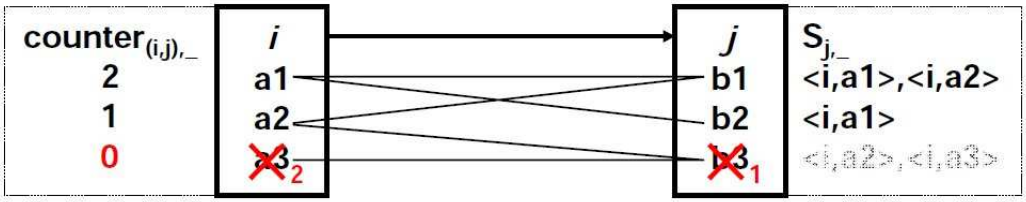


FIGURE E.7 – Exemple AC-4 (deuxième partie)

Algorithm E.5 INITIALIZE

Function : INITIALIZE()

```
1: Q ← {} ;
2: S ← {} ;                          ▷ Initialiser chaque élément de la structure S
3: for each (V_i, V_j) dans arcs(G) do   ▷ (V_i, V_j) et (V_j, V_i) sont les mêmes éléments
4:     for each a dans D_i do
5:         for each b dans D_j do
6:             Total ← 0
7:             if (a, b) est consistant par rapport à la contrainte (V_i, V_j) then
8:                 Total ← Total + 1;
9:                 Sj, b ← Sj, b ∪ {< i, a >} ;
10:            end if
11:        end for
12:        counter[(i, j), a] ← Total ;
13:        if counter[(i, j), a] = 0 then
14:            Supprimer a de D_i ;
15:            Q ← Q ∪ {< i, a >} ;
16:        end if
17:    end for
18: end for
19: return Q
```

Algorithm E.6 AC-4

Function : AC-4()

```
1: Q ← INITIALIZE ;
2: while not(Q) vide do
3:     Sélectionner et supprimer un arc (j, b) de Q ;
4:     for each < i, a > dans Sj, b do
5:         counter[(i, j), a] ← counter[(i, j), a] − 1 ;
6:         if counter[(i, j), a] = 0 & a est toujours dans D_i then
7:             Supprimer a de D_i
8:             Q ← Q ∪{< i, a >}
9:         end if
10:    end for
11: end while
```

l'exemple suivant. Après l'initialisation, l'algorithme AC-4 réalise donc la re-révision pour seulement les couples de valeurs des variables secondaire qui sont affectées par une précédente révision.

En d'autres termes, premièrement, AC-4 initialise ses structures internes qui sont utilisées pour mémoriser les couples de valeurs consistantes (ou inconsistantes) des variables secondaires/auxiliaires ("incidental variables" en anglais) (nœud) - structure $S_{i,a}$. Cette initialisation compte les valeurs de support dans le domaine de la variable secondaire (structure $\text{counter}_{(i,j),a}$) et il élimine les

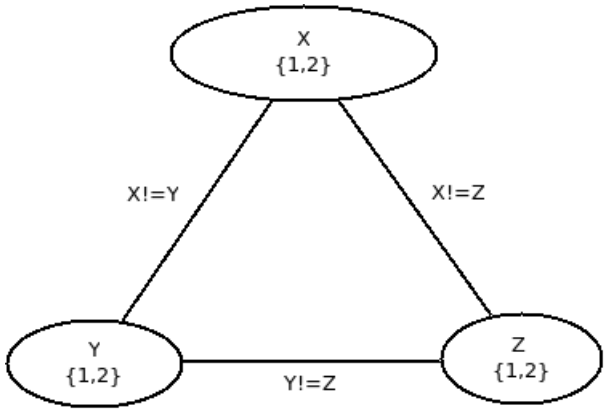


FIGURE E.8 – Consistance d'arc

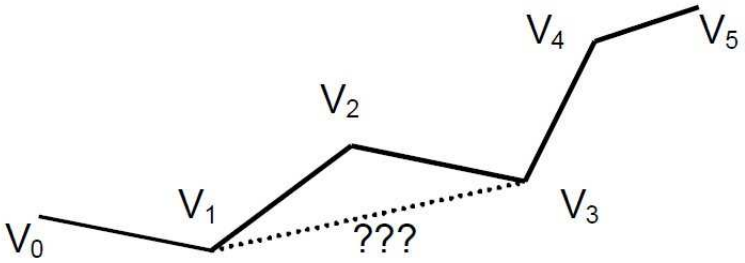


FIGURE E.9 – Consistance de chemin testée le long du chemin de V_0 à V_5

valeurs qui n'ont pas de support. Une fois que la valeur est éliminée du domaine, l'algorithme ajoute le couple $<Variable, Value>$ à la liste Q pour re-révision des valeurs affectées des variables correspondantes.

AC-3 et AC-4 sont très utilisés pour maintenir la consistance d'arc. Néanmoins, il existe d'autres algorithmes d'arc consistance : AC-5 (combine AC-4 et AC-3), AC-6 (améliore AC-4 en gardant en mémoire juste un support), AC-7 (améliore AC-6 en exploitant la symétrie de la contrainte) qui ne sont pas aussi fréquemment utilisés.

Dans la figure E.8, le graphe est arc consistant. Mais il n'y a pas de solution qui puisse satisfaire toutes les contraintes.

E.3 La consistance de chemin

Par définition, le chemin $(V_0, V_1, ..., V_m)$ est chemin-consistant si et seulement si pour chaque couple de valeurs $x \in D_0$ à $y \in D_m$ satisfaisant toutes les contraintes binaires sur V_0 à V_m, il existe une affectation de variables V_1, ...,V_{m-1} telle que toutes les contraintes binaires entre les variables voisines soient satisfaites.

La figure E.9 montre un exemple de recherche consistance de chemin. Montanari (Montanari, 1974) a montré qu'un CSP est chemin consistant si et seulement

si tous les chemins de longueur 2 sont chemin-consistants. Les algorithmes de consistance de chemin fonctionnent sur des triplets de variables (pour avoir un chemin de longueur 2). Il existe plusieurs algorithmes de consistance de chemin comme PC-1 et PC-2.

Annexe F

Un exemple de DICSP

Nous présentons dans cette annexe un exemple de DICSP que nous avons utilisé.

Constraints
xk0 = xk_10 + dsk0*cos(tk_10 + dtk0/2),
yk0 = yk_10 + dsk0*sin(tk_10 + dtk0/2),
tk0 = tk_10 + dtk0,
xk1 = xk0 + dsk1*cos(tk0 + dtk1/2),
yk1 = yk0 + dsk1*sin(tk0 + dtk1/2),
tk1 = tk0 + dtk1,
xk2 = xk1 + dsk2*cos(tk1 + dtk2/2),
yk2 = yk1 + dsk2*sin(tk1 + dtk2/2),
tk2 = tk1 + dtk2,
xk3 = xk2 + dsk3*cos(tk2 + dtk3/2),
yk3 = yk2 + dsk3*sin(tk2 + dtk3/2),
tk3 = tk2 + dtk3,
xk4 = xk3 + dsk4*cos(tk3 + dtk4/2),
yk4 = yk3 + dsk4*sin(tk3 + dtk4/2),
tk4 = tk3 + dtk4,
xk5 = xk4 + dsk5*cos(tk4 + dtk5/2),
yk5 = yk4 + dsk5*sin(tk4 + dtk5/2),
tk5 = tk4 + dtk5,
xk6 = xk5 + dsk6*cos(tk5 + dtk6/2),
yk6 = yk5 + dsk6*sin(tk5 + dtk6/2),
tk6 = tk5 + dtk6,
xk7 = xk6 + dsk7*cos(tk6 + dtk7/2),
yk7 = yk6 + dsk7*sin(tk6 + dtk7/2),
tk7 = tk6 + dtk7,
xk8 = xk7 + dsk8*cos(tk7 + dtk8/2),
yk8 = yk7 + dsk8*sin(tk7 + dtk8/2),
tk8 = tk7 + dtk8 ;

Variables
dsk0 in [1.313186, 1.715310],
dtk0 in [-0.013000, -0.012000],
xk0 in [519.827976, 533.040033],
yk0 in [-144.433148, -117.452743],
tk0 in [2.845714, 3.021247],
xk_10 in [521.086562, 534.744141],
yk_10 in [-144.922623, -117.602573],
tk_10 in [2.858714, 2.945981],
dsk1 in [0.298451, 0.593761],
dtk1 in [-0.004000, -0.003000],
xk1 in [519.436256, 532.648313],
yk1 in [-144.372604, -117.392199],
tk1 in [2.841714, 3.018247],
dsk2 in [0.358142, 0.659734],
dtk2 in [-0.004000, -0.003000],
xk2 in [518.781657, 532.306366],
yk2 in [-144.328008, -117.196051],
tk2 in [2.837714, 3.015247],
dsk3 in [0.358142, 0.659734],
dtk3 in [-0.004000, -0.003000],
xk3 in [518.259673, 531.801844],
yk3 in [-144.219347, -117.108617],
tk3 in [2.833714, 3.012247],
dsk4 in [0.477522, 0.791681],
dtk4 in [-0.003000, -0.002000],
xk4 in [517.804822, 531.016879],
yk4 in [-144.017169, -117.036764],
tk4 in [2.830714, 3.010247],
dsk5 in [0.477522, 0.791681],
dtk5 in [-0.002000, -0.001000],
xk5 in [517.222263, 530.434320],
yk5 in [-143.887448, -116.907043],
tk5 in [2.828714, 3.009247],
dsk6 in [0.477522, 0.791681],
dtk6 in [-0.003000, -0.002000],
xk6 in [516.573640, 529.785697],
yk6 in [-143.735279, -116.754874],
tk6 in [2.825714, 3.007247],
dsk7 in [0.596903, 0.923628],
dtk7 in [-0.002000, -0.001000],
xk7 in [515.658397, 529.218512],
yk7 in [-143.655033, -116.467070],
tk7 in [2.823714, 3.006247],
dsk8 in [0.596903, 0.923628],
dtk8 in [-0.002000, -0.001000],
xk8 in [515.142972, 528.355028],
yk8 in [-143.408277, -116.427872],
tk8 in [-1000.000000, 1000.000000] ;

Ce DICSP représente l'état du système à l'instant 8. On observe dans le premier encadré, l'ensemble des contraintes et dans le second encadré, l'ensemble de variables du problème. A chaque variable est associé un domaine réprésenté par un intervalle. dsk8, dtk8, xk8, yk8 et tk8 sont les dernières variables ajoutées au problème à l'instant courant.

Annexe G

L'algorithme HC3

Nous présentons, dans cette annexe[65], l'algorithme HC3 qui a été plusieurs fois cité dans cette thèse. La consistance d'enveloppe ou 2B-Consistance est une approximation de la consistance d'arc appliquée aux domaines continus. Elle est implémentée dans deux algorithmes connus sous le nom de HC3 (Cleary, 1987) et HC4 (Benhamou et al., 1999). HC3 maintient la consistance d'enveloppe sur des contraintes primitives.

Soit P=(X, D, C), un ICSP et $c \in C$ une contrainte n-aire sur les variables (x_1, ... , x_n). La contrainte c est enveloppe-consistante si et seulement si :

$\forall i$, $D_{x_i} = Hull_{\square}\{v_i \in D_{x_i} | \exists v_1 \in D_{x_1}, ..., \exists v_{i-1} \in D_{x_{i-1}}, \exists v_{i+1} \in D_{x_{i+1}}, ..., \exists v_n \in D_{x_n}$ tel que $c(v_1, ..., v_{i-1}, x_i, v_{i+1}, ..., v_n)\}$

Algorithm G.1 HC3

Function : HC3(C, D)

1: $Q \leftarrow \{< x_i, c_j > | c_j \in C$ ET $x_i \in Var(C_j)\}$
2: **while** ($Q \neq \emptyset$) **do**
3: Extraire $< x_i, c_j >$ dans Q
4: $D' \leftarrow$narrowing(D, x_i, c_j)
5: **if** ($D' \neq D$) **then**
6: $D \leftarrow D'$
7: $Q \leftarrow Q \cup \{< x_l, C_k > | C_k \neq C_j \wedge (x_l, x_i) \in Var(C_k)\}$
8: **end if**
9: **end while**
10: **return** D

L'algorithme G.1 s'appelait IN-1[66](Cleary, 1987). Il réalise une approximation des fonctions de projection. Cependant, la projection d'une contrainte sur une variable n'est généralement pas un intervalle mais une union d'intervalles. Il faut donc réunir les intervalles avec une fonction enveloppe[67] intervalle $Hull_{\square}$.

$AP_i(c, I_1 \times \cdots \times I_k)$ est une approximation minimale de la fonction de projection $\pi_i(c, I_1 \times \cdots \times I_k)$ si et seulement si :

65. Cette annexe est, en partie, inspirée de (Rueher, 2005)
66. Interval Narrowing
67. D'où le nom "consistance d'enveloppe"

Algorithm G.2 narrowing

Function : narrowing(D, x_i, C_j)
1: $m \leftarrow Min_{x_i}(C, D_{x_i})$
2: $M \leftarrow Max_{x_i}(C, D_{x_i})$
3: **return** $D[D_{x_i} \leftarrow [m, M]]$

$AP_i(c,\ I_1 \times \cdots \times I_k) = Hull_{\square}(\pi_i(c,\ I_1 \times ... \times I_k))$

$AP_i(c,\ I_1 \times \cdots \times I_k) = [Min(\pi_i(c,\ I_1 \times ... \times I_k)),\ Max(\pi_i(c,\ I_1 \times ... \times I_k))]$

$AP_i(c, I_1 \times \cdots \times I_k)$ étant le plus petit intervalle contenant la projection $\pi_i(c,\ I_1 \times ... \times I_k)$.

On peut alors affirmer que :

- une contrainte c est 2B–Consistante sur $< I_1, ..., I_k >$ si pour tout i dans $\{1, ..., k\}$, $I_i = AP_i(c, I_1 \times \cdots \times I_k)$.

Min_{x_i}et Max_{x_i} sont les fonctions extremum dans la fonction "narrowing".

On calcule simplement et facilement AP_i pour les fonctions monotones : il s'agit d'évaluer la fonction de projection pour les bornes de chaque domaine D_i.

Cependant, dans le cas général, il est difficile de définir les fonctions Min et Max. Par conséquent, AP_i ne peut pas être calculée directement, dans le cas général. Il faut donc décomposer les contraintes en fonctions primitives pour lesquelles, on peut facilement identifier les parties monotones.

G.1 Exemple de filtrage par HC3.

Considérons le CSP suivant :

$$\begin{cases} D_x = [-4, 5], \\ D_y = [3, +\infty] \\ D_z = [1, 2] \\ c_1 :\ x + y = 1 \\ c_2 :\ y = 2 \times z \end{cases}$$

On déduit la file des couples <contrainte,variable> à traiter comme suit :

$Qu = \{a :< c_1, x >, b :< c_1, y >, c :< c_2, y >, d :< c_2, z >\}$

– Sélection de a, $D_x \leftarrow [-4, 2]$
  Aucun élément n'est ajouté dans Qu
– Sélection de b, $D_y \leftarrow [3, 5]$
  Aucun élément n'est ajouté dans Qu
– Sélection de c, $D_y \leftarrow [3, 4]$
  $Qu = \{d :< c_2, z >,\ a :< c_1, x >\}$
– Sélection de d, $D_y \leftarrow [1.5, 2]$
  Aucun élément n'est ajouté dans Qu
– Sélection de a, $D_x \leftarrow [-3, -2]$

Aucun élément n'est ajouté dans Qu et $Qu = \emptyset$

G.2 Évaluation des fonctions de projection

Soit le CSP :
$$\begin{cases} x = y + z^2 \\ D_x = [0, 6], \\ D_y = [5, 9], \\ D_z = [-2, 4]. \end{cases}$$
La fonction de projection pour y est $D'_y \leftarrow (D_x - D_z^2) \cap D_y$. D'ou : $D'_y = ([0, 6]-[0, 16])\cap[5, 9] = [5, 6]$. Pour le calculer D_z^2 on utilise la fonction puissance [68].

Soit $x = y^2$ et $D'_y \leftarrow \sqrt{D_x} \cap D_y$

L'évaluation de la fonction de projection demande dans ce cas une analyse des parties monotones : si $D_x = [4, 9]$ et $D_y = [-1, 5]$ alors D'_y vaut $[2, 3]$ et non $[-1, 3]$ car aucune des valeurs entre -1 et $\sqrt{D_x}$ n'a de support dans D_x. Cependant, si $D_x = [4, 9]$ et $D_y = [-2, 5]$ alors D'_y =$[-2, 3]$ car -2 a pour support $-\sqrt{4}$.

G.3 Décomposition des contraintes

Pour ce qui est de la décomposition, il est utile de noter que la décomposition d'un système de contraintes C en un système de contraintes primitives nommé $decomp(C)$ ne change pas la sémantique du problème. C et $decomp(C)$ ont les mêmes solutions [69]. La décomposition réduit la portée des vérifications effectuées par la consistance d'enveloppe. Si une variable x a plusieurs occurrences dans une contrainte $c \in C$, alors les différentes occurrences de x dans $decomp(c)$ pourront être contractées indépendamment les unes des autres. Par conséquent, le filtrage par la consistance d'enveloppe de $decomp(C)$ sera plus faible que celui de C. Prenons un exemple. Considérons le CSP suivant :
$$\begin{cases} D_{x_1} = [-1, 1] \\ D_{x_2} = [0, 1] \\ c : x_1 + x_2 - x_1 = 0 \end{cases}$$
Après décomposition, on aura :
$$decomp(c) = \begin{cases} x_1 + x_2 - x_3 = 0 \\ x_1 = x_3 \end{cases}$$
Chaque fonction de projection de $decomp(c)$ peut être évaluée à l'aide des opérations du calcul d'intervalle, par exemple, $AP_1(x_1 + x_2 - x_3 = 0, D_{x_1}, D_{x_2}, D_{x_3})$ est $D_{x_1} \cap (D_{x_3} \Theta D_{x_2})$. La contrainte c n'est pas enveloppe-consistante parce qu'il n'y a pas de support pour $x_2 = 1$. Par contre, $decomp(c)$ est enveloppe-consistante étant donné que les valeurs $x_1 = -1$ et $x_3 = 0$ satisfont la contrainte $x_1 + x_2 - x_3 = 0$, lorsque $x_2 = 1$.

68. On utilise l'extension aux intervalles de la fonction puissance

69. On attend des deux systèmes les mêmes solutions. Cela ne veut pas dire qu'on obtient forcément les mêmes solutions

Bibliographie

Ackermann, J. and Bünte, T. (1997). Yaw disturbance attenuation by robust decoupling of car steering. Control Engineering Practice, 5(8) :1131–1136. 5.5.2.2

Alander, J. (1985). On interval arithmetic range approximation methods of polynomials and rational functions. Computers & Graphics, 9(4) :365–372. 3.4, 5.7.1

Aldon, M.-J. (2001). Capteurs et méthodes pour la localisation des robots mobiles. Techniques de l'ingénieur. Informatique industrielle, 6(S7852) :1–19. (document), 5.3.2, 5.1

Alefeld, G. and Herzberger, J. (1983). Introduction to interval computations. 4.5.3.1

Andresen, S. L. (2001). Herbert a. simon : Ai pioneer. Intelligent Systems, IEEE, 16(4) :71–72. 2.1

Archimedes (1897). On the measurement of the circle. Thomas L. Heath (ed.), The Works of Archimedes, Cambridge University Press, Cambridge, pages 91–98. 3.1.1

Baumann, E. (1988). Optimal centered forms. BIT Numerical Mathematics, 28(1) :80–87. 5.7.1

Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J. (1999). Revising hull and box consistency. In International Conference on Logic Programming, pages 230–244. 4.1, 4.5.2, 4.5.2, 4.5.2.1, 4.5.3.2, 5.6.2.2, 5.6.2.3, G

Benhamou, F., McAllester, D., and Van Hentenryck, P. (1994). Clp (intervals) revisited. In International Symposium on Logic programming, pages 124–138. 4.1, 4.5.2, 4.5.3, 4.5.3.1, 4.5.3.1, 5.6.2.2

Benhamou, F. and Older, W. (1997). Applying interval arithmetic to real, integer, and boolean constraints. The Journal of Logic Programming, 32(1) :1–24. 4.1

Berger, N., Granvilliers, L., et al. (2009). Some interval approximation techniques for minlp. In Eighth Symposium on Abstraction, Reformulation, and Approximation. 4.6.7

Berger, N., Soto, R., Goldsztejn, A., Caro, S., and Cardou, P. (2010). Finding the maximal pose error in robotic mechanical systems using constraint programming. Trends in Applied Intelligent Systems, pages 82–91. 4.6.7

Bessiere, C. (1994). Arc-consistency and arc-consistency again. Artificial intelligence, 65(1) :179–190. 4.1

Bishop, R. (2000a). Intelligent vehicle applications worldwide. Intelligent Systems and their Applications, IEEE, 15(1) :78–81. 2.1

Bishop, R. (2000b). A survey of intelligent vehicle applications worldwide. In Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE, pages 25–30. IEEE. 2.1

Bradis, V. M. (1927). An experience of the verification of practically useful operations with approximate numbers. Proceedings of Tver Pedagogical Institute, (3). 3.1.1

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. Artificial intelligence, 114(1) :3–55. 5.5.1.2

Chabert, G. (2007). Ibex, an interval-based explorer. 4.6.6

Chabert, G. and Jaulin, L. (2009). Hull consistency under monotonicity. International Conference on Principles and Practice of Constraint Programming, pages 188–195. 5.6.2.3

Cleary, J. (1987). Logical arithmetic. Future computing systems, 2(2) :125–149. 4.1, 4.5.2, G, G

Clowes, M. (1971). On seeing things. Artificial intelligence, 2(1) :79–116. 4.1

Collavizza, H., Delobel, F., and Rueher, M. (1998). A note on partial consistencies over continuous domains. Principles and Practice of Constraint Programming, pages 147–161. 4.5.2

Collavizza, H., Delobel, F., and Rueher, M. (1999). Comparing partial consistencies. Reliable computing, 5(3) :213–228. 4.5.1

Comba, J. L. D. and Stol, J. (1993). A ne arithmetic and its applications to computer graphics. In Proceedings of VI SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing), pages 9–18. Citeseer. 5.7.1

Davis, E. (1987). Constraint propagation with interval labels. Artificial intelligence, 32(3) :281–331. 4.1

Dechter, R. and Dechter, A. (1988). Belief maintenance in dynamic constraint networks. AAAI, Conference on Artificial Intelligence, pp 37-42. 4.4, 5.6.3.2

Dechter, R. and Pearl, J. (1989). Tree clustering for constraint networks. Artificial Intelligence, 38(3) :353–366. 4.1, 5.6.2.1

Delafosse, M., Clerentin, A., Delahoche, L., Brassart, E., and Marhic, B. (2004). Multi sensor fusion and constraint propagation to localize a mobile robot. In Industrial Technology, 2004. IEEE ICIT'04. 2004 IEEE International Conference on, volume 1, pages 66–71. IEEE. 5.5.2.2

Domes, F. (2009). Gloptlab : a configurable framework for the rigorous global solution of quadratic constraint satisfaction problems. Optimization Methods & Software, 24(4-5) :727–747. 4.6.5

Domes, F. and Neumaier, A. (2007). Verified global optimization with gloptlab. PAMM, 7(1) :1020101–1020102. 4.6.5

Dwyer, P. S. (1951). Linear computations. J. Wiley. 3.1.1

Everett, H. (1995). Sensors for mobile robots : theory and application. AK Peters, Ltd. 2.2

Figueiredo, L. H. and Stolfi, J. (1996). Adaptive enumeration of implicit surfaces with affine arithmetic. In Computer Graphics Forum, volume 15, pages 287–296. Wiley Online Library. 5.7.1

Filliat, D. and Meyer, J.-A. (2003). Map-based navigation in mobile robots : : I. a review of localization strategies. Cognitive Systems Research, 4(4) :243–282. 5.1

Filliat, D., Meyer, J.-A., et al. (2002). Global localization and topological map-learning for robot navigation. From Animals to Animats, 7 :131–140. 5.2.2

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization : Efficient position estimation for mobile robots. AAAI/IAAI, 1999 :343–349. 5.5.1.3

Fu, Y., Tully, S., Kantor, G., and Choset, H. (2011). Monte carlo localization using 3d texture maps. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pages 482–487. IEEE. 5.5.1.3

Gallaire, H. (1985). Logic programming : Further developments. In IEEE Symposium on Logic Programming, pages 88–99. 4.1

Gning, A. and Bonnifait, P. (2004). Guaranteed dynamic localization using constraints propagation techniques on real intervals. In IEEE International Conference on Robotics and Automation, pages 1499–1504. 4.1, 5.5.2.2

Granvilliers, L. (2000). Towards cooperative interval narrowing. Frontiers of Combining Systems, pages 18–31. 4.1, 5.6.2.3

Granvilliers, L. and Benhamou, F. (2006). Algorithm 852 : Realpaver : an interval solver using constraint satisfaction techniques. ACM Transactions on Mathematical Software, 32(1) :138–156. 4.6.2, 5.6.4

Gruyer, D., Belaroussi, R., Vigneron, V., and Cord, A. (2014). How to manage conflict and ambiguities in localization and map matching. Journal of Intelligent Systems, 23(2) :171–182. 5.1

Hansen, E. (1992). Global optimization using interval analysis. Marcel Dekkar, New York. 3.6.1

Hansen, E. and Walster, G. W. (2003). Global optimization using interval analysis : revised and expanded, volume 264. CRC Press. 3.6.1

Hvilshj, M., Bgh, S., Madsen, O., and Kristiansen, M. (2009). The mobile robot 'little helper' : concepts, ideas and working principles. In Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on, pages 1–4. IEEE. 5.3.1

Hvilshøj, M. and Bøgh, S. (2011). 'little helper' - an autonomous industrial mobile manipulator concept. International Journal of Advanced Robotic Systems, 8(2). 5.3.1

Hyvönen, E. (1989). Constraint reasoning based on interval arithmetic. In International Joint Conference on Artificial Intelligence, pages 1193–1198. 4.1

Hyvönen, E. (1992). Constraint reasoning based on interval arithmetic : the tolerance propagation approach. Artificial Intelligence, 58(1) :71–112. 4.3

Hyvönen, E. and De Pascale, S. (1999). A new basis for spreadsheet computing. interval solver (tm) for microsoft excel. In 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99), pages 799–806. 4.3

Hyvönen, E., De Pascale, S., and Lehtola, A. (1993). Interval constraint satisfaction tool inc++. In Fifth International Conference on Tools with Artificial Intelligence, TAI'93, pages 298–305. IEEE. 4.3

Ivanjko, E. and Petrovic, I. (2004). Extended kalman filter based mobile robot pose tracking using occupancy grid maps. In Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean, volume 1, pages 311–314. IEEE. 5.5.1.1

Jaffar, J. and Lassez, J. (1987). Constraint logic programming. In ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pages 111–119. 4.1

Jaulin, L. (2001). Applied interval analysis : with examples in parameter and state estimation, robust control and robotics. Springer. 5.5.2.1

Jaulin, L. (2006). Localization of an underwater robot using interval constraint propagation. International Conference on Principles and Practice of Constraint Programming, pages 244–255. 4.1, 5.5.2.2

Jaulin, L. (2009). Quimper, a language for quick interval modelling and programming in a bounded-error context. artificial intelligence. Artificial Intelligence. 4.6.6

Jaulin, L. and Walter, E. (1993). Set inversion via interval analysis for nonlinear bounded-error estimation. Automatica, 29(4) :1053–1064. 5.5.2.1, A, A, B

Jensfelt, P. and Christensen, H. I. (2001). Pose tracking using laser scanning and minimalistic environmental models. Robotics and Automation, IEEE Transactions on, 17(2) :138–147. 5.5.1.1

Kalman, R. (1960). A new approach to linear filtering and prediction problems. Basic Engineering, 82 :34–45. C.2

Kearfott, R. (1987). Some tests of generalized bisection. ACM Transactions on Mathematical Software (TOMS), 13(3) :197–220. B

Kearfott, R. B. (1996a). Rigorous global search continuous problems. Springer. 3.6.2

Kearfott, R. B. (1996b). Test results for an interval branch and bound algorithm for equality-constrained optimization. In State of the art in global optimization, pages 181–199. Springer. B

Keil, C. (2007). Verified linear programming–a comparison. Reliable Computing, submitted. 4.6.7

Keil, C. (2008). A comparison of software packages for verified linear programming. Preprint Institute of Reliable Computing, Hamburg University of Technology. 4.6.7

Kumar, V. (1992). Algorithms for constraint-satisfaction problems : A survey. AI magazine, 13(1) :32. E

Lambert, A., Gruyer, D., Vincke, B., and Seignez, E. (2009). Consistent outdoor vehicle localization by bounded-error state estimation. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1211–1216. (document), 2.3, 2.6, 5.6.4.4, 5.7.3, 5.7.4.2

Lamon, P., Kolski, S., Triebel, R., Siegwart, R., and Burgard, W. (2006). The smartter for elrob 2006-a vehicle for fully autonomous navigation and mapping in outdoor environments. Autonomous Systems Lab, Ecole polytechnique Fdrale de Lausanne, Switzerland, Autonomous Intelligent Systems, Albert-Ludwigs-University of Freiburg, Germany, Tech. Rep. 2.1, 2.1

Lawler, E. and Wood, D. (1966). Branch-and-bound methods : A survey. Operations research, 14(4) :699–719. B

Lebbah, Y. (2009). Icos : a branch and bound based solver for rigorous global optimization. Optimization Methods & Software, 24(4-5) :709–726. 4.6.4

Leonard, J. J. and Durrant-Whyte, H. F. (1991). Mobile robot localization by tracking geometric beacons. Robotics and Automation, IEEE Transactions on, 7(3) :376–382. 5.3.2

Lhomme, O. (1993). Consistency techniques for numeric csps. In International Joint Conference on Artificial Intelligence. 4.1, 4.5.2, 4.5.4, 4.5.4.1, 5.6.2.2, 5.7.1

Lingemann, K., Nüchter, A., Hertzberg, J., and Surmann, H. (2005). High-speed laser localization for mobile robots. Robotics and Autonomous Systems, 51(4) :275–296. 5.5.1.1

Loh, E. and Walster, G. W. (2002). Rump's example revisited. Reliable Computing, 8(3) :245–248. 3.1.2.1

Luc Jaulin, Marc Christie, L. G. (2002). Quelques applications de la propagation de contraintes sur les domaines continus en automatique. JFPLC'2002. 4.6.7

Maaref, H., Oussalah, M., and Barreti, C. (1999). Fusion de données capteurs en vue de la localisation absolue d'un robot mobile par une méthode basée sur la théorie des possibilités. comparaison avec le filtre de kalman. Traitement du Signal, 16(5). 5.5

Mackworth, A. (1977). Consistency in networks of relations. Artificial intelligence, 8(1) :99–118. 4.1

Maybeck, P. (1979). Stochastic models, estimation and control, volume 1. Academic Pr. C.3

Mei, C. and Rives, P. (2007). Cartographie et localisation simultanée avec un capteur de vision. Journées Nationales de la Recherche en Robotique. 2.2.1.9

Merlet, J. (2001). Projet coprin : Contraintes, optimisation, résolution par intervalles. Rapport, INRIA Sophia-Antipolis, 21. 4.6.3

Merlet, J.-P. (2000). Alias : an interval analysis based library for solving and analyzing system of equations. SEA, June. Automation, pages 1964–1969. 4.6.3

Mohr, R., Masini, G., et al. (1988). Good old discrete relaxation. In European Conference on Artificial Intelligence, pages 651–656. 4.1

Montanari, U. (1974). Networks of constraints : Fundamental properties and applications to picture processing. Information sciences, 7 :95–132. E.3

Moore, R. (1962). Interval arithmetic and automatic error analysis in digital computing. Technical report, DTIC Document. 3.1.1

Moore, R. (1966). Interval analysis. Prentice-Hall Englewood Cliffs, NJ. 3.1.1, 5.7.1

Moore, R., Kearfott, R., and Cloud, M. (2009). Introduction to interval analysis. Society for Industrial Mathematics. 3.1.2.2, 3.2.1, 5.7.1, 5.7.1

Moore, R. E. (1959). Automatic error analysis in digital computation. Technical report, Space Div. Report LMSD84821, Lockheed Missiles and Space Co. 3.1.1

Ng, G. W. (2003). Intelligent systems : fusion, tracking and control. Research Studies Press. 2.1

Perrollaz, M. (2008). Détection d'obstacles multi capteurs supervisée par stéréovision. PhD thesis, Université Pierre et Marie Curie. (document), 2.7

Ratz, D. (1998). Automatic slope computation and its application in nonsmooth global optimization. Shaker. 5.7.1

Rodriguez, F., Frémont, V., Bonnifait, P., et al. (2009). An experiment of a 3d real-time robust visual odometry for intelligent vehicles. In Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on, pages 1–6. IEEE. 2.2.1.9

Rohani, M., Gingras, D., Vigneron, V., and Gruyer, D. (2013). A new decentralized bayesian approach for cooperative vehicle localization based on fusion of gps and inter-vehicle distance measurements. In Connected Vehicles and Expo (ICCVE), 2013 International Conference on, pages 473–479. 5.4

Rossi, F., Petrie, C., and Dhar, V. (1990). On the equivalence of constraint satisfaction problems. In European Conference on Artificial Intelligence, pages 550–556. 4.1, 5.6.2.1

Rueher, M. (2005). Solving continuous constraint systems. In 8th International Conference on Computer Graphics and Artificial Intelligence. (document), 4.5, 65

Rump, S. (1999). INTLAB - INTerval LABoratory. pages 77–104. http://www.ti3.tuhh.de/rump/. 3.1.2.1, 3.2.5

Rump, S. M. (1988). Algorithms for verified inclusions theory and practice. Reliability in computing, 19 :109–126. 3.1.2.1

Seignez, E., Kieffer, M., Lambert, A., Walter, E., and Maurin, T. (2005). Experimental vehicle localization by bounded-error state estimation using interval analysis. In Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on, pages 1084–1089. IEEE. 5.5.2.1

Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). Introduction to autonomous mobile robots. MIT press. (document), 5.1

Stol, J. and De Figueiredo, L. H. (1997). Self-validated numerical methods and applications. In Monograph for 21st Brazilian Mathematics Colloquium, IMPA, Rio de Janeiro. Citeseer. 5.7.1

Sunaga, T. (1958). Theory of interval algebra and its application to numerical analysis. Research Association of Applied Geometry (RAAG) Memoirs, Ggujutsu Bunken Fukuy-kai, 2 :29–46. 3.1.1

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2001). Robust monte carlo localization for mobile robots. Artificial intelligence, 128(1) :99–141. 5.5.1.3

Tóth, B. and Csendes, T. (2005). Empirical investigation of the convergence speed of inclusion functions in a global optimization context. Reliable Computing, 11(4) :253–273. 3.4, 5.7.1

Tóth, B., Fernández, J., and Csendes, T. (2007). Empirical convergence speed of inclusion functions for facility location problems. Journal of computational and applied mathematics, 199(2) :384–389. 3.4, 5.7.1, 5.7.1

Van Hentenryck, P., Deville, Y., and Teng, C. (1992). A generic arc-consistency algorithm and its specializations. Artificial Intelligence, 57(2) :291–321. 4.1

Van Hentenryck, P., Michel, L., and Deville, Y. (1997). Numerica : a modeling language for global optimization. MIT press. 4.6.1

Vigneron, V. (2007). Some applications of interval analysis to statistical problems. In Computational and Ambient Intelligence, pages 564–579. Springer. 3.1.1

Vinas, H. P., Sainz, M. A., Vehi Casellas, J., and Jaulin, L. (2004). Quantified set inversion with applications to control. In IEEE International Symposium on Computer Aided Control Systems Design, pages 179–183. 3.1.1

Vincke, B. and Lambert, A. (2009). Enhanced constraints propagation for guaranteed localization predictive step. In IEEE/RSJ IROS, Workshop on Planning, Perception and Navigation for Intelligent Vehicles, pages 30–36. 4.1, 5.5.2.2

Vincke, B. and Lambert, A. (2011). Experimental comparison of bounded-error state estimation and constraints propagation. In IEEE International Conference on Robotics and Automation, pages 4724–4729. 1.2, 5.7.1

Vincke, B., Lambert, A., Gruyer, D., Elouardi, A., and Seignez, E. (2010). Static and dynamic fusion for outdoor vehicle localization. In International Conference on Control Automation Robotics & Vision, pages 437–442. 4.1, 5.5.2.2

Walster, G., Hansen, E., and Sengupta, S. (1985). Test results for a global optimization algorithm. SIAM Numerical Optimization, pages 272–287. B

Waltz, D. (1975). Understanding line drawings of scenes with shadows. Psychology of Computer Vision, McGraw-Hill,New York. (document), 4.1, D, D.1, D.2, D.3

Waltz, D. L. (1972). Generating semantic descriptions from drawings of scenes with shadows. PhD thesis, AI Lab, MIT. 4.1, 5.6.1, D

Warmus, M. (1956). Calculus of approximations. Bulletin de l'Academie Polonaise de Sciences, 4(5) :253–257. 3.1.1

Young, R. C. (1931). The algebra of multi-valued quantities. Mathematische Annalen, 104 :260–290. 3.1.1

Young, W. H. (1908). Sull due funzioni a piu valori constituite dai limiti d'una funzione di variable reale a destra ed a sinistra di ciascun punto. Rendiconti Academia di Lincei, Classes di Scienza Fiziche, 17 :582–587. 3.1.1

Yuanlin, Z. and Yap, R. (2000). Arc consistency on n-ary monotonic and linear constraints. International Conference on Principles and Practice of Constraint Programming, pages 470–483. 4.1