Master Systèmes Dynamiques et Signaux

Mémoire de master

Réalisation d'un contracteur optimal appliqué à une contrainte triangulaire

---

Auteur :
M. Arthur Ignazi

Jury :
Dr. R. Guyonneau
Pr. A. Humeau- Heurtier
Dr. M. Hilal
Dr. S. Lagrange
Dr. M.-L. Pannier

Président :
Pr. L. Hardouin

Version du
13 juillet 2022

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce mémoire.

Merci à Sébastien Lagrange d'avoir accepté d'être mon tuteur pour ce stage de recherche, pour les nombreuses réflexions que nous avons partagées et l'aide précieuse qu'il m'a apportée. Ses conseils et son expertise m'ont permis de continuer à avancer durant tout le stage, et de continuer à être motivé malgré certains échecs. Son accompagnement et sa disponibilité ont été d'une grande aide, et me donne aujourd'hui encore l'envie de poursuivre en doctorat.

Je tiens à remercier Nicolas Delanoue, qui m'a conseillé et appris beaucoup durant ce stage. Ses connaissances ont été un support sur lequel j'ai pu compter. Son regard extérieur au problème et sa culture nous ont permis d'avancer à maintes reprises.

Je remercie aussi Rémy Guyonneau, qui m'a fait découvrir à travers son cours l'analyse par intervalle au premier semestre. J'ai pu commencer mon stage avec de bonnes bases, et sa disponibilité sur le sujet m'a aidé tout au long du stage.

Enfin, merci à mes amis et collègues Clément, Emma, Yann, Clémence et Ahmed. Partager le bureau comme les réflexions m'a permis de maintenir ma motivation et de découvrir d'autres sujets auxquels je n'avais jamais été confronté.

Table des matières

Table des figures

List of acronyms

CSP *Constraint Satisfaction Problem*

SIVIA *Set Inversion Via Interval Analysis*

SDP *Semi-Définie positive*

SOS *Sum Of Square* : Un polynôme pouvant s'écrire comme une somme de carré

DDL *Degré de liberté*

Introduction

L'analyse par intervalle permet de faire des calculs exacts, en prenant en compte les incertitudes de mesure ou de calcul. Dans un contexte de localisation d'un robot, nous allons utiliser l'analyse par intervalle pour gagner en précision tout en garantissant un résultat exact.

Dans notre cas, la position du robot A n'est pas un point mais une boite, qui est la représentation de deux intervalles $[x]$ et $[y]$. Ce robot va, à l'aide de capteurs, détecter deux obstacles, B et C. De même, la position de ces obstacles n'est pas ponctuelle mais située dans des boites. Nous connaissons les distances entre le robot et chaque obstacle, ainsi que la distance entre les deux obstacles. On peut en déduire le triangle résultant, comme le montre la figure 1.

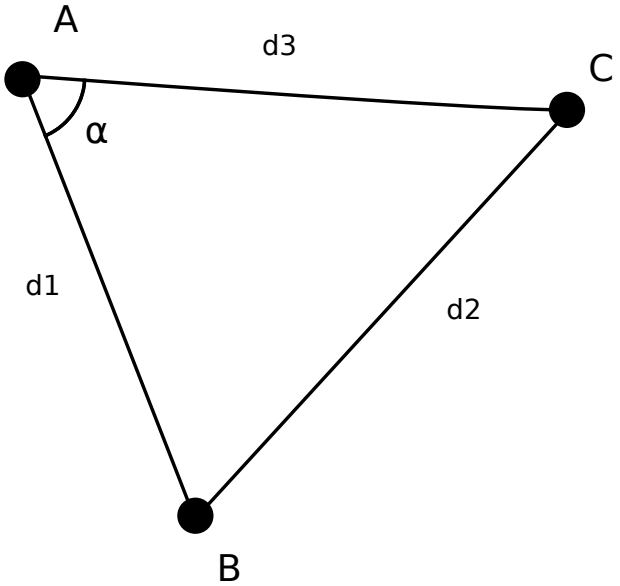

FIGURE 1 – Localisation d'un robot

L'objectif de ce travail de recherche est de trouver un contracteur [1] qui va réduire au maximum la taille des intervalles, sans perdre de valeurs.

1. Contracteur : 1.3

Nous allons commencer par faire un état de l'art et expliquer les différents outils utilisés par la suite. Nous étudierons ensuite une première méthode basée sur la géométrie du problème. Un seconde méthode basée sur la résolution d'un problème d'optimisation et le livre de Jean Bernard Lasserre [10] sera présentée. Enfin, nous conclurons en résumant ces méthodes et en voyant les axes d'améliorations.

Chapitre 1

État de l'art et présentation du sujet

1.1 Analyse par intervalles

Pour rappel, un intervalle[4, 3] est l'ensemble des valeurs comprises entre deux valeurs appelées bornes. On note l'intervalle x : $[x] = [\underline{x}, \overline{x}]$ avec $\underline{x}$ la borne inférieur et $\overline{x}$ la borne supérieur.

Une boite est un vecteur intervalles. Dans cette section, je vais m'appuyer sur les cours de Mr Guyonneau et de codes disponibles sur son gitlab [1].

1.1.1 La classe Intervalle

Un intervalle a deux attributs : sa borne inférieure et sa borne supérieure. De nombreuses opérations doivent être implémentées pour cette classe : addition, soustraction, division, multiplication.

Toutes ces opérations doivent être implémentées entre deux intervalles mais aussi entre un intervalle et un nombre, lorsque c'est possible. Une implémention partielle en python peut se trouver en annexe.

Entre deux intervalles

- Addition : $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- Soustraction : $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- Multiplication : $[\underline{x}, \overline{x}] * [\underline{y}, \overline{y}] = [min(x_i * y_i), min(x_i * y_i)]$
  pour $x_i \in \{\underline{x}, \overline{x}\}$ et $y_i \in \{\underline{y}, \overline{y}\}$

1. https://gitlab.u-angers.fr/cours/interval_analysis_student

- Division : $[\underline{x}, \overline{x}]/[\underline{y}, \overline{y}]$. Le cas de la division est plus complexe. Nous allons inverser $[y]$ et le multiplier par $[x]$.
  - Si y est un ensemble vide, le résultat sera un ensemble vide
  - Si y contient 0 et 0 n'est pas une borne, le résultat sera $[x] * [-\infty, \infty]$
  - Si $\overline{y} == 0$, le résultat sera $[x] * [-\infty, \frac{1}{\underline{y}}]$
  - Si $\underline{y} == 0$, le résultat sera $[x] * [\frac{1}{\overline{y}}, \infty]$
  - Sinon, le résultat sera $[x] * [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}]$

1.1.2 Les fonctions d'intervalle

En analyse par intervalles, il y a deux autres opérateurs : l'intersection et l'union.

Intersection

L'intersection de deux intervalles donnera un intervalle avec toutes les valeurs communes aux deux intervalles. Le symbole représentant l'intersection est $\cap$.

Si ($\underline{x} < \underline{y}$ et $\overline{x} < \overline{y}$) ou ($\underline{x} > \underline{y}$ et $\overline{x} > \overline{y}$), alors

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

Sinon, si $[x] \subset [y]$, alors $[x] \cap [y] = [x]$

Sinon, l'intersection des deux intervalles est nulle : $[x] \cap [y] = \emptyset$

Union

Il existe deux opérateurs d'union : l'union, $\cup$, et l'union conjointe, $\sqcup$.

Si les deux intervalles sont disjoints, c'est-à-dire que leur intersection est nulle, l'union de deux intervalles est un ensemble d'intervalles, noté $[x] \cup [y]$, sinon, c'est l'intervalle qui comprend les deux autres.

Exemples :
$[2, 7] \cup [5, 12] = [2, 12]$
$[2, 5] \cup [7, 12] = [2, 5] \cup [7, 12]$

L'union conjointe de deux intervalles est un intervalle qui contient les deux intervalles en paramètre. On note $\sqcup$ l'union conjointe.

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

Dans la suite du rapport, nous considérerons l'union comme l'union conjointe.

1.2 Problème de satisfaction de contraintes

Les problèmes de satisfaction de contraintes (CSP [2]) [2, 8, 14] sont définis par 3 ensembles :

- Un ensemble de variables
- Un ensemble de domaines pour les variables
- Un ensemble de contraintes qui font apparaître des relations entre les variables

L'objectif est de réduire les domaines pour les variables tout en respectant les contraintes.

1.2.1 Un exemple, le Sudoku

Beaucoup de jeux de logique sont basés sur ce système : nous avons un ensemble de variables, un ensemble de domaines pour ces variables et un ensemble de contraintes sur ces variables. Un des exemples les plus connus est le sudoku. À chaque case correspond une variable. Chaque variable appartient à l'ensemble discret composé des entiers compris entre 1 et 9. Il y a trois contraintes : chaque chiffre n'est présent qu'une fois dans la ligne, chaque chiffre n'est présent qu'une fois dans la colonne et chaque chiffre n'est présent qu'une fois dans des carrés délimités de 3 par 3. On y ajoute les contraintes des chiffres déjà présents dans la grille, qui sont des variables contraintes à une valeur.

Les trois ensembles sont définis, nous pouvons donc appliquer des algorithmes propres aux CSP pour la résolution.

1.2.2 Résolution des CSP

Une fois le problème posé, nous allons chercher à le résoudre. Trois types de résolution sont connus et généralement utilisés.

Backtracking

Le Backtracking est un algorithme récursif. À l'initialisation du CSP, toutes les variables sont vides, aucune valeur n'est assignée nulle part. Nous allons alors fixer une variable et rappeler l'algorithme sur la variable suivante. Si une ou plusieurs contraintes ne sont pas respectées, alors on change la dernière variable. Si aucune valeur de la variable ne correspond, on remonte à la variable précédente et nous réitèrons l'opération, jusqu'à trouver une combinaison satisfaisant toutes les contraintes ou jusqu'à avoir testé toutes les possibilités. Nous pouvons comparer cette méthode à du brute force[7].

2. *Constraint Satisfaction Problem*

Propagation de contrainte

Pour chaque contrainte, on construit un contracteur qui permet de réduire les domaines des variables et on réitère chaque contracteur jusqu'à atteindre un point fixe.

Recherche locale

Cette méthode fonctionne par batch, c'est-à-dire par morceau. À l'initialisation, toutes les variables vont prendre une valeur quelconque dans leur ensemble de définition. On va ensuite venir changer des ensemble de variables par batch. Cette méthode est plus rapide mais n'assure pas de trouver une solution, même s'il en existe une. Dans certains cas, où les contraintes et les domaines concordent bien, s'il y a plusieurs solutions par exemple, cette méthode rapide sera la plus adaptée.

Les méthodes de recherche locale et de backtracking ne fonctionnent que sur des CSP dit discrets, comme le Sudoku par exemple, où les domaines des variables sont dénombrables. Dans notre cas, nous avons un CSP pour lequel les domaines des variables sont des intervalles donc continus. Nous utiliserons donc les techniques de propagation de contraintes.

1.3 Contracteurs entre deux intervalles

Un contracteur[4, 8] est associé à une contrainte. Le contracteur permet de réduire des intervalles admissibles pour les variables en respectant cette contrainte. On peut utiliser un contracteur ou un ensemble de contracteurs pour résoudre un CSP.

De nombreux contracteurs existent et sont utilisés, pour des opérations élémentaires comme l'addition, la soustraction, la multiplication, mais aussi par exemple pour une distance entre deux intervalles.

Nous allons présenter un contracteur simple, le contracteur d'addition, puis un contracteur plus complexe, le contracteur de distance.

1.3.1 Contracteur d'addition

Soit trois intervalles $[a]$, $[b]$ et $[c]$. Considérons la contrainte $[a] = [b] + [c]$. L'objectif du contracteur d'addition est de réduire les domaines pour les variables en satisfaisant la contrainte $[a] = [b] + [c]$.

Algorithme de contraction :

- $[a] = [a] \cap ([b] + [c])$
- $[b] = [b] \cap ([a] - [c])$

- $[c] = [c] \cap ([a] - [b])$

Exemple

Nous voulons $[a] = [b] + [c]$ avec les intervalles $[a]$, $[b]$ et $[c]$ suivant :

$[a] = [3, 15]$, $[b] = [2, 5]$, $[c] = [3, 13]$

On calcule la somme de $[b]$ et $[c]$, et la différence entre $[a]$ et $[b]$, puis celle entre $[a]$ et $[c]$:

$[b] + [c] = [5, 18]$, $[a] - [c] = [-10, 12]$, $[a] - [b] = [-2, 13]$

On contracte ensuite en faisant l'intersection entre les intervalles et leur image :

$[a] = [3, 15] \cap [5, 18] = [5, 15]$

$[b] = [2, 5] \cap [-10, 12] = [2, 5]$

$[c] = [3, 13] \cap [-2, 13] = [3, 13]$

De la même façon, il existe des contracteurs pour la multiplication et la puissance.

1.3.2 Contracteur de distance

Pour le contracteur de distance, nous considérons la contrainte :

$$[dst]^2 = ([x_B] - [x_A])^2 + ([y_B] - [y_A])^2$$

Cette contrainte va s'appliquer entre deux vecteurs d'intervalles à 2 composantes :

$$\begin{pmatrix} [x_A] \\ [y_A] \end{pmatrix} et \begin{pmatrix} [x_B] \\ [y_B] \end{pmatrix}$$

Contrairement au contracteur d'addition, le contracteur de distance est un contracteur qui s'applique sur une contrainte complexe que nous allons décomposer en contraintes simples. Nous allons pour cela créer des variables intermédiaires. L'assignation des variables est le Forward. Le passage dans le sens inverse est le Backward. Les mêmes variables étant utilisées dans plusieurs équations, on fait l'opération Backward plusieurs fois pour être sûr d'avoir la meilleure contraction.

Forward

- $[a] = [x_B] - [x_A]$
- $[b] = [a]^2$
- $[c] = [y_B] - [y_A]$

- $[d] = [c]^2$
- $[e] = [b] + [d]$
- $[f] = [dst]^2$

Backward

- $[f] = [f] \cap [e]$
- $[e] = [e] \cap [f]$
- $[e], [b], [d] = contracte\_addition([e], [b], [d])$
- $[d], [c] = contracte\_carre([d], [c])$
- $[b], [a] = contracte\_carre([b], [a])$
- $[x_A], [c], [x_B] = contracte\_addition([x_A], [c], [x_B])$
- $[y_A], [a], [y_B]s = contracte\_addition([y_A], [a], [y_B])$
- $[f], [dst] = contracte\_carre([f], [dst])$

Après itération du processus de Backward, nous obtenons un point fixe, c'est à dire que le contracteur renvoie la même chose en sortie que ce qu'il prend en entrée. Les ensembles pour $\begin{pmatrix}[x_A]\\[y_A]\end{pmatrix}$, $\begin{pmatrix}[x_B]\\[y_B]\end{pmatrix}$ et $[dst]$ sont réduits au mieux.

1.4 SIVIA

L'algorithme de SIVIA [3] est un algorithme de découpage qui permet de résoudre des problèmes d'inversion ensembliste, c'est-à-dire calculer l'image inverse d'un ensemble par une fonction, via l'analyse par intervalle[1]. L'algorithme de SIVIA est le suivant :

1. $\mathcal{L} = \{[\mathbf{x}_0]\}$ *Initialisation avec le domaine de départ à tester*
2. Tant que $\mathcal{L} \neq \emptyset$:
3. Extraire $[\mathbf{x}]$ de $\mathcal{L}$
4. si $[f]([\mathbf{x}]) \in \mathbb{Y}$, *dessiner [x] en rouge*
5. sinon si $[f]([\mathbf{x}]) \cap \mathbb{Y} = \emptyset$, *dessiner [x] en bleu*
6. sinon si $w([\mathbf{x}]) < \epsilon$, *dessiner [x] en jaune*
7. sinon bissecter($[\mathbf{x}]$) et mettre les deux parties dans $\mathcal{L}$

Avec $[f]([\mathbf{x}])$ la fonction d'inclusion de f. $[f]([\mathbf{x}])$ est un intervalle ou une boite qui contient toutes les images de $f([\mathbf{x}])$.

Les trois couleurs dépendantes du tests sont des conventions. Dans la suite du rapport, ces conventions seront utilisées lors de l'utilisation de SIVIA.

3. *Set Inversion Via Interval Analysis*

1.5 La méthode de Newton

La méthode de Newton [9] permet de trouver les racines d'une fonction $f, \mathbb{R}^n \to \mathbb{R}^n$, c'est-à-dire les valeurs $\mathbf{x}^*$ pour lesquelles la fonction s'annule, $f(\mathbf{x}^*) = \mathbf{0}$.

Il y a des cas où la méthode diverge, cela peut dépendre de la fonction ou du point initial choisi. Aujourd'hui, certaines théories, comme la théorie de Kantorovitch ou la théorie alpha de Smale [5], permettent d'assurer la convergence vers une racine.

La fonction f doit satisfaire certains critères :

- Elle doit être dérivable sur l'ensemble de définition considéré
- Elle doit être continue sur l'ensemble de définition considéré
- Sa dérivée doit être continue sur l'ensemble de définition considéré

C'est une méthode itérative qui construit une suite de points x_i qui converge vers une racine de f.

Avant d'expliquer comment fonctionne la méthode de Newton, nous allons voir pourquoi nous avons choisi cette méthode, avec l'exemple de la dichotomie.

La dichotomie permet, comme la méthode de Newton, de trouver un antécédent correspondant à une image. Pour cela, la méthode prend l'intervalle de définition, le coupe en deux, et regarde si la valeur cherchée se trouve dans l'image de chaque partie. Elle va garder la partie la comprenant, et rejeter l'autre. Nous avons donc :

$$\|x_k - x^*\| \leqslant (\tfrac{1}{2})^k \cdot \|x_0 - x^*\|$$

avec x_k la valeur estimée à k itérations, x^* la valeur recherchée, et x_0 la valeur initiale choisie.

Comme montré dans le livre de Jean-Pierre Dedieu [5], dans le cas de la méthode de Newton, nous avons une convergence quadratique :

$$\|x_k - x^*\| \leqslant (\tfrac{1}{2})^{2^k - 1} \cdot \|x_0 - x^*\|$$

On voit donc que la méthode de Newton converge plus rapidement que la dichotomie, c'est pourquoi nous avons choisi cette méthode.

1.5.1 Newton classique

La méthode de Newton fonctionne comme suit :

- Choix du x_0 initial
- Calcul de l'image $f(x_0)$

- Calcul de la tangente à f en ce point
- Calcul du point d'intersection entre la tangente et l'axe des abscisses

Le procédé est itéré en prenant le point d'intersection comme nouveau point de départ. L'algorithme s'arrête lorsque la précision souhaitée est atteinte s'il y a convergence ou si la méthode diverge.

L'équation de la tangente en un point x_k est

$$y = f'(x_k) \cdot (x - x_k) + f(x_k)$$

avec $f'(x_k)$ le coefficient directeur de la tangente.

Ici, nous cherchons à trouver $y = 0$, donc

$$\begin{aligned} 0 &= f'(x_k) \cdot (x - x_k) + f(x_k) \\ \Leftrightarrow x &= x_k - f'(x_k))^{-1} f(x_k) \end{aligned}$$

L'opérateur de Newton prend donc la forme suivante :

$$N_f(\mathbf{x}) = \mathbf{x} - Df^{-1}(\mathbf{x}) f(\mathbf{x})$$

Cet opérateur renvoie la valeur de x pour l'itération suivante. La figure 1.1 montre la construction graphique des premières itérations de la méthode de Newton sur une fonction quelconque.

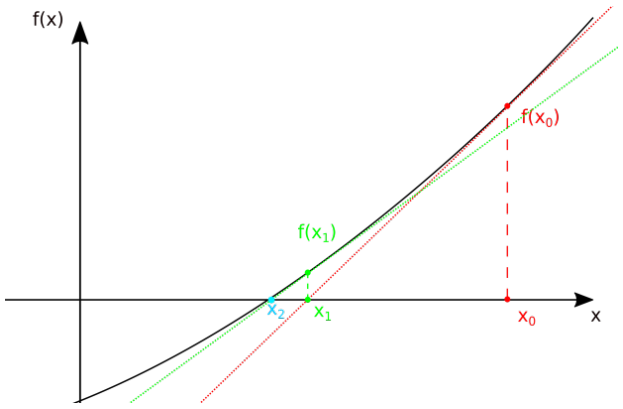

FIGURE 1.1 – Schéma des premières itérations de la méthode de Newton

Dans le cadre de cette démarche, nous allons nous intéresser à la méthode de Newton étendue aux intervalles.

1.5.2 Newton étendue aux intervalles

La méthode de Newton étendue aux intervalles[6] permet de trouver les racines d'une fonction, mais sert également à prouver l'existence d'une racine de f.

$$\text{Soit } f : \mathbb{R}^n \to \mathbb{R}^n$$
$$\exists \mathbf{x}^* \in \mathbb{R}^n, f(\mathbf{x}^*) = \mathbf{0}$$

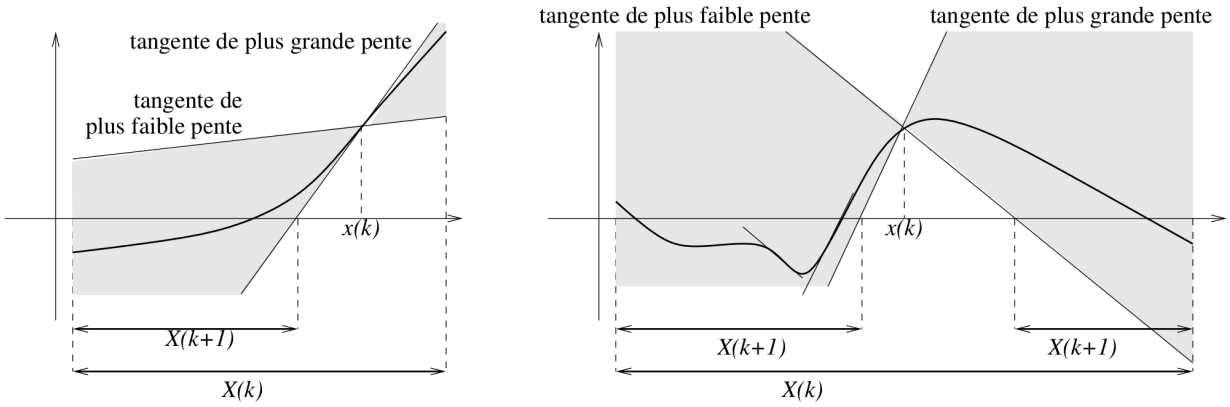

FIGURE 1.2 – Méthode de Newton par intervalles [6]

L'opérateur de Newton [6]

$$N_f(\tilde{x}, [\mathbf{x}]) = \tilde{x} - Df^{-1}([\mathbf{x}])f(\tilde{x})$$

avec $\tilde{\mathbf{x}} \in \mathbf{x}$. L'ensemble de l'intervalle $[\mathbf{x}]$ est pris en compte dans la dérivée puisque nous allons considérer l'ensemble des pentes.

Nous allons ensuite appliquer un algorithme pour trouver les intervalles contenant les racines de la fonction f.

Algorithme de Newton par intervalles [6] dans le cas d'un seule variable :
Données :
$\boldsymbol{f}$ une extension intervalle de f et $\boldsymbol{f'}$ une extension de f'
$\boldsymbol{x}$ un pavé de recherche
Résultat :
Res une liste de pavés susceptibles de contenir un zéro de f
Initialisations :
$\mathcal{L} := \{\boldsymbol{x}\}$ la liste des pavés en attente de traitement
Res := $\emptyset$
tant que $\mathcal{L} \neq \emptyset$ **faire**

sortir un pavé $\boldsymbol{x}$ de $\mathcal{L}$
$\boldsymbol{x'} := N(\tilde{x}, \boldsymbol{x})$ avec N un pas d'une méthode de Newton
si $\boldsymbol{x'} \neq \emptyset$ alors
  si $N(\tilde{x'}, \boldsymbol{x'}) \subset \boldsymbol{x'}$ alors ¡¡ $\boldsymbol{x'}$ contient un zéro ¿¿
  si $N(\tilde{x'}, \boldsymbol{x'}) \subset \text{int}(\boldsymbol{x'})$ alors ¡¡ $\boldsymbol{x'}$ contient un zéro ¿¿
  si $\boldsymbol{x'}$ satisfait le critère d'arrêt alors
    ranger $\boldsymbol{x'}$ dans Res
  sinon
    si $\boldsymbol{x'}$ est assez réduit par rapport à $\boldsymbol{x}$ dans $\mathcal{L}$
      ranger $\boldsymbol{x'}$ dans $\mathcal{L}$
    sinon
      $(\boldsymbol{x}_1, \boldsymbol{x}_2) := \text{split}(\boldsymbol{x'})$ (bissection de $\boldsymbol{x'}$)
      si $\boldsymbol{f}(\boldsymbol{x}_1) \ni 0$ alors ranger $\boldsymbol{x}_1$ dans $\mathcal{L}$
      si $\boldsymbol{f}(\boldsymbol{x}_2) \ni 0$ alors ranger $\boldsymbol{x}_2$ dans $\mathcal{L}$

Une fois le critère d'arrêt déterminé nous pouvons appliquer l'algorithme. Nous aurons donc un intervalle ou un ensemble d'intervalles qui contiennent une racine de f. Ce critère est le plus souvent une taille minimale d'intervalle.

La propriété que nous allons utiliser dans la suite est la suivante :

$$\text{si } N(\tilde{x'}, \boldsymbol{x'}) \subset \boldsymbol{x'} \text{ alors ¡¡ } \boldsymbol{x'} \text{ contient un zéro ¿¿}$$

Chapitre 2

Méthodes de résolution

2.1 Présentation du sujet

2.1.1 Présentation du problème

L'objectif de ce travail de recherche est de concevoir un contracteur global optimal $C_f^*([x],[y])$ capable de prendre en compte plusieurs contraintes complexes, en étudiant une contrainte triangulaire. On considère trois boites A, B et C, et trois distances : $d_1 = AB, d_2 = BC, d_3 = CA$ qui forment un triangle, comme le montre la figure 2.1.

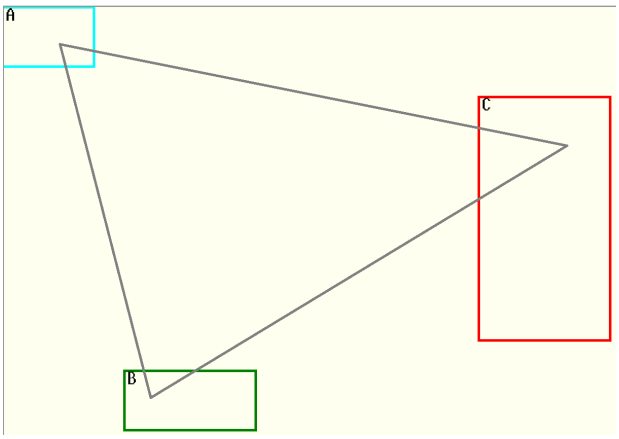

FIGURE 2.1 – Présentation du problème

Combinaison de contracteur de distance

La première approche a été d'appliquer le contracteur de distance sur les trois contraintes de distance (2.1),(2.2),(2.3).

$$d_1^2 = (x_A + x_B)^2 + (y_A + y_B)^2 \tag{2.1}$$
$$d_2^2 = (x_B + x_C)^2 + (y_B + y_C)^2 \tag{2.2}$$
$$d_3^2 = (x_A + x_C)^2 + (y_A + y_C)^2 \tag{2.3}$$

Pour vérifier si cette première contraction est optimale ou non, nous avons ensuite appliqué SIVIA sur l'ensemble, avec une fonction pour chaque boite. Pour le boite A :

$$f(\begin{pmatrix}[x_A]\\[y_A]\\\theta\end{pmatrix}) = \begin{pmatrix}[x_B]\\[y_B]\\[x_C]\\[y_C]\end{pmatrix}, \text{ avec } \theta \text{ l'angle entre } AC \text{ et l'horizontal.}$$

Sur les figures 2.2, on peut voir SIVIA appliqué à notre problème. La zone rouge est une zone où il existe au moins une solution. Ainsi, si un bord d'une boite touche la zone rouge, on ne pourra pas contracter ce bord. À l'inverse, si un bord ne touche pas de rouge, alors il est contractable. On peut donc voir qu'il reste des bords contractables.

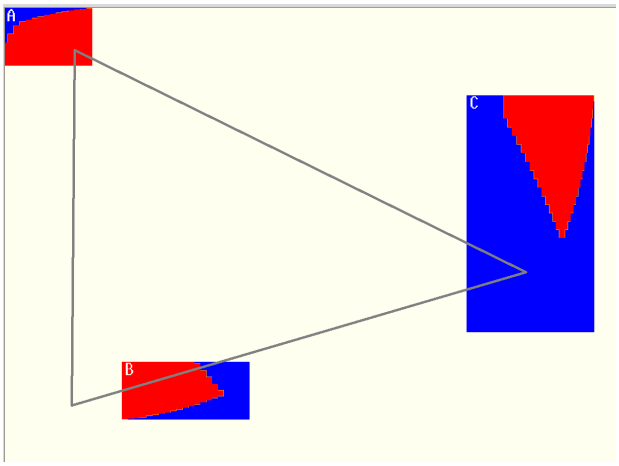

FIGURE 2.2 – SIVIA, distance AC compatible

2.1.2 Objectif du stage

L'objectif de ce travail de recherche est de concevoir un contracteur optimal permettant d'avoir des intervalles minimaux pour les boites A, B et C. Si nous reprenons l'exemple présenté en introduction, nous voulons être le plus précis possible sur les positions du robot et des deux obstacles détectés.

2.2 Méthode basée sur la géométrie du problème

Cette première méthode s'appuie sur la dimension géométrique du problème. Nous avons trois boites, séparées respectivement par trois distances, pouvant être représentées par un triangle.

2.2.1 Présentation du problème de satisfaction de contraintes

Nous avons donc six variables et six domaines, avec des contraintes sur ces variables. Les trois distance d_1, d_2, d_3 sont considérées comme fixes. Nous pouvons poser le CSP suivant :

- Ensemble de variables : $\{x_A, y_A, x_B, y_B, x_C, y_C\}$
- Ensemble de domaines :

$$\begin{aligned}\{x_A &\in [\underline{x_A}, \overline{x_A}],\\ y_A &\in [\underline{y_A}, \overline{y_A}],\\ x_B &\in [\underline{x_B}, \overline{x_B}],\\ y_B &\in [\underline{y_B}, \overline{y_B}],\\ x_C &\in [\underline{x_C}, \overline{x_C}],\\ y_C &\in [\underline{y_C}, \overline{y_C}]\}\end{aligned}$$

- Ensemble de contraintes :

$$\begin{aligned}\{d_1^2 &= (x_A + x_B)^2 + (y_A + y_B)^2,\\ d_2^2 &= (x_B + x_C)^2 + (y_B + y_C)^2,\\ d_3^2 &= (x_A + x_C)^2 + (y_A + y_C)^2\}\end{aligned}$$

Les variables x_i et les variables y_i étant liées, nous pouvons faire un projection dans $\mathbb{R}^2$. Ainsi, nous avons trois DDL [1] : T_x, T_y et R_z.

2.2.2 Propriétés du contracteur

Dans la suite, nous allons utiliser deux propriétés importantes pour notre algorithme.

Propriété 1

Les bords de S sont issus des bords des boites [A], [B] et [C].

1. *Degré de liberté*

Nous avons trois DDL : translation sur x, translation sur x et rotation sur z. La contraposée est la suivante : si on ne se trouve pas sur un bord des boites $[A]$, $[B]$ et $[C]$, alors on ne se trouve pas sur un bord de S. En effet, si on ne se trouve pas sur un bord de $[A]$, $[B]$ ou $[C]$, alors on peut translater sur x dans les deux sens, translater sur y dans les deux sens et tourner sur z dans les deux sens. Il y a donc un espace autour du point considéré qui est un espace solution. Ne pas être sur un bords de $[A]$, $[B]$ ou $[C]$ implique donc de ne pas être sur un bord de S.

Propriété 2

Les bords contractables sont les bords sur lesquels on ne peut pas placer de triangle.

Un bord va pouvoir être contracté tant qu'aucune configuration de triangle n'est possible sur celui-ci. Nous allons parcourir tous les triplets de bords existants et utiliser la méthode de Newton étendue aux intervalles pour faire la liste des bords où un triangle peut être placé.

Comme expliqué dans l'état de l'art, nous allons chercher une racine d'une fonction. Cette fonction prendra trois variables, qui sont les trois bords considérés. Nous aurons aussi des paramètres fixés. Pour chaque boite, nous aurons ainsi une coordonnée en variable, et une coordonnée en paramètre. La fonction considérée sera une fonction composée des trois contraintes de distance.

2.2.3 Présentation de la solution

Notre solution se passe en trois étapes.

Contraction contrainte à contrainte

Pour commencer, nous appliquons les contracteurs propres à chaque contrainte. Cette partie ne nous donne pas la solution optimale, mais c'est une première réduction des ensembles.

Détection des bords contractable

La deuxième étape est la détection des bords contractables. Nous allons pour cela utiliser la seconde propriété, avec la méthode de Newton étendue aux intervalles. En testant tous les triplets de bords existants, nous allons avoir la liste des bords qui ne sont pas contractables, et la liste des bords contractables. Nous allons dans la troisième partie chercher à contracter ceux-ci.

Recherche des sommets de l'ensemble solution

D'après la propriété 1, les bords de l'ensemble solution sont issus des bords des ensembles $[A]$, $[B]$ et $[C]$. Les bords de la boite $[C]$ réduite au mieux sont donc issus des bords de $[A]$ et de $[B]$. Nous allons donc fixer les trois DDL en utilisant les bords de $[A]$ et de $[B]$. Pour cela, nous allons placer le sommet A du triangle sur un coin de la boite $[A]$, ce qui fixe les deux translations à un extremum, puis nous allons fixer la rotation en fixant le sommet B sur un bord de $[B]$, comme illustrer sur la figure 2.3.

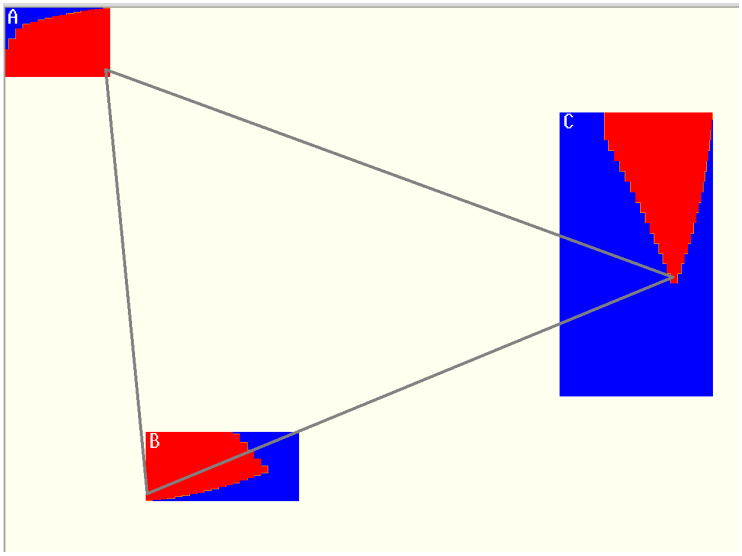

FIGURE 2.3 – Extremums de [C]

Nous allons ensuite itéré ce processus pour chaque coin de $[A]$ avec les bords de $[B]$, et chaque coin de $[B]$ avec les bords de $[A]$. Nous aurons ainsi un ensemble de points, comme le montre la figure 2.4. De cet ensemble de point, nous allons réduire les bords contractables jusqu'à ce qu'ils arrivent à un point existant. Tous les bords de $[C]$ auront donc une configuration possible de triangle, ils seront donc tous réduits au maximum. Nous itérons ensuite ce procédé sur les boites $[A]$ et $[B]$ pour avoir les trois ensembles réduits au mieux.

2.2.4 Conclusion

Avantages

Cet algorithme est rapide comparer au suivant. En effet, placer les triangles est une opération simple, et la méthode de Newton n'est utilisée que pour prouver l'existence d'une racine, nous ne l'utilisons pas pour trouver ces racines, ce qui pourrait prendre du temps.

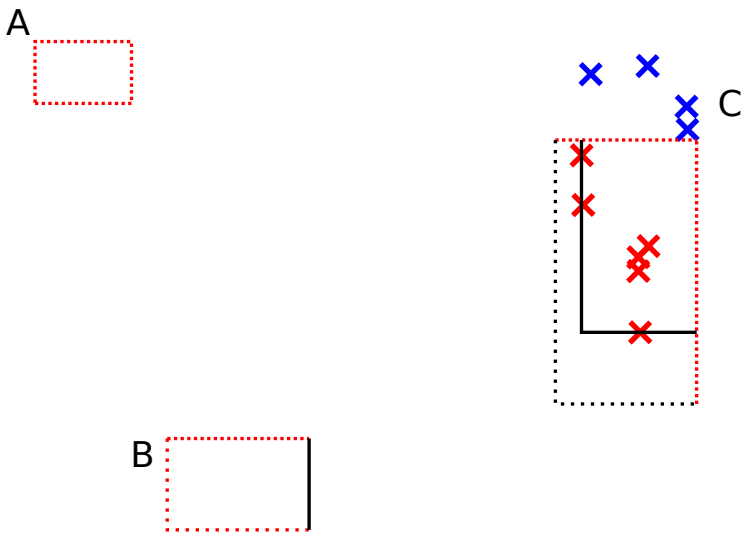

FIGURE 2.4 – Contraction de C

Inconvénients

Cette solution est difficile à implémenter, et elle répond à ce problème spécifiquement. De plus, certains soucis restent présents. Par exemple, lors du passage de la méthode de Newton, si les intervalles de deux boites ne sont pas disjoints ($[x_A] \cap [x_B] \neq \emptyset$ ou $[y_A] \cap [y_B] \neq \emptyset$), nous trouvons une division par 0, ambiguïté qu'il faut lever.

2.3 Méthode basée sur la résolution d'un problème d'optimisation

Le problème peut être considéré comme un problème d'optimisation, avec une fonction objectif linéaire et des contraintes quadratiques :

$$\begin{array}{rl} \text{Minimiser} & x_A \\ \text{Sous contraintes} & d_1^2 = (x_A - x_B)^2 + (y_A - y_B)^2 \\ & d_2^2 = (x_B - x_C)^2 + (y_B - y_C)^2 \\ & d_3^2 = (x_A - x_C)^2 + (y_A - y_C)^2 \\ & x_A \in [x_A],\ y_A \in [y_A] \\ & x_B \in [x_B],\ y_B \in [y_B] \\ & x_C \in [x_C],\ y_C \in [y_C] \end{array}$$

Cette méthode s'appuie sur le livre de Jean Bernard Lasserre [10], dans lequel il présente une méthode de résolution de problèmes non linéaires par l'approche des SOS [2].

2.3.1 Sum Of Square

Un polynôme homogène $h(x)$ de degré pair p est SOS s'il peut être écrit sous la forme :

$$h(x) = \sum_{i=1}^{k} g_i(x)^2$$

avec $g_i(x)$ des polynômes de degré inférieur ou égal à la moitié de p.

Polynômes homogènes

Un polynôme homogène est un polynôme dont tous les monômes non nuls sont de même degré total. Autrement dit, on peut écrire un polynôme $p(x, y)$ homogène de degré j :

$$p(x, y) = \sum_{i=0}^{j} k_i \cdot x^i \cdot y^{j-i}$$

avec $k_i \in \mathbb{R}$. Ce polynôme est homogène de degré j car, pour tout monôme, la somme des exposants de x et y est égale à j.

2. *Sum Of Square* : Un polynôme pouvant s'écrire comme une somme de carré

Somme de carrés

Un polynôme $h(\mathbf{x}) \subset \mathbb{R}[\mathbf{x}]_{2d}$ est SOS si et seulement si il existe une matrice $\boldsymbol{Q} \in \mathbb{R}^{s(d)\times s(d)}$ symétrique et SDP [3] telle que :

$$h(\mathbf{x}) = \mathbf{v}_d(\mathbf{x})^T \boldsymbol{Q} \mathbf{v}_d(\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^n$$

avec $\mathbf{v}_d(\mathbf{x})$ un vecteur combinaison des $x_i \in \mathbf{x}$.

Une matrice SDP est une matrice dont les valeurs propres sont positives ou nulles. On note $\boldsymbol{Q} \succeq 0$.

Exemple de SOS [10]

On considère le polynôme bivarié sur $\mathbb{R}[\mathbf{x}] = \mathbb{R}[x_1, X_2]$

$$\mathbf{x} \mapsto f(\mathbf{x}) = 2x_1^4 + 2x_1^3x_2^1 - x_1^2x_2^2 + 5x_2^4,\ \mathbf{x} \in \mathbb{R}^2$$

On veut vérifier si f est SOS ou non.

$$\begin{aligned} f(x,y) &= 2x_1^4 + 2x_1^3x_2^1 - x_1^2x_2^2 + 5x_2^4 \\ &= \begin{pmatrix} x_1^2 & x_2^2 & x_1x_2 \end{pmatrix} \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{bmatrix} \begin{pmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \end{pmatrix} \\ &= q_{11}x_1^4 + q_{22}x_2^4 + (q_{33} + 2q_{12})x_1^2x_2^2 + 2q_{13}x_1^3x_2 + 2q_{23}x_1x_2^3 \end{aligned}$$

avec $\mathbf{Q} = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{bmatrix}$.

On veut donc $\mathbf{Q} \succeq 0$:

$$\begin{cases} q_{11} = 2 \\ q_{22} = 5 \\ q_{33} + 2q_{12} = -1 \\ 2q_{13} = 2 \\ q_{23} = 0 \end{cases}$$

On a donc, d'après [10] :

$$0 \preceq \boldsymbol{Q} = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix} = \boldsymbol{H}\boldsymbol{H}^T, pour \boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & 0 \\ -3 & 1 \\ 1 & 3 \end{bmatrix}$$

3. *Semi-Définie positive*

On en déduit l'écriture de $f(x_1, x_2)$:

$$f(x_1, x_2) = \frac{1}{2}(2x_1^2 - 3x_2^2 + x_1x_2)^2 + \frac{1}{2}(x_2^2 + 3x_1x_2)^2$$

$f(x_1, x_2)$ est donc SOS.

2.3.2 Démarche

Comme présenté au début de la section, le problème peut être vu comme un problème d'optimisation avec une fonction objectif linéaire avec des contraintes quadratiques. Nous allons alternativement minimiser et maximiser chaque variable pour trouver les bornes des intervalles minimisés.

Les contraintes étant quadratiques, donc SOS, on peut appliquer la méthode décrite dans le livre de Lasserre [10] pour résoudre ce problème.

On a utilisé des bibliothèques appelées Ipopt[11] et JuMP[12], qui implémentent ce système de résolution en Julia [13], un langage de programmation de haut niveau.

2.3.3 Conclusion

Avantages

Si un problème similaire mais de dimension supérieure se pose, il suffit d'ajouter des variables et des contraintes. Cette méthode est donc très adaptative.

Inconvénients

Cette méthode est plutôt lente. L'utilisation de méthodes d'optimisation sur des problèmes quadratiques n'est pas encore développée pour les problèmes similaires au notre.

Conclusion et perspectives

Rappel du problème

L'objectif de ce travail de recherche était de trouver un contracteur réduisant au mieux des intervalles avec des contraintes données. Nous avons trois boites, A, B et C, et trois distances, d_1, d_2 et d_3. Après un état de l'art et une réduction avec les contracteurs de distance 2 à 2 existants, nous avons constaté que les boites n'avaient pas atteint leur taille minimale.

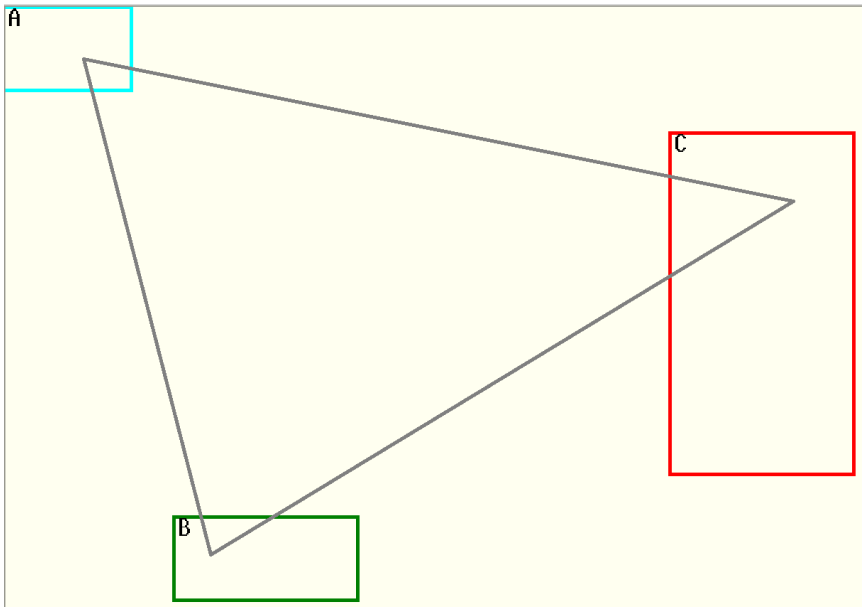

FIGURE 2.5 – Rappel du problème

Méthodes de résolution du problème

Méthode s'appuyant sur la géométrie du problème

La première méthode considère le côté géométrique du problème. Nous avons cherché à générer un nuage de points, faisant partie de la boite à réduire, à partir des extremums des deux autres boites. Depuis ce nuage de points, nous avons sélectionné les bords à réduire grâce à la méthode de Newton étendue aux intervalles, puis réduit ces bords. Nous avons itéré le procédé sur les trois boites pour trouver une contraction minimale.

Méthode basée sur la résolution d'un problème d'optimisation

La seconde méthode proposée s'appuyait sur l'approche d'un problème d'optimisation. Nous avons des variables à maximiser et à minimiser, pour trouver les bornes des intervalles, et nous avons des contraintes de distance. La fonction objectif est donc linéaire, et les contraintes quadratiques. Grâce à la méthode proposée par Jean Bernard Lasserre [10], nous pouvons résoudre ce problème d'optimisation. Celui-ci a été implémenté en Julia, grâce aux bibliothèques Ipopt et JuMP.

Axes d'amélioration

La seconde méthode étant une utilisation d'un procédé existant, il n'y a pas d'axe d'amélioration pour notre partie.

La première méthode pourrait être adaptée aux problèmes de dimensions supérieures, dans le cas où le robot détecterait trois ou quatre obstacles par exemple. De plus, l'implémentation faite en python n'est pas complète. On pourrait aussi considérer choisir un autre langage, de plus bas niveau et donc plus rapide pour optimiser les calculs.

Bibliographie

[1] L. Jaulin et E. Walter. « Set inversion via interval analysis for nonlinear bounded-error estimation ». In : *Automatica* 29.4 (1993), p. 1053-1064.

[2] Sally C. Brailsford, Chris N. Potts et Barbara M. Smith. « Constraint satisfaction problems : Algorithms and applications ». In : *European Journal of Operational Research* 119.3 (16 déc. 1999), p. 557-581. ISSN : 0377-2217. DOI : 10.1016/S0377-2217(98)00364-6. URL : https://www.sciencedirect.com/science/article/pii/S0377221798003646 (visité le 05/02/2022).

[3] Götz Alefeld et Günter Mayer. « Interval analysis : theory and applications ». In : *Journal of Computational and Applied Mathematics* 121.1 (1er sept. 2000), p. 421-464. ISSN : 0377-0427. DOI : 10.1016/S0377-0427(00)00342-3. URL : https://www.sciencedirect.com/science/article/pii/S0377042700003423 (visité le 06/02/2022).

[4] Luc Jaulin et al. *Applied Interval Analysis*. Sous la dir. de Luc Jaulin et al. Springer, 2001, p. 379. ISBN : 978-1-4471-0249-6. DOI : 10.1007/978-1-4471-0249-6_1. URL : https://doi.org/10.1007/978-1-4471-0249-6_1 (visité le 05/02/2022).

[5] Jean-Pierre Dedieu. *Points fixes, zéros et la méthode de Newton*. 2003, p. 207. ISBN : 978-3-540-30995-6. URL : https://link.springer.com/content/pdf/10.1007/3-540-37660-7.pdf (visité le 09/06/2022).

[6] Nathalie Revol. *Introduction à l'analyse par intervalles*. Rapp. tech. 2004, p. 37. URL : https://hal.inria.fr/inria-00072290/document (visité le 08/06/2022).

[7] Ababneh Mohammad, Oqeili Saleh et Rawan A. Abdeen. « Occurrences Algorithm for String Searching Based on Brute-force Algorithm ». In : *Journal of Computer Science* 2.1 (1er jan. 2006), p. 82-85. ISSN : 15493636. DOI : 10.3844/jcssp.2006.82.85. URL : http://www.thescipub.com/abstract/?doi=jcssp.2006.82.85 (visité le 09/02/2022).

[8] Gilles Chabert et Luc Jaulin. « Contractor programming ». In : *Artificial Intelligence* 173.11 (1er juill. 2009), p. 1079-1100. ISSN : 0004-3702. DOI : 10.1016/j.artint.2009.03.002. URL : https://www.sciencedirect.com/science/article/pii/S0004370209000381 (visité le 05/02/2022).

[9] Tan LEI. « La méthode de Newton et son fractal ». In : (2009), p. 20. URL : https://hal.archives-ouvertes.fr/hal-00585123/document (visité le 14/06/2022).

[10] Jean Bernard LASSERRE. *An Introduction to Polynomial and Semi-Algebraic Optimization*. 2015, p. 340. ISBN : 9781107447226.

[11] URL : https://www.juliapackages.com/p/ipopt (visité le 29/06/2022).

[12] URL : https://www.juliapackages.com/p/jump (visité le 29/06/2022).

[13] URL : https://julialang.org/ (visité le 29/06/2022).

[14] Bernhard NEBEL, Julien HUÉ et Stefan WÖLFL. « Constraint Satisfaction Problems - Qualitative Representation and Reasoning ». In : . *Object* (), p. 61.

Abstract — The objective of this master is to find a new triangle contractor. This is used for example in robot localisation, when a sensor gets two obstacles. We want to have a better estimation of the position of the robot. As we use interval analysis, the positions are not points but boxs, which is a vector of 2 intervals.

To do so, we used two methods. The first one used the geometrical problem. We compute a cloud of points from two boxs, and we reduce the third one.

The second method used optimization. We have a set of variable, a set of domains for the variables and a set of constraints. We want to minimize the intervals, so we want to find the minimum and the maximum for each variable. We used Jean Bernard Lasserre's work to resolve this problem, with linear objective function and quadratic constraints.

Keywords : Interval analysis, contractors, global contractors, CSP, Newton method, Optimization

Polytech Angers
62, avenue Notre Dame du Lac
49000 Angers