MASTER Ingénierie des Systèmes Industriels et des Projets

SPÉCIALITÉ : SYSTÈMES DYNAMIQUES ET SIGNAUX

Année 2009 / 2010

Thèse de Master SDS

Présentée et soutenue par :

Rémy GUYONNEAU

le 16 juillet 2010
Au sein de l'Institut des Sciences et Techniques de l'Ingénieurs d'Angers

TITRE

Cartographie par des Approches Ensemblistes

JURY

Président :	Laurent Hardouin	Responsable du Master Systèmes Dynamiques et Signaux de l'université d'Angers et de l'équipe Modèles Dynamiques et Système du Lisa
Examinateurs :	Sébastien Lagrange	Maître de Conférences
	Bertran Cottenceau	Maître de Conférences

Directeur de thèse : **Mr Sébastien Lagrange**

Laboratoire :

62, avenue Notre Dame du Lac
49000 Angers

Remerciements

Je tiens à remercier tous ceux que je n'ai pas mentionné dans ce qui suit.

Je souhaite remercier Mr Jean-Louis Boimond, directeur du LISA pour m'avoir accueilli au sein du laboratoire.

Je tiens particulièrement à remercier Mr Sébastien Lagrange, mon maître de stage, pour son aide, ses explications et sa disponibilité.

Je remercie également Mr Laurent Hardouin pour m'avoir donné la chance de faire ce stage.

Je tiens aussi à remercier ceux qui ont partagé le bureau des Masters SDS avec moi pour leur aide et leur bonne humeur.

Mes remerciements vont enfin à tous les membres du LISA pour leur accueil et leur soutien.

Table des matières

1 Introduction

Ce rapport fait suite au stage que j'ai eu la chance d'effectuer au LISA (Laboratoire d'Ingénierie des Systèmes Automatisés de l'université d'Angers), à la fin de mon Master Recherche option Systèmes Dynamiques et Signaux. Pendant ce stage j'ai travaillé sur l'approche ensembliste d'un problème de type SLAM[1].

Dans un premier temps, on définira ce qu'est l'analyse par intervalles. Cette notion a eu une place importante tout au long de ce stage. On abordera l'arithmétique des intervalles ainsi que les contracteurs et la propagation de contraintes.

Ensuite nous présenterons la problématique SLAM (Simultaneous Localization and Mapping), ainsi que son approche ensembliste. On introduira aussi l'algorithme ICP (Iterative Closest Point) utilisé pour des SLAM non-ensemblistes.

Pour finir on présentera les travaux effectués dans le but de réaliser un recalage ensembliste. On présentera deux algorithmes, un utilisant la propagation de contraintes, l'autre se servant de la bissection d'intervalles.

[1] Simultaneous Localization And Mapping.

2 Analyse par intervalles

Cette section a été largement inspiré par [2]. Afin de bien comprendre les avantages à utiliser l'analyse par intervalles pour résoudre des problèmes liés à la robotique, il est important de définir certaines notions, bases, de cette approche ensembliste.

2.1 Présentation de l'analyse par intervalles

L'analyse par intervalles est une approche mathématique pour laquelle on n'utilise non plus des nombres, mais des intervalles.

Définitions :

→ Un *intervalle* est un compact de $\mathbb{R}$. Soit une variable x, on définit l'intervalle $[x] = [\underline{x}, \overline{x}]$, avec $\underline{x}$ la plus petite valeur possible pour x et $\overline{x}$ la plus grande.

→ Soit A un ensemble de $\mathbb{R}$, $[A]$ correspond à *l'enveloppe intervalle* de A.

En plus des opérations propres aux ensembles (intersection, union...) qui s'appliquent naturellement aux intervalles, on peut étendre les opérations des nombres sur les intervalles et ainsi développer une arithmétique des intervalles.

2.1.1 Une arithmétique des intervalles

Comme énoncé plus haut, on peut appliquer les opérations des ensembles sur les intervalles. Soit $[x]$ et $[y]$ deux intervalles, on peut définir l'intersection :

$$[x] \cap [y] = \{z | z \in [x] \text{ et } y \in [y]\}.$$

On peut aussi définir l'union $\sqcup$:

$$[x] \sqcup [y] = [[x] \cup [y]].$$

Exemples :

$$[5, 10] \cap [7, 18] = [7, 10],$$
$$[5, 10] \sqcup [7, 18] = [5, 18],$$
$$[3, 5] \sqcup [8, 10] = [3, 10] \text{ (figure 2.1).}$$

FIG. 2.1 – L'union de deux intervalles : $[3, 5] \sqcup [8, 10] = [3, 10]$.

En plus de ces opérations, on peut généraliser les quatre opérations classiques de l'arithmétique des réels aux intervalles. Soit $\diamond = \{+, -, \times, \div\}$, on a :

$$[x] \diamond [y] = [\{x \diamond y \in \mathbb{R} | x \in [x], y \in [y]\}].$$

On peut écrire, à titre d'exemple :

$$([1, 2.2] \times [0, 2]) + [1, 3] = [0, 4.4] + [1, 3] = [1, 7.4],$$
$$1/[-2, 2] =]-\infty, \infty[,$$
$$[3, 4]/[0, 0] = \emptyset.$$

En plus de ces opérations, on peut associer les fonctions utilisées sur les nombres, sur les intervalles. Soit $[x]$ un intervalle, on peut donc calculer $cos([x])$, $sin([x])$, $([x])^2$, $\sqrt{[x]}$, ... A titre d'exemple :

$$([-5, 3])^2 = [0, 25],$$
$$cos([0, \pi/2]) = [0, 1].$$

2.1.2 Vecteur d'intervalles

Un vecteur d'intervalles $[\mathbf{x}]$ est un sous ensemble de $\mathbb{R}^n$ qui est défini comme le produit cartésien de n intervalles non vides.

$$[\mathbf{x}] = [x_1] \times [x_2] \times ... \times [x_n], \text{ avec } [x_i] = [\underline{x_i}, \overline{x_i}] \text{ pour } i = 1, ..., n.$$

Durant le stage les vecteurs d'intervalles les plus utilisés ont été de dimensions $n = 2$ et $n = 3$.

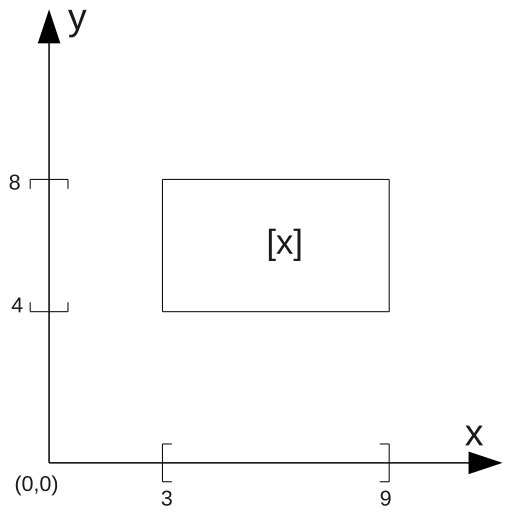

FIG. 2.2 – Un vecteur d'intervalles de dimension 2, $[3, 9] \times [4, 8]$.

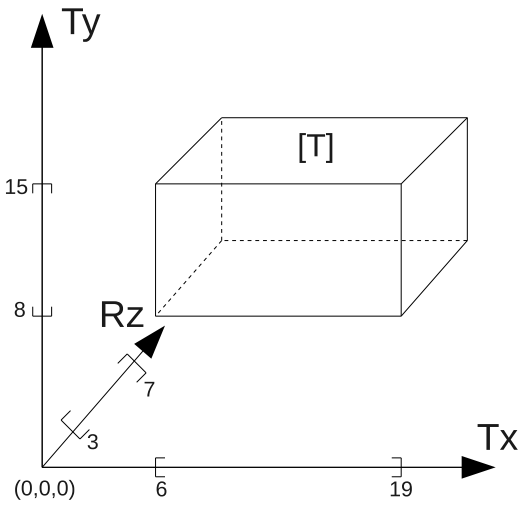

FIG. 2.3 – Un vecteur d'intervalles de dimension 3, $[6, 19] \times [8, 15] \times [3, 7]$.

2.2 Constraint Satisfaction Problem

Considérons un CSP[1] : soit $V = \{x_1, x_2, ..., x_n\}$ l'ensemble des variables, $D = \{[x_1], [x_2], ..., [x_n]\}$ l'ensemble des domaines pour ces variables et $C = \{c_1, c_2, ..., c_m\}$ l'ensemble des contraintes (relations) reliant les variables entre elles. La propagation de contraintes correspond au fait de réduire les domaines en utilisant les contracteurs des contraintes.

Un contracteur est un opérateur associée à une contrainte qui permet de réduire des domaines

[1]Constraint Satisfaction Problem.

d'appartenances des variables tout en respectant cette contrainte.

Exemple : considérons le CSP suivant :

$$\begin{cases} [x_1] = [-\infty, 10] \\ [x_2] = [-\infty, 7] \\ [x_3] = [1, +\infty] \\ x_3 = x_1 + x_2 \end{cases}.$$

- $x_3 = x_1 + x_2 \Rightarrow [x_3^*] = [1, +\infty] \cap ([-\infty, 10] + [-\infty, 7]) = [1, +\infty] \cap [-\infty, 17] = [1, 17]$.
- $x_1 = x_3 - x_2 \Rightarrow [x_1^*] = [-\infty, 10] \cap ([1, +\infty] - [-\infty, 7]) = [-\infty, 10] \cap [-6, +\infty] = [-6, 10]$.
- $x_2 = x_3 - x_1 \Rightarrow [x_2^*] = [-\infty, 7] \cap ([1, +\infty] - [-\infty, 10]) = [-\infty, 7] \cap [-9, +\infty] = [-9, 7]$.

On obtient donc les domaines contractés suivants : $[x_1^*] = [-6, 10]$, $[x_2^*] = [-9, 7]$ et $[x_3^*] = [1, 17]$. On a réduit les domaines des variables sans perdre de solutions vis à vis des contraintes.

2.2.1 Quelques contracteurs simples

Ci suivent les algorithmes des quatre contracteurs des opérations $+, -, \times, \div$:

Algorithm 1 Contracteur Addition ($z = x + y$)

Require: $[x], [y], [z]$
1: $[z^*] = [z] \cap ([x] + [y])$
2: $[y^*] = [y] \cap ([z] - [x])$
3: $[x^*] = [x] \cap ([z] - [y])$
4: **return** $[x^*], [y^*], [z^*]$.

Algorithm 2 Contracteur Soustraction ($z = x - y$)

Require: $[x], [y], [z]$
1: $[z^*] = [z] \cap ([x] - [y])$
2: $[y^*] = [y] \cap ([x] - [z])$
3: $[x^*] = [x] \cap ([z] + [y])$
4: **return** $[x^*], [y^*], [z^*]$.

Algorithm 3 Contracteur Multiplication ($z = x \times y$)

Require: $[x], [y], [z]$
1: $[z^*] = [z] \cap ([x] \times [y])$
2: $[y^*] = [y] \cap ([z]/[x])$
3: $[x^*] = [x] \cap ([z]/[y])$
4: **return** $[x^*], [y^*], [z^*]$.

Algorithm 4 Contracteur Division ($z = x \div y$)

Require: $[x], [y], [z]$

1: $[z^*] = [z] \cap ([x] \div [y])$
2: $[y^*] = [y] \cap ([x] \div [z])$
3: $[x^*] = [x] \cap ([z] \times [y])$
4: **return** $[x^*], [y^*], [z^*]$.

Pour la contraction en présence de contraintes plus complexes, on peut décomposer la (les) contrainte(s) en contraintes élémentaires.

Exemple : La contrainte $y = (x_1 + x_2) \times \dfrac{x_3}{x_4} + x_5$ se décompose en :

- $[a] = [x_1] + [x_2]$,
- $[b] = \dfrac{[x_3]}{[x_4]}$,
- $[c] = [a] \times [b]$,
- $[y] = [c] + [x_5]$.

On utilise donc les contracteurs de ces nouvelles contraintes. Afin de contracter les domaines au maximum on peut être amené à effectuer la contraction des variables à plusieurs reprises.

3 Simultaneous Localization and Mapping

D'après [6], un problème SLAM[1] peut se définir comme suit : *"Un robot se déplace dans un environnement inconnu avec deux buts : se localiser (déterminer sa trajectoire) et établir une carte de son environnement"*. Le problème c'est que pour se localiser on a besoin d'une carte, et pour construire une carte il faut savoir où on est...

3.1 Le contexte

Dans le cadre du défi CAROTTE (CArtographie par RObоT d'un TErritoire) organisé par l'ANR[2], un consortium s'est mis en place afin de participer au dis défit. Le consortium est composé du LISA[3], du LORIA[4] et de WANY robotics.

"Il s'agit de retenir de 4 à 8 consortiums qui s'affronteront autour du défi de réaliser un système robotisé autonome, capable de s'orienter dans un espace clos et de reconnaître des objets présents dans ce local, ce qui lui permettra de réaliser une cartographie accompagnée d'annotations sémantiques d'un espace inconnu"[5].

Afin de concourir à la première manche de ce défit, un robot prototype a été mis en place. Il s'agit d'un robot à roues différentielles équipé d'un PC104 et d'un télémètre à balayage laser (URG-04LX de chez Hokuyo) monté sur un portique articulé (figure 3.1).

Le robot est capable d'avoir une approximation de sa position à l'aide de l'odométrie, cependant cette dernière est entachée d'erreurs (dues principalement au glissement des roues) qui se cumulent au fur et à mesure de son déplacement (figure 3.4).

Le télémètre permet d'avoir une évaluation des distances entre le robot et les différents objets

[1] Simultaneous Localization And Mapping.
[2] Agence Nationale de la Recherche.
[3] Laboratoire d'Ingénierie des Systèmes Automatisés.
[4] Laboratoire Lorrain de Recherche en Informatique et ses Applications.
[5] http ://www.agence-nationale-recherche.fr/AAP-240-Carotte.html.

de l'environnement (figure 3.2).

Pour limiter la dérive du robot, une solution propose de scanner régulièrement l'environnement et de recaler les scans entre eux. On scanne l'environnement, on se déplace et on re-scanne. On recale le deuxième scan sur le premier ce qui nous donne une évaluation du déplacement fait (figure 3.2). L'algorithme ICP[6] permet de faire ce recalage.

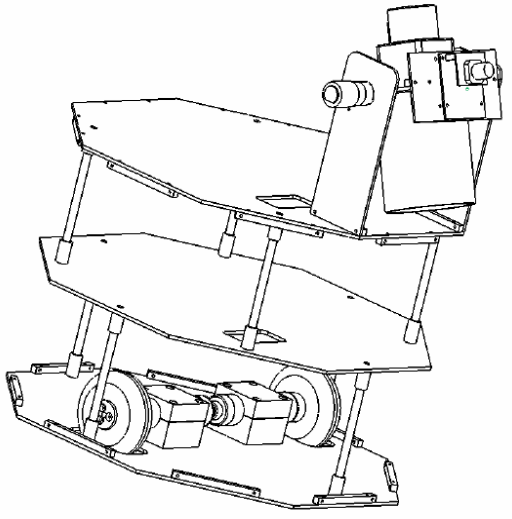

FIG. 3.1 – Illustration CAO du prototype.

FIG. 3.2 – Deux scans faits à l'aide d'un URG-04LX. En recalant le jeu gris sur le jeu noir on aura une estimation du déplacement effectué entre les deux scans.

[6]Iterative Closest Point.

3.2 Présentation d'ICP

Iterative Closest Point est un algorithme utilisé pour minimiser la différence entre deux jeux de points. Dans le cas d'une problématique SLAM, il est utilisé pour recaler deux jeux de mesures entre eux et ainsi avoir une indication sur le déplacement qu'a fait le robot entre ces deux jeux. C'est un algorithme itératif qui trouve la transformation (rotation et translation) permettant de minimiser l'écart spatial entre deux nuages de points.

Soit $M = \{X_{M_1}, ..., X_{M_m}\}$ et $S = \{X_{S_1}, ..., X_{S_n}\}$ deux jeux de mesures avec $X_{A_b} = (x_{A_b}, y_{A_b})$ les coordonnées de chaque mesure. On cherche de façon itérative la transformation T (en partant d'une approximation T_0) qui permet de recaler S sur M de telle sorte que la somme des écarts au carrés soit minimale. ICP fonctionne en deux étapes (algorithme 5) :

- $\rightsquigarrow$ *Match :* A chaque X_{S_i} on associe un X_{M_i}. Le critère classique pour l'association est celui du plus proche voisin : on associe à X_{S_i} le X_{M_i} le plus proche.
- $\rightsquigarrow$ *Optimisation :* On cherche la transformation T qui minimise la somme des distances euclidiennes au carré entre X_{S_i} et le X_{M_i} associé.

Algorithm 5 ICP

Require: soit $M = \{X_{M_1}, ..., X_{M_m}\}$ et $S = \{X_{S_1}, ..., X_{S_n}\}$ deux scans avec $X_{A_b} = (x_{A_b}, y_{A_b})$, et $T_0 = (t_{x_0}, t_{y_0}, r_{z_0})^T$ une transformation initiale.

1: $T \leftarrow T_0$
2: On transforme S avec T
3: **repeat**
4: **for all** $X_{S_i} \in S$ **do**
5: *Match :* On cherche le point $X_{M_k} \in M$ le plus proche de X_{S_i}. On obtient ainsi un jeu de correspondances $C = \{(X_{M_k}, X_{S_i})\}_i$.
6: **end for**
7: *Optimisation :* On trouve $T' = (t'_x, t'_y, r'_z)^T$ qui minimise $\epsilon = \Sigma_{\forall i}(distance(X_{M_k}, X_{S_i}))^2$
8: On transforme S en utilisant T' et on met à jour T
9: **until** Convergence
10: **return** T.

Ligne 7 : *distance* retourne la distance euclidienne entre les deux points X_{M_k} et X_{S_i}.

Il existe une multitude de variations à cet algorithme [10, 4], cependant elles ont toutes la même structure. Elles se différencient essentiellement sur les deux étapes *Match* et *Optimisation*.

L'étape *Match* qui a une complexité de $o(n^2)$ peut être rendue plus rapide en utilisant des k-d trees (annexe A.2). C'est ce qui est fait en pratique.

Pour ce qui est de l'étape *Optimisation*, en pratique on utilise les algorithmes de Gauss-Newton (annexe B.1) ou Levenberg-Marquardt (annexe B.2).

3.3 Une approche ensembliste

Il existe différentes approches pour un problème de type SLAM, ce qui nous intéresse ici c'est l'approche ensembliste, comme présentée dans [7, 8].

Dans une approche ensembliste un robot est identifié par un vecteur d'intervalles de dimension trois : un intervalle caractérisant l'incertitude de position en x et y et un intervalle pour l'incertitude de direction, ou cap (figure 3.3). On considère toutes les données des capteurs du robot comme étant des mesures incertaines qu'on en-capsule dans des intervalles. Ainsi la position, la direction et la détection d'objets sont représentées par des intervalles et des vecteurs d'intervalles, qui garantissent l'inclusion de la valeur réelle de ces variables.

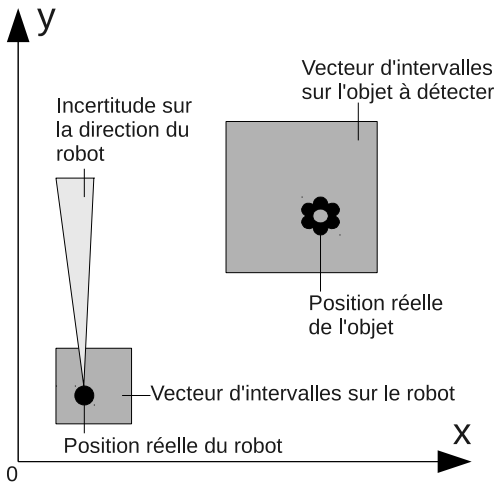

FIG. 3.3 – Approche ensembliste.

Le problème : plus un robot se déplace de façon autonome, plus il se perd dans son environnement. On admet que le robot connaît précisément sa position initiale. Il se déplace et ré-évalue sa position à l'instant $t+1$. Comme il a une incertitude sur son déplacement, il obtient une incertitude sur sa nouvelle position. Lors d'un déplacement à un temps $t+2$, cette incertitude augmentera et ainsi de suite, c'est ce qui est représenté sur la figure 3.4. Le robot, ne sachant plus précisément où il se trouve, ne peut plus localiser les objets identifiés dans l'environnement exploré.

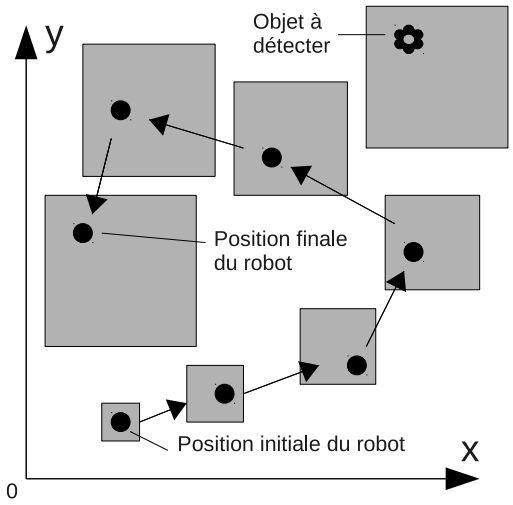

FIG. 3.4 – Plus un robot se déplace, moins il sait où il se trouve. On dit qu'il dérive.

3.3.1 La propagation/rétro-propagation de contraintes

Un moyen de contrer cette dérive est de pouvoir ré-évaluer précisément la position du robot de manière ponctuelle pendant l'exploration et de propager cette nouvelle information dans les relevés précédents (figure 3.5).

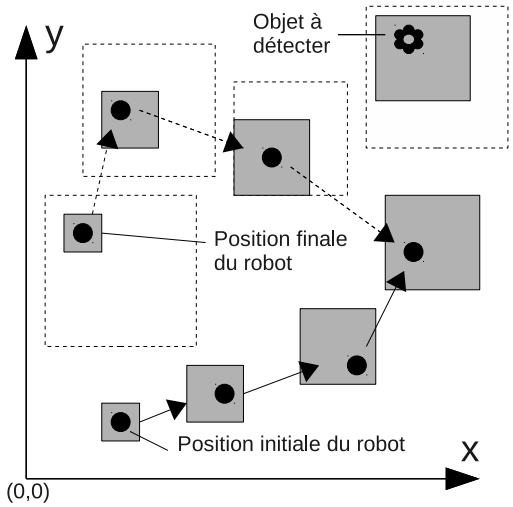

FIG. 3.5 – Propagation de contraintes inverse sur la position.

Toutes les positions du robot durant l'exploration sont reliées entre elles par le déplacement de ce dernier. On peut donc voir l'exploration d'un environnement comme un ensemble de variables et de contraintes. Une évaluation précise de la position à un instant t permet de rajouter une contrainte à notre CSP, et donc peut permettre de réduire les différents domaines des variables et ainsi améliorer

la précision de la connaissance des déplacements et donc de la localisation du robot.

Cette technique suppose qu'à chaque déplacement le robot soit capable d'évaluer son déplacement et donc sa position $t + 1$ en fonction de sa position t. Un moyen d'avoir ces informations est d'utiliser l'odométrie. Si cette technique permet d'avoir une évaluation du déplacement elle n'est pas très robuste et facilement sujette à des erreurs. Si le robot glisse, percute un élément de l'environnement les mesures des capteurs de l'odométrie seront faussées. Une alternative peut être d'utiliser un recalage visuel (seul [3] ou en complément de l'odométrie). Un télémètre laser permet d'avoir une image de l'environnement à un instant t, puis le robot se déplace et obtient une nouvelle image de l'environnement à l'instant $t + 1$. En recalant les images entre elles on peut évaluer le déplacement du robot. L'algorithme ICP[7], comme précisé plus haut, permet de faire ce genre de recalage.

[7]Iterative Closest Point.

4 Des SLAM Ensemblistes

Les travaux qui sont présentés ici ont eu pour objectifs de faire du recalage ensembliste. On détaillera deux pistes : l'utilisation de la propagation de contraintes et l'utilisation de la bissection d'intervalles.

Le problème : On a deux jeux de mesures (deux ensembles de points à deux dimensions) provenant du télémètre laser, $M = \{(x_{M_i}, y_{M_i})\}_i = \{X_{M_i}\}_i$ et $S = \{(x_{S_i}, y_{S_i})\}_i = \{X_{S_i}\}_i$. Le jeu M est réalisé à un instant t, et le jeu S à un instant $t+1$. Entre ces deux scans, le robot s'est déplacé d'une transformation $T = (t_x, t_y, r_z)^T$. L'objectif étant de retrouver T à l'aide de S et M. Le robot est aussi capable d'évaluer son déplacement, à l'aide de l'odométrie. On a donc une évaluation de T, $T_{odemetrie} = T_0$.

Pour simplifier les notations à venir on définit $T^h = \begin{pmatrix} cos(r_z) & -sin(r_z) & t_x \\ sin(r_z) & cos(r_z) & t_y \\ 0 & 0 & 1 \end{pmatrix}$ et $X^h_{A_b} = \begin{pmatrix} X_{A_b} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{A_b} \\ y_{A_b} \\ 1 \end{pmatrix}$, ce qui nous permet d'écrire : $X^h_{S_i} = T^h \times X^h_{M_i}$.

On définit aussi : $[T^h] = \begin{pmatrix} cos([r_z]) & -sin([r_z]) & [t_x] \\ sin([r_z]) & cos([r_z]) & [t_y] \\ 0 & 0 & 1 \end{pmatrix}$ et $[X^h_{A_b}] = \begin{pmatrix} [X_{A_b}] \\ 1 \end{pmatrix} = \begin{pmatrix} [x_{A_b}] \\ [y_{A_b}] \\ 1 \end{pmatrix}$.

4.1 Un algorithme utilisant la propagation/rétro-propagation de contraintes

Cette première sous-section n'est valide que pour des cas théoriques. On considère un jeu $M\{X_{M_1}, ..., X_{M_m}\}$ que l'on transforme à l'aide d'une transformation $T = (t_x, t_y, r_z)^T$ pour obtenir un jeu $S = \{X_{S_1}, ..., X_{S_n}\}$. On est donc en présence de deux jeux théoriques pour lesquels il existe un recalage qui permet une erreur (somme des distances au carré) nulle. On se donne aussi un domaine de recherche pour T nommé $[T_0] = ([t_{x_0}], [t_{y_0}], [r_{z_0}])^T$ tel que $T \in T_0$. L'objectif est de réduire $[T_0]$.

On a donc :

$$
\begin{aligned}
x_{S_i} &= cos(r_z)x_{M_i} - sin(r_z)y_{M_i} + t_x, \\
y_{S_i} &= sin(r_z)x_{M_i} + cos(r_z)y_{M_i} + t_y, \\
&\text{mais aussi :} \\
x_{M_i} &= cos(r_z)(x_{S_i} - t_x) + sin(r_z)(y_{S_i} - t_y), \\
y_{M_i} &= -sin(r_z)(x_{S_i} - t_x) + cos(r_z)(y_{S_i} - t_y).
\end{aligned}
$$

L'approche utilisée était de dire que l'on voulait recaler la scène (S) sur le modèle (M). On se retrouve donc avec le CSP suivant, à trois variables et $2 \times i$ contraintes :

$$
\begin{cases}
[t_x] = [t_{x_0}] \\
[t_y] = [t_{y_0}] \\
[r_z] = [r_{z_0}] \\
x_{M_i} = cos(r_z)(x_{S_i} - t_x) + sin(r_z)(y_{M_i} - t_y) \\
y_{M_i} = -sin(r_z)(x_{S_i} - t_x) + cos(r_z)(y_{M_i} - t_y)
\end{cases}.
$$

L'algorithme utilisé pour contracter $[T_0]$ est le suivant :

Algorithm 6 Recalage par propagation de contraintes

Require: M, S, $[T_0]$
1: $[T] \leftarrow [T_0]$
2: **for all** $X_{S_i} \in S$ **do**
3: $\quad [X^h_{S_i}] = ([T^h])^{-1} \times X^h_{S_i}$
4: $\quad [X_{S_i}] = contracter(M, [X_{S_i}])$
5: $\quad [T] = propager([X_{S_i}], X_{S_i}, [T])$
6: **end for**
7: **return** $[T]$.

Ligne 4 : *contracter* est une fonction qui permet de contracter sans utiliser de contracteurs. D'après l'énoncé du problème, $X_{M_i} \in [X_{S_i}]$. On peut donc réduire $[X_{S_i}]$ au plus petit intervalle contenant les X_{M_k} tel que $X_{M_k} \in [X_{S_i}]$ (figure 4.1). On réduit ainsi notre domaine et on peut propager cette réduction sur $[T]$.

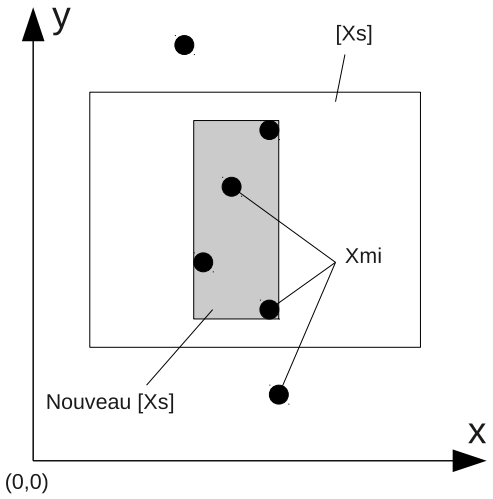

FIG. 4.1 – Résultat de la fonction *contracter* sur un exemple.

Ligne 5 : *propager* est une fonction qui permet de propager l'information obtenue par la contraction sur $[T]$. On décompose les contraintes en contraintes élémentaires, on évalue X_{M_i} par $[X_{S_i}]$ et on propage/rétro-propage les contraintes. On peut ainsi réduire $[T]$ et on le fait pour tous les points $X_{S_i} \in S$.

Pour améliorer la contraction on peut rajouter des contraintes. Plus on a de contraintes plus on est susceptible de réduire les domaines des variables t_x, t_y et r_z. Pour en rajouter on peut considérer des segments $(X_{S_i}; X_{S_{i+1}})$ au lieu de se limiter à contracter sur les points (X_{S_i}). Voici donc trois autres contraintes que l'on peut rajouter à notre CSP :

$$(x_{S_i} - x_{S_{i+1}})^2 + (y_{S_i} - y_{S_{i+1}})^2 = ([x_{S_i}] - [x_{S_{i+1}}])^2 + ([y_{S_i}] - [y_{S_{i+1}}])^2,$$
$$[x_{S_i}] - [x_{S_{i+1}}] = sin(r_z)(y_{S_i} - y_{S_{i+1}}) + cos(r_z)(x_{S_i} - x_{S_{i+1}}),$$
$$[y_{S_i}] - [y_{S_{i+1}}] = cos(r_z)(y_{S_i} - y_{S_{i+1}}) - sin(r_z)(x_{S_i} - x_{S_{i+1}}).$$

4.1.1 Résultats

Soit deux jeux M et S. S correspond à la transformation de M par $T = (t_x, t_y, t_z)^T = (-75, -80, 12)^T$, figure 4.1.1. Les valeurs numériques sont en *mm* pour les distances et *degrés* pour les angles.

On considère un domaine de recherche $[T_0] = ([t_{x_0}], [t_{y_0}], [r_{z_0}])^T = ([-100, 100], [-100, 100], [5, 20])^T$. Sur la figure 4.1.1 on peut voir l'évaluation de S par $[T_0]$. On remarque que notre initialisation est très imprécise.

La figure 4.1.1 présente le résultat de l'algorithme (obtenu en 3.50s). On trouve $[T] = ([-73.31, -72.66], [-82.50, -77.69], [11.03, 13.02])^T$. On a bien réussi à contracter T_0, en ayant tou-

jours $T \in [T]$.

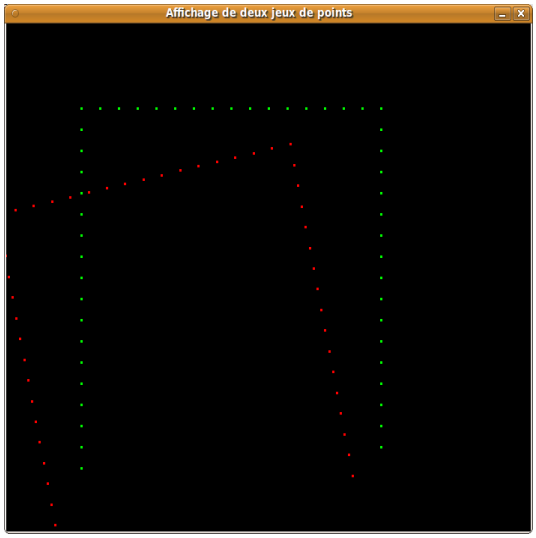

FIG. 4.2 – Les deux jeux de départ. Le jeu *model* (M, en vert) et le jeu *scene* (S, en rouge).La transformation entre les deux jeux est $t_x = -75$, $t_y = -80$ et $r_z = 12^\circ$.

FIG. 4.3 – Le jeu *model* est toujours en vert, en rouge on peut voir la transformation du jeu *scene* par $[T_0]$, avec $[t_{x_0}] = [-100, 100]$, $[t_{y_0}] = [-100, 100]$ et $[r_{z_0}] = [5^\circ, 20^\circ]$.

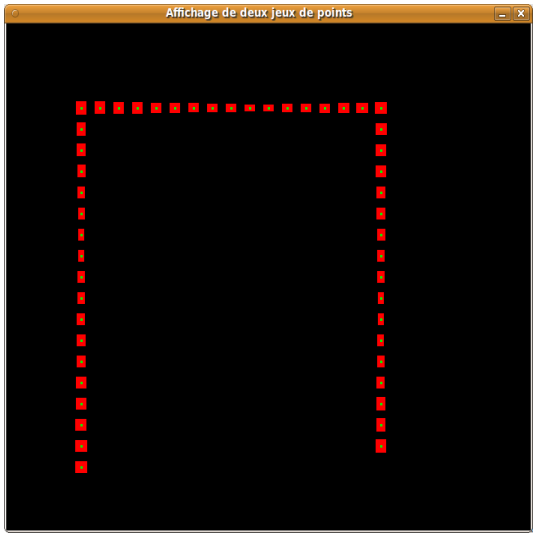

FIG. 4.4 – Résultat de la propagation de contraintes. Au final on obtient $[T]$, avec $[t_x] = [-77.31, -72.66]$, $[t_y] = [-82.50, -77.69]$ et $[r_z] = [11.03^\circ, 13.02^\circ]$, pour un temps d'exécution de 3,50s.

4.2 Un algorithme utilisant la bissection d'intervalles

Définition : La *bissection* d'intervalles est une opération qui consiste à découper un intervalle en deux parties égales.

L'approche de l'algorithme précédent était de contracter T_0 pour gagner en précision, ici on veut découper T_0 et ne garder que les parties qui sont susceptible contenir T.

Dans un contexte réel, il existe des points présents dans la scène S qui n'ont pas de représentant dans le modèle M. Il y a aussi le fait que les points présents dans les deux jeux soient entachés d'erreurs. Dans ce contexte on doit introduire la notion de Q-intersection.

4.2.1 La Q-Intersection

Dans un contexte réel on ne veut plus trouver une transformation $[T]$ qui permet de faire 100% de correspondances entre la scène S et le modèle M, mais Q correspondances. Soit N vecteurs d'intervalles, la Q-intersection (ou intersection relaxée) permet de renvoyer le(s) intervalle(s) représentant l'intersection de Q ou plus intervalles, figure 4.5 et 4.6.

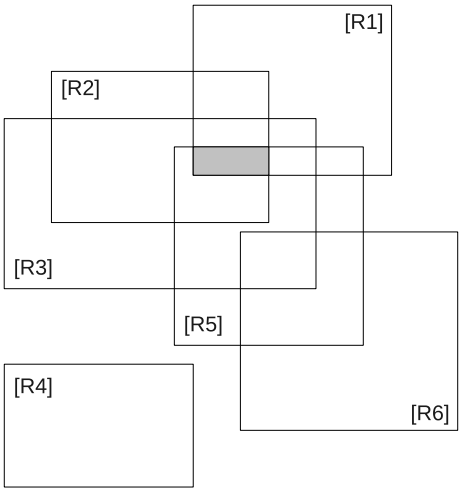

FIG. 4.5 – Soit 6 vecteurs d'intervalles $[R_1], ..., [R_6]$. La partie grise représente le résultat de la Q-intersection pour Q=4. L'intersection non relaxée de ces vecteurs est nulle.

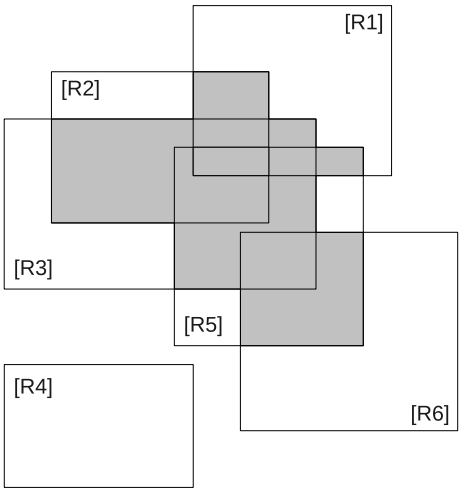

FIG. 4.6 – Résultat de la Q-intersection pour Q=2.

En relaxant l'intersection on peut ainsi prendre en compte des mesures aberrantes, la difficulté étant de trouver un compromis entre la tolérance et la précision du résultat. Dans les exemples ci dessus on peut considérer la Q-intersection avec $Q = 2$, on ne gagne pas beaucoup en ce qui concerne l'apport d'informations, mais si on considère l'intersection relaxée pour $Q = 6$ ou $Q = 5$ on obtient un ensemble vide.

4.2.2 Principe de l'algorithme

On a deux jeu de mesures M et S. On se donne un domaine de recherche pour la transformation $[T_0]$. On veut trouver un *pavé* aussi petit que possible qui contienne T, permettant de recaler la scène sur le modèle.

Remarque : on appelle *pavé* (*boîte*) un vecteur d'intervalles à trois dimensions (figure 2.3).

Pour se faire on part d'un pavé initial $[T_0]$. On décide d'un seuil à partir duquel un pavé est considéré unitaire (le plus petit pavé possible). L'idée c'est de bissecter le pavé initial et de trouver le(s) pavé(s) unitaire(s) qui permet(tent) le *meilleur* recalage.

Ici le *meilleur* recalage est considéré comme étant celui qui renvoie un $[T]$ qui maximise le nombre d'intersection entre M et S. Pour calculer le nombre d'intersections on fait :

Algorithm 7 Calculer le nombre d'intersections

Require: M, S, $[T]$
1: $nbInter = 0$
2: **for all** $X_{S_i} \in S$ **do**
3: $\quad [X^h_{S_i}] = ([T^h])^{-1} \times X^h_{S_i}$
4: $\quad$ **if** $[X_{S_i}] \cap M \neq \emptyset$ **then**
5: $\quad\quad nbInter = nbInter + 1$
6: $\quad$ **end if**
7: **end for**
8: **return** $nbInter$.

L'objectif du recalage est donc d'identifier tous les pavés unitaires qui permettent le maximum d'intersections. Ce seuil (le nombre maximum d'intersections) est calculé dynamiquement.

4.2.3 L'algorithme

L'algorithme 8 présente la méthode.

Algorithm 8 Recalage ensembliste 2

Require: M, S, $[T_0]$
1: $nbPavesValides = 0$
2: $nbInterSeuil \leftarrow getNbIntersection(M, S, [T_0])$
3: **while** $nbPavesValides = 0$ **do**
4: on met $[T_0]$ dans $pileRecherche$
5: **while** $pileRecherche$ n'est pas vide **do**
6: $[X] \leftarrow$ dernier élément de $pileRecherche$
7: $nb = getNbIntersection(M, S, [X])$
8: **if** $nb \geq nbInterSeuil$ **then**
9: **if** $[X]$ est unitaire **then**
10: On met $[X]$ dans $pilePavesValides$
11: $nbPavesValides = nbPavesValides + 1$
12: **else**
13: On bissecte $[X]$ en $[X_1]$ et $[X_2]$
14: On met $[X_1]$ et $[X_2]$ dans $pileRecherche$
15: **end if**
16: **end if**
17: **end while**
18: $nbInterSeuil = nbInterSeuil - 1$
19: **end while**
20: $[T] \leftarrow hull(pilePavesValides)$
21: **return** $T[X]$.

Ligne 2 : $getNbIntersection(M,S,[T])$ est une fonction qui permet de calculer le nombre d'intersections entre les $X_{M_i} \in M$ et les $[X_{S_i}]$ avec $[X_{S_i}^h] = [T^h] \times X_{S_i}^h$, $X_{S_i} \in S$ (algorithme 7).

Ligne 20 : *hull* est une fonction qui permet de récupérer la hull[1] de tous les pavés valides : Il se peut qu'il n'y ai pas 1 pavé mais plusieurs pavés unitaires qui permettent un nombre d'intersections suffisant. Le résultat de l'algorithme est donc la hull de tous les pavés trouvés afin de ne pas perdre de solutions.

4.2.4 Résultats

Le recalage qui suit a été obtenu en 17.2 secondes (pour 519 mesures dans M et 85 mesures dans S). Le pavé de départ était le suivant : $[t_{x_0}] = [-100, 0]$, $[t_{y_0}] = [800, 1100]$ et $[r_{z_0}] = [6\,^\circ, 86\,^\circ]$. La solution donnée par l'algorithme est la suivante : $[t_x] = [-66.40, -65.63]$, $[t_y] = [931.83, 932.43]$ et $[r_z] = [49.43\,^\circ, 49.76\,^\circ]$ (les translations sont en mm, les données proviennent d'un URG-04LX).

[1] le plus petit pavé qui contient tous les pavés valides.

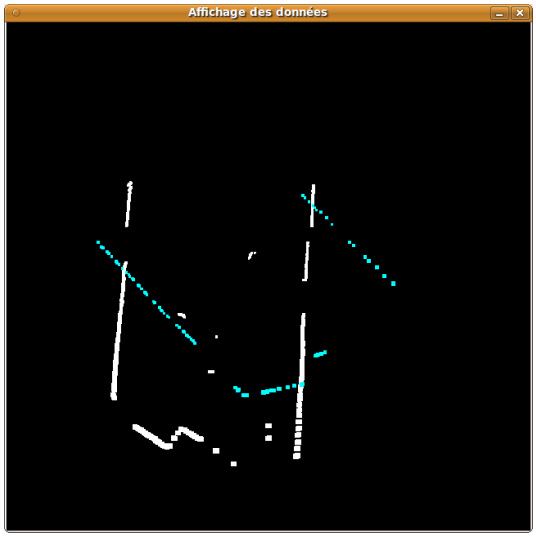

FIG. 4.7 – Avant l'algorithme de recalage. Le jeu blanc correspond au modèle et le jeu bleu à la scène.

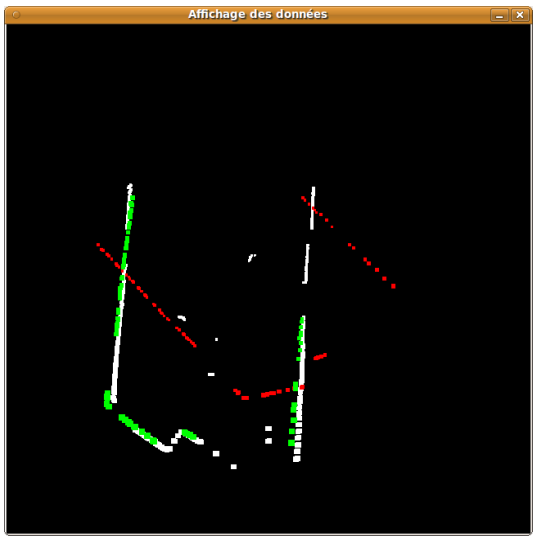

FIG. 4.8 – Après l'algorithme de recalage. Le jeu blanc correspond toujours au modèle, le jeu rouge à la scène avant recalage, et le jeu vert à la scène après recalage.

5 Conclusion

Dans ce rapport on a présenté les principales notions théoriques qui ont été abordées tout au long de ce stage, ainsi que les résultats des travaux de recalage de jeux de mesures.

On a présenté l'analyse par intervalles et la problématique SLAM, les deux notions principales pour ces travaux. Ensuite on s'est intéressé au problème de localisation et plus précisément de recalage de jeux de mesures afin d'évaluer le déplacement de robots autonomes.

Les résultats obtenus sont très encourageants pour la suite des travaux qu'il sera possible de faire en considérant l'approche ensembliste pour la localisation et la cartographie.

Les algorithmes type ICP sont très performants localement. Pour une initialisation proche de la valeur réelle, l'analyse par intervalle ne peut pas rivaliser en terme de temps de calcul. Par contre pour les cas où l'initialisation est loin de la valeur réelle les algorithmes ICP ne sont plus utilisables.

Pour la suite des travaux on travaillera dans un premier temps à améliorer les temps de calcul pour les deux algorithmes. Deux pistes :

- $\rightsquigarrow$ travailler la succession des contraintes pour la propagation et rétro-propagation afin d'itérer le moins possible pour un résultat optimal.
- $\rightsquigarrow$ réfléchir à un moyen de réduire le temps de calcul pour la récupération de la hull (compléxité de $o(n^2)$ actuellement).

Une autre idée serait de coupler l'analyse par intervalles avec une méthode ICP. On peut ainsi envisager un ICP qui permettrait un recalage global et non plus local.

Il faudra aussi adapter l'algorithme utilisant la propagation/rétro-propagation de contraintes pour qu'il puisse être appliqué sur des jeux de mesures réels (utilisation de la Q-intersection).

L'objectif final serait d'avoir un algorithme de recalage complètement ensembliste qui permettrait un recalage de jeux de mesures sans aucune connaissance de $[T_0]$: un recalage global ensembliste. On serait ainsi en mesure de résoudre le problème de kidnapping en robotique par exemple.

Bibliographie

[1] *Introduction à l'algorithmique*, chapter I et III. Dunod, 1994.

[2] *Applied Interval Analysis, With Examples in Parameter and State Estimation, Robust Control and Robotics*, chapter 2.Interval Analysis. Springer, 2001.

[3] H. Yoshitaka K. Hirohiko O. Akihisa and Y. Shin'ichi. Mobile robot localization and mapping by scan matching using laser reflection intensity of the sokuiki sensor. In *Proc. 32nd Annual Conf. IEEE Industrial Electronics*. 2006.

[4] Felix Calderon Carlos Lara, Leonardo Romero. A robust iterative closest point algorithm with augmented features. In *MICAI 2008 : advances in artificial intelligence*. 2008.

[5] I. D. Coope. Technical note - circle fitting by linear and nonlinear least squares. *Journal Of Optimisation THeory And Applications*, 1993.

[6] L. Jaulin G. Chabert. propagation de contraintes sur les intervalles pour la planification d'-expérience en slam. Submitted to *RIA*.

[7] Luc JAULIN Jan SLIWKA, Fabrice LE BARS. Calcul ensembliste pour la localisation et la cartographie robustes. 2009.

[8] Luc Jaulin. A nonlinear set membership approach for the localization and map building of underwater robots. *Trans. Rob.*, 25(1) :88–98, 2009.

[9] Raphael Ari Finkel Jerome H. Friedman, Jon Louis Bentley. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 1977.

[10] Ziv Yaniv. Rigid registration the iterative closest point algorithm. http ://i-siswiki.georgetown.edu/zivy/writtenMaterial/icp.pdf.

A Des structures de données

Pour améliorer la recherche du plus proche voisin, et plus généralement la recherche de données dans un jeu, il existe des structures de données appelées des *arbres*. Un arbre est un graphe non orienté, connexe et acyclique [1]. Un arbre est composé d'une *racine*, de *nœuds* et de *feuilles* (figure A.1). Au sein d'un arbre, la *profondeur* d'un nœud représente la longueur du chemin entre ce nœud et la racine de l'arbre. La plus grande profondeur dans un arbre correspond à la *hauteur* de l'arbre.

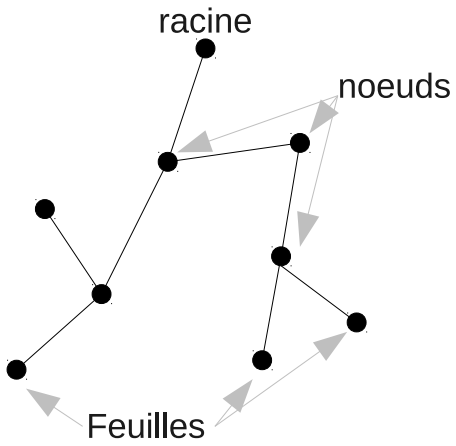

FIG. A.1 – Un arbre.

A.1 les arbres binaires

Cette sous-sous-section est inspirée de [1]. Un arbre binaire est un arbre construit de façon aléatoire, dans le cas optimal, pour lequel chaque nœud représente une donnée. Chaque nœud a au plus deux *fils* (deux nœuds en dessous de lui), son fils droit lui est supérieur et son fils gauche inférieur, figure A.2. Un arbre binaire permet certaines opérations comme *RECHERCHER*, *INSERER*, *SUPPRIMER*... Pour chaque opération les arbres binaires permettent une complexité de $O(lg\ n)$. La création d'un arbre binaire a aussi une complexité de $O(lg\ n)$.

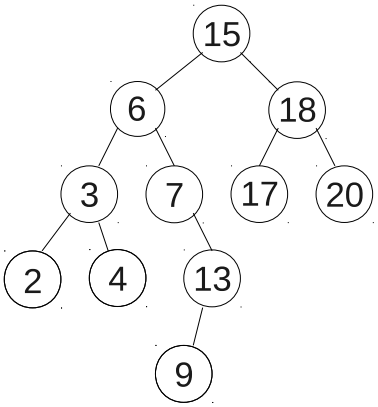

FIG. A.2 – Un arbre binaire.

Exemple : prenons l'arbre de la figure A.2. On recherche la clé[1] 13 dans ce dernier (Algorithme 9). Pour la trouver on suit le chemin suivant : 15(*racine*) $\rightarrow$ 6 $\rightarrow$ 7 $\rightarrow$ 13. Le maximum est trouvé en prenant toujours le nœud de droite, et le minimum en prenant le nœud de gauche (Algorithme 10).

Algorithm 9 Arbre binaire - Rechercher - Itératif

Require: x un nœud et k une donnée à trouver

```
1: while x ≠ NULL et k ≠ cle[x] do
2:     if k < cle[x] then
3:         x ← gauche[x]
4:     else
5:         x ← droit[x]
6:     end if
7: end while
8: return x.
```

Algorithm 10 Arbre binaire - Minimum

Require: x un nœud (la racine de l'arbre)

```
1: while gauche[x] ≠ NULL do
2:     x ← gauche[x]
3: end while
4: return x.
```

[1]La clé d'un nœud correspond à la donnée qu'il représente

A.2 les kd-trees

Les kd-trees[2] [9] sont une généralisation des arbres binaires pour des données à k-dimensions. La racine de l'arbre représente toutes les données et chaque nœud représente une partition de ces données. La complexité pour la construction d'un tel arbre est de $kN(log\ N)$, quand à la recherche du plus proche voisin la complexité est de $O(log\ N)$.

Ces arbres sont utilisés dans certains algorithmes ICP pour améliorer le temps d'exécution du *Match*. Les jeux de mesures sont représentés comme des données à deux dimensions (x, y).

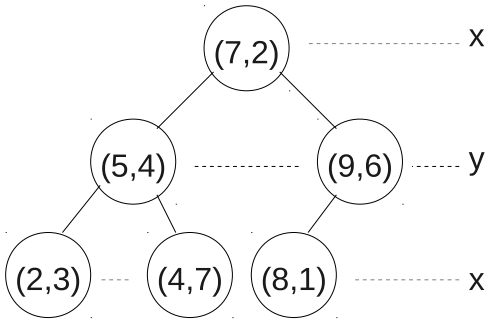

FIG. A.3 – Un kd-tree, à deux dimensions ($k = 2$).

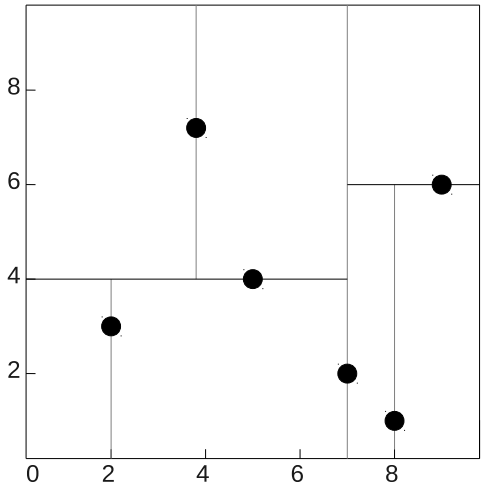

FIG. A.4 – Résultat du kd-tree de la figure A.3.

Le principe de construction du kd-tree est le même que pour l'arbre binaire. Chaque nœud a au plus deux fils, un fils droit qui est plus grand et un fils gauche qui est plus petit. La particularité

[2]Arbres à k-dimensions

c'est que pour comparer les clés des nœuds on se place alternativement dans chacune des k-dimensions. Dans la figure A.3, on *coupe* dans un premier temps selon les x, puis selon les y, et ainsi de suite. Ce qui nous donne le découpage de l'espace à 2 dimensions visible sur la figure A.4.

La recherche du plus proche voisin est un peu plus complexe que pour un arbre binaire. Elle se fait en deux étapes. Dans un premier temps, on recherche un *bon candidat*. Comme pour l'arbre binaire on descend dans l'arbre jusqu'à obtenir une feuille qui représente un bon candidat pour le plus proche voisin, mais pas forcément le meilleur. On remonte alors dans l'arbre pour chercher si on ne trouve pas une meilleur solution.

A.3 Les interval trees

Les interval trees[3] sont des structures de données qui permettent de retrouver efficacement la (les) intersection(s) entre les dites données et un point ou un intervalle. Il existe plusieurs façons de construire un arbre d'intervalles (Segment tree, Range tree...), la structure de données détaillée ici est celle présentée dans [1]. A la façon des arbres binaires, l'arbre d'intervalle a une racine, des nœuds et des feuilles. Chaque nœud a au plus deux fils, un fils supérieur à droite et inférieur à gauche. Les intervalles sont comparés en ne tenant compte que de leurs bornes inférieures. En plus des intervalles chaque nœud contient la valeur maximale des bornes supérieures du sous-arbres enraciné en ce nœud (le sous arbre pour lequel le nœud est la racine). La figure A.5 représente un exemple d'une telle structure.

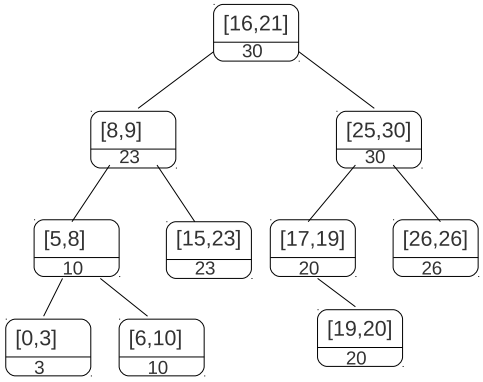

FIG. A.5 – Un arbre d'intervalles.

Pour ce qui est de la complexité, la recherche d'un intervalle qui intersecte est de $0(lg\ n)$, l'algorithme 11 permet de trouver un tel intervalle.

[3] Arbres d'intervalles

Algorithm 11 Arbre d'intervalles - Rechercher Intervalle

Require: T un arbre et i un intervalle

1: $x \leftarrow racine[T]$
2: **while** $x \neq NULL$ et i ne se superpose pas avec $interval[x]$ **do**
3: **if** $gauche[x] \neq NULL$ et $max[gauche[x]] \geq inf[i]$ **then**
4: $x \leftarrow gauche[x]$
5: **else**
6: $x \leftarrow droit[x]$
7: **end if**
8: **end while**
9: **return** $interval[x]$.

B Optimisation non linéaire

Dans le chapitre précédent on a présenté comment on pouvait organiser les données pour rendre la partie *Match* d'ICP plus performante. Maintenant nous allons présenter la partie *Optimisation*. Quelles sont les techniques que l'on peut utiliser pour trouver la meilleur transformation au sens des moindres carrés ?

B.1 Gauss-Newton

La méthode de Gauss-Newton[1] est une méthode itérative qui permet de résoudre des problèmes de moindres carrés non linéaires. Dans le cas d'un ICP on a deux jeux *"matchés"* (pour tout $X_{S_i} \in S$ on associe un point $X_{M_j} \in M$) : $M = \{X_{M_1}, ..., X_{M_m}\}$ et $S = \{X_{S_1}, ..., X_{S_m}\}$. L'objectif étant de recaler la scène S sur le modèle M. Par recaler on entend minimiser la somme des distances résiduelles au carré entre les points $X_{S_i} \in S$ et les points $X_{M_i} \in M$ associés.

$$\underset{T}{Min} \sum_{i=1}^{m} \{distance(X_{M_i}, X'_{S_i})\}^2,$$

avec

$$distance(X_{M_i}, X'_{S_i}) : \text{la distance euclidienne entre } X_{M_i} \text{ et } X'_{S_i},$$
$$X'^h_{S_i} = \begin{pmatrix} X'_{S_i} \\ 1 \end{pmatrix} = (T^h)^{-1} \times X^h_{S_i}.$$

Ce qui nous donne :

$$\begin{aligned} x'_{S_i} &= cos(r_z)x_{S_i} - sin(r_z)y_{S_i} + t_x, \\ y'_{S_i} &= sin(r_z)x_{S_i} + cos(r_z)y_{S_i} + t_y. \end{aligned}$$

Si on réécrit le problème à minimiser on obtient :

$$\underset{t_x,t_y,r_z}{Min} \sum_{i=1}^{m} \{(x_{M_i} - cos(r_z)x_{S_i} + sin(r_z)y_{S_i} - t_x)^2 + (y_{M_i} - sin(r_z)x_{S_i} - cos(r_z)y_{S_i} - t_y)^2\}.$$

On est donc en présence d'un problème non linéaire, qui peut être résolu par un algorithme de Gauss-Newton. Ce dernier permet de trouver, si l'initialisation n'est pas trop loin de la solution, le minimum de la somme des carrés :

[1] Carl Friedrich Gauss et Isaac Newton.

$$S(\beta) = \sum_{i=1}^{m} r_i^2(\beta),$$

avec m fonctions $r_1, ..., r_m$ de n variables $\beta = (\beta_1, ..., \beta_n)$, avec $m > n$. On initialise l'algorithme avec une approximation de β : β^0, et on construit un nouveau β^i de façon itérative :

$$\beta^{s+1} = \beta^s + \triangle,$$

avec $\triangle = -(J_r^T J_r)^{-1} J_r^T r$, J_r étant la Jacobienne de r.

Dans notre cas

$$\beta = T = (t_x, t_y, r_z)^T \text{ et}$$
$$r_i(T) = \sqrt{(x_{M_i} - cos(r_z)x_{S_i} + sin(r_z)y_{S_i} - t_x)^2 + (y_{M_i} - sin(r_z)x_{S_i} - cos(r_z)y_{S_i} - t_y)^2},$$
$$r = \begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix},$$
$$J_r = \begin{pmatrix} \frac{\partial r_1}{\partial t_x} & \frac{\partial r_1}{\partial t_y} & \frac{\partial r_1}{\partial r_z} \\ \vdots & \vdots & \vdots \\ \frac{\partial r_i}{\partial t_x} & \frac{\partial r_i}{\partial t_y} & \frac{\partial r_i}{\partial r_z} \end{pmatrix}.$$

B.2 Levenberg-Marquardt

Comme pour l'algorithme précédent, Levenberg-Marquardt est une méthode qui permet de minimiser :

$$S(\beta) = \sum_{i=1}^{m} r_i^2(\beta).$$

A mi-chemin entre la descente de gradient et l'algorithme de Gauss-Newton, elle permet de converger plus rapidement vers une solution que Gauss-Newton et d'être plus stable que la descente de gradient. C'est une méthode itérative qui évalue une nouvelle solution en fonction de la précédente de la façon suivante :

$$\beta^{s+1} = \beta^s - (J_r^T J_r + \lambda diag(J_r^T J_r))^{-1} J_r^T r,$$

λ étant un coefficient et $diag(J_r^T J_r)$ représentant la matrice diagonale contenue dans $J_r^T J_r$.

Dans notre cas :

$$\beta = T = (t_x, t_y, r_z)^T,$$
$$r_i(T) = \sqrt{(x_{M_i} - cos(r_z)x_{S_i} + sin(r_z)y_{S_i} - t_x)^2 + (y_{M_i} - sin(r_z)x_{S_i} - cos(r_z)y_{S_i} - t_y)^2},$$
$$r = \begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix},$$
$$J_r = \begin{pmatrix} \frac{\partial r_1}{\partial t_x} & \frac{\partial r_1}{\partial t_y} & \frac{\partial r_1}{\partial r_z} \\ \vdots & \vdots & \vdots \\ \frac{\partial r_i}{\partial t_x} & \frac{\partial r_i}{\partial t_y} & \frac{\partial r_i}{\partial r_z} \end{pmatrix}.$$

On remarque bien que pour un λ petit on se rapproche de la méthode de Gauss-Newton et pour un λ grand on est plus proche d'une descente de gradient.

Le coté délicat de cet algorithme est de trouver le meilleur moyen d'initialiser et de ré-évaluer le coefficient λ à chaque itération, afin de tendre le plus rapidement vers la solution et de rester le plus stable possible.

Algorithm 12 Levenberg - Marquardt

Require: S, M ($matché$ à S), T_0
$\epsilon_{init} \leftarrow 0, T \leftarrow T_0, T_{tmp} \leftarrow T$
for all $X_{S_i} \in S$ **do**
  $\epsilon_{init} \leftarrow \epsilon_{init} + (r_i(T))^2$
end for
$\epsilon_{est} \leftarrow \epsilon_{init}$
while $nbIteration < seuil_{nbIt}$ **or** $\epsilon_{est} > seuil_{\epsilon}$ **do**
  $T_{tmp} \leftarrow T_{tmp} - (J_r^T J_r + \lambda diag(J_r^T J_r))^{-1} J_r^T r$
  $\epsilon_{est} = 0$
  for all $X_{S_i} \in S$ **do**
    $\epsilon_{est} = \epsilon_{est} + (r_i(T_{tmp}))^2$
  end for
  if $\epsilon_{est} < \epsilon_{init}$ **then**
    $\epsilon_{init} = \epsilon_{est}$
    $T \leftarrow T_{tmp}$
    On diminue λ
  else
    $T_{tmp} \leftarrow T$
    On augmente λ
  end if
end while
return T.

Titre : **Cartographie par des Approches Ensemblistes**

Mots Clés : Simultaneous Localization and Mapping, Analyse par Intervalles, Propagation de Contraintes, Iteratives Closest Point, Structures de données, Optimisation.

Résumé : Durant ce stage il a été question de travailler sur une approche ensembliste pour un problème de type SLAM. Ce rapport rapporte le résultat les différentes recherches qui ont pu être effectuées afin d'identifier les problématiques ainsi que les solutions existantes. En plus des différentes présentations des outils étudiés, comme l'analyse par intervalles et le SLAM ensembliste, ce document présente deux solutions originales ensemblistes pour le recalage de données télémétriques à deux dimensions.

Title : **Map Making using Interval Approach**

Keywords : Simultaneous Localization and Mapping, Interval Analysis, Constraint Propagation, Iteratives Closest Point, Data stuctures, Optimisation.

Abstract : This placement was about working on a SLAM problem using interval analysis. This report shows the results of the several researches which were achieved in order to target the problematic and the existing solutions. In addition to the presentations of the tools studied, like interval analysis and SLAM, this document present two new approaches to the two dimensional telemetric data matching using interval analysis.

Laboratoire :

62, avenue Notre Dame du Lac
49000 Angers