

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : STIC / Automatique
École Doctorale SICMA

présentée par

Fabrice Le Bars

Préparée à l'ENSTA Bretagne (ex ENSIETA), Equipe OSM, Pôle STIC

Analyse par intervalles pour la localisation et la cartographie simultanées; Application à la robotique sous-marine

Thèse soutenue le 17 octobre 2011
devant le jury composé de :

Luc JAULIN
Professeur des Universités, ENSTA Bretagne / *directeur de thèse*

Philippe BONNIFAIT
Professeur des Universités, UTC / *rapporteur*

Patrick RIVES
Directeur de Recherche, INRIA Sophia-Antipolis / *rapporteur*

Pierre DE LOOR
Professeur des Universités, ENIB / *président du jury*

Eva CRUCK
Responsable-adjoint I2R, DGA/DS/MRIS / *examinateur*

Alain BERTHOLOM
Responsable d'Essais, DGA/DET/GESMA / *examinateur*

Analyse par intervalles pour la localisation et la cartographie simultanées ; Application à la robotique sous-marine

Fabrice LE BARS

23 novembre 2011

Table des matières

1 Introduction **17**

1.1 Contexte . . . 17

1.2 Objectifs, hypothèses, contraintes, restrictions et contributions . . . 18

1.3 Annonce du plan . . . 19

2 Robotique et sous-marins **21**

2.1 Présentation . . . 21

2.2 Exemples de robots sous-marins autonomes . . . 26

2.2.1 Le *Redermor* et la *Daurade*, AUVs du GESMA munis d'un sonar latéral . . . 27

2.2.2 *SAUC'ISSE*, AUV de l'ENSTA Bretagne muni d'un sonar rotatif . . . 32

2.3 Exemples de robots terrestres, sortes d'équivalents terrestres des sous-marins . . . 42

2.3.1 Les JOGs de l'ENSTA Bretagne, munis de télémètres sonar et infrarouges . . . 42

2.3.2 Les robots BOURRICOTs de l'ENSTA Bretagne, munis d'un télémètre laser rotatif . . . 43

2.4 Conclusion du chapitre . . . 49

3 Calcul par intervalles **51**

3.1 Présentation . . . 51

3.2 Intervalles . . . 53

3.3 Généralisation des intervalles . . . 56

3.3.1 Principe . . . 56

3.3.2 Intervalles vectoriels . . . 56

3.3.3 Tubes . . . 57

3.4 Algorithmes intervalles . . . 59
3.4.1 Contraction et propagation . . . 59
3.4.2 Algorithmes de bissection . . . 63
3.4.3 Combinaison de contraction, propagation et bissection . . . 64
3.5 Bibliothèques et outils pour faire du calcul par intervalles . . . 65
3.6 Conclusion du chapitre . . . 68

4 SLAM **73**
4.1 Principe . . . 74
4.2 Méthodes existantes . . . 75
4.3 SLAM par intervalles . . . 76
4.3.1 Principe . . . 76
4.3.2 Application dans les cas des sous-marins *Daurade* et *Redermor* . . . 77
4.4 Validation de différentes méthodes de résolution du problème de SLAM . . . 98
4.5 Conclusion du chapitre . . . 108

5 Estimation d'état et SLAM avec données fugaces **115**
5.1 Principe . . . 115
5.2 Formalisation et résolution du problème des données fugaces avec les tubes . . . 117
5.2.1 Problème des données fugaces . . . 117
5.2.2 Résolution du problème d'estimation d'état avec données fugaces . . . 120
5.3 Extension au SLAM sous-marin . . . 134
5.4 Conclusion du chapitre . . . 138

6 Conclusion et perspectives **139**

Articles, conférences et rapports **143**

Résumé des contributions principales **147**

Activités annexes **149**

Index **149**

Bibliographie **152**

Table des figures

2.1 Position $\mathbf{p} = (x, y, z)$ et orientation φ, θ, ψ (angles de roulis, tangage et lacet) d'un robot sous-marin auquel on associe le repère mobile $(\mathbf{p}, \mathbf{i}_r, \mathbf{j}_r, \mathbf{k}_r)$, par rapport au repère de référence $(\mathbf{O}, \mathbf{i}, \mathbf{j}, \mathbf{k})$ où $\mathbf{O}$ est la position du robot lors de sa mise à l'eau, le vecteur $\mathbf{i}$ indique le Nord, $\mathbf{j}$ indique l'Est et $\mathbf{k}$ est orienté vers le centre de la Terre (l'axe des z est souvent tourné vers le bas pour un sous-marin, plutôt vers le haut pour un robot aérien) avec mesure d'angle α_m, β_m et distance d_m à un amer $\mathbf{m} = (x_m, y_m, z_m)$. 22

2.2 Position et orientation du repère mobile par rapport au repère de référence. 23

2.3 Position et orientation x, y, θ d'un robot char se déplaçant en 2D (ici θ correspond plutôt au ψ du robot 3D) dans le plan dans une pièce rectangulaire coïncidant avec le repère de référence avec mesure d'angle α_m, β_m et distance d_m à un amer $\mathbf{m} = (x_m, y_m, z_m)$ (contexte de SLAM 2D/3D). 23

2.4 Classification possible de capteurs utilisés couramment sur des robots mobiles. 27

2.5 La *Daurade* et le *Redermor* (photos DGA/GESMA). 28

2.6 Sous-marin muni d'un sonar latéral bâbord et tribord : au fur et à mesure que le sous-marin avance, on peut générer une image sonar qui correspond à une vue du dessus du fond marin. 29

2.7 Vue arrière du sous-marin et capture d'écran des résultats correspondants, fournis par le logiciel SonarPro (signal brut du sonar montrant l'intensité de l'écho reçu en fonction de la distance et image). 30

2.8 Une mine sous-marine sur une image de sonar latéral tribord, et des rides de sable en dessous. 31

2.9 Détection d'une mine sous-marine par le sonar latéral tribord d'un sous-marin. 32

2.10 *SAUC'ISSE* dans une piscine de l'IFREMER (Brest, France). 33

2.11 Pipeline jaune au fond de la zone de compétition. 33

2.12 Bouée orange dans la zone de compétition. 34

2.13 Missions du concours SAUC-E 2010, à La Spezia en Italie. 34

2.14 Organisation intérieure de *SAUC'ISSE*. 35

2.15 Capteurs et actionneurs de *SAUC'ISSE*. 36

2.16 Image sonar de la zone de compétition pendant SAUC-E 2009. 38

2.17 Connexion au bureau à distance. 38

2.18 Localisation et détection. 39

2.19 Principe du contrôle du robot sous-marin. 40

2.20 Exemple de script de mission autonome. 41

2.21 Détection d'objets selon la couleur en prenant en compte l'absorption des couleurs dans l'eau selon la distance. 41

2.22 Robots JOGs. 43

2.23 Capteurs et actionneurs des JOGs. 44

2.24 BOURRICOT, la meute de robots de l'ENSTA Bretagne pour le concours CAROTTE 2010. . 44

2.25 Meute de robots cartographiant un bâtiment. 45

2.26 Capteurs et actionneurs d'un robot BOURRICOT. 46

2.27 Image fournie par un télémètre laser rotatif. 47

2.28 Agencement des LEDs sur les robots, pour pouvoir déterminer la distance et l'angle sous lequel on voit un robot, ainsi que son orientation. 48

3.1 Un pavé de $\mathbb{R}^2$. 56

3.2 Un tube $[x](t)$, avec un pas de temps δ. $[x](k)$ est sa $k^{ième}$ tranche. 57

3.3 Tube (formé ici par la superposition de pavés) contenant la trajectoire d'un robot char (dans le plan $(\mathbf{O}, \mathbf{i}, \mathbf{j})$). La trajectoire noire $\mathbf{x}(t)$ est la trajectoire réelle, générée par une simulation et l'enveloppe grise $[\mathbf{x}](t)$ est le tube obtenu après traitement des données et équations. On voit que la trajectoire réelle ne sort jamais du tube, comme les méthodes intervalles nous le garantissent. 58

3.4 Un tube permet d'obtenir des informations intéressantes sur la trajectoire qu'il représente. On peut par exemple voir ici que la fonction $x(t)$ s'annule au moins 4 fois. 58

3.5 Une contrainte de croissance sur $x(t)$ permet d'éliminer les parties grises du tube initial. . . . 61

3.6 Evolution de l'erreur d'estimation de l'orientation du robot en fonction des mesures indiquées par sa centrale inertielle ($\theta_{m,i}$ en abscisses et $\theta_i - \theta_{m,i} - \pi$ en ordonnées). 65

3.7 CSP rentré dans Proj2D. 66

3.8 Ensemble solution dans l'espace des paramètres (p_1, p_2) trouvé par Proj2D (le temps de calcul est inférieur à la seconde sur un ordinateur à base de processeur Intel Core 2 Duo). . . 67

3.9 Superposition du modèle théorique (en prenant p_1, p_2 inconnus avec PROJ2D) et de la courbe des points de mesures. 68

3.10 Script en langage QUIMPER utilisé avec QSOLVE. 69

3.11 Superposition du modèle théorique (en prenant p_1, p_2, p_3 inconnus avec QSOLVE) et de la courbe des points de mesures. 70

3.12 Commande utilisée avec QSOLVE. 70

3.13 Fichier de résultats généré par QSOLVE. Chaque ligne correspond à des intervalles $\left[p_1^-, p_1^+\right]$, $\left[p_2^-, p_2^+\right]$, $\left[p_3^-, p_3^+\right]$ solutions du problème. 71

4.1 En haut, le *Redermor* et sa trajectoire effectuée (vue du dessous), avec la position des amers (carrés noirs). En bas, la *Daurade* lors d'une expérience dans la baie de Douarnenez (photos DGA/GESMA). 78

4.2 Capture d'écran de GESMI, programme qui calcule et affiche la position des mines, l'enveloppe de la trajectoire, l'erreur en position d'un sous-marin lors d'une expérience de quadrillage de zone inconnue, et qui aide à identifier les mines parmi les différentes détections. . . 81

4.3 Fenêtre de GESMI montrant les données de navigation à un instant t donné. 82

4.4 Fenêtre de GESMI permettant de voir et modifier les données relatives aux détections d'amers. 82

4.5 Fenêtre de GESMI permettant de voir graphiquement la trajectoire du sous-marin (en 2D et vue du dessous) et les positions des amers. Les pavés enveloppant la position du sous-marin à chaque instant t sont dessinés en vert, et leur centre est indiqué en rose. Le pavé correspondant à la position courante du sous-marin (que l'on peut faire varier en bougeant le curseur de la souris sur les différentes fenêtres de GESMI) est dessiné en bordeau. Les pavés enveloppant les positions des amers sont dessinés en bleu. 83

4.6 Fenêtre de GESMI permettant de voir les positions des amers déterminées par le programme, sous forme d'intervalles. 84

4.7 Fenêtre de GESMI permettant de voir l'évolution de l'erreur (axe vertical en m) d'estimation des positions x, y, z du sous-marin (x en rouge, y, en vert et z en bleu) en fonction du temps (axe horizontal en s). 84

4.8 Fenêtre de GESMI montrant l'image sonar tribord et les zones où il devrait y avoir un amer. Les enveloppes vertes, bleues et rouges représentent la distance robot-amer (axe horizontal) en fonction du temps (axe vertical) : en rouge l'amer est plutôt devant le sous-marin et en bleu plutôt derrière. C'est le fait qu'à un moment l'amer était devant et qu'à un autre moment il se trouve derrière qui nous permet (par le théorème des valeurs intermédiaires) de déterminer qu'à un instant t, le sonar aurait dû voir l'amer (zone entourée de vert). C'est aussi à cet instant que la distance robot-amer est localement minimale. Sur cette image, on distingue deux mines : une en haut à droite, qui a été détectée par l'opérateur humain (les mines avec un petit carré rouge sont celles que l'utilisateur a détecté lui-même) et une autre en-dessous (non détectée par l'opérateur, mais que GESMI met en évidence à l'intérieur de l'enveloppe verte). On voit aussi un câble les reliant. 85

4.9 Capture d'écran d'une image sonar (bâbord et tribord) affichée par SonarPro. L'opérateur humain doit faire défiler l'image sonar de bas en haut pour parcourir toutes les données dans le temps. On voit à droite (côté tribord) une forme correspondant à une épave. 88

4.10 a) Enveloppe de la trajectoire après une propagation *forward* (temps de calcul inférieur à 1 s sur un ordinateur à base de processeur Intel Core 2 Duo), dessinée en gris. Le centre des pavés correspondant à la position du sous-marin à chaque instant est dessiné en gris foncé. Le repère est $[-200, 1000]$ m $\times$ $[-200, 1000]$ m (dans le plan (x, y)). b) Erreurs en position en x (noir), y (gris foncé), z (gris clair) en fonction du temps après une propagation *forward*. Le repère est $[0, 18445.75]$ s $\times$ $[0, 200]$ m. c) Enveloppe de la trajectoire après une propagation *forward-backward* (temps de calcul inférieur à 2 s). d) Erreurs en position après une propagation *forward-backward*. e) Enveloppe de la trajectoire après 10 contractions, en prenant en compte les amers détectés sur l'image sonar (temps de calcul inférieur à 10 s). Les pavés contenant les positions des amers sont dessinés en noir. f) Erreurs en position après 10 contractions, en prenant en compte les amers détectés sur l'image sonar. 90

4.11 Image sonar reconstituée schématiquement montrant un amer détecté et la prédiction de la distance robot-amer à un autre instant obtenue grâce à la prise en compte des données de navigation du sous-marin. 91

4.12 Résultats d'une propagation *forward* sur les données de navigation du *Redermor* avec GESMI. Dans la fenêtre *Position error* (voir figure 4.7 pour plus de détails sur ce que représente cette fenêtre), on constate que l'erreur d'estimation de position augmente régulièrement en fonction du temps. En effet, seul le point GPS initial aide à l'estimation de trajectoire dans le cas d'une propagation *forward* sur l'équation d'évolution, les erreurs d'estimation ne peuvent que s'accumuler au cours du temps. L'enveloppe de la trajectoire correspondante est dessinée dans la fenêtre *2D scene* (voir la figure 4.5 pour plus de détails sur ce que représente cette fenêtre). L'enveloppe devrait être la plus fine près du point de départ, qui se situe au centre de la fenêtre, mais il est difficile de le distinguer à cause de la forme de la trajectoire qui provoque des superpositions, d'où l'intérêt de la fenêtre *Position error*. Une partie de l'image sonar tribord est dessinée dans la fenêtre principale (voir la figure 4.8 pour plus de détails sur ce que représente cette fenêtre), on y voit notamment deux mines reliées par un câble, qui vont être utilisées dans la suite. 92

4.13 Résultats d'une propagation *forward-backward* sur les données de navigation du *Redermor* avec GESMI. Dans la fenêtre *Position error* (voir figure 4.7 pour plus de détails sur ce que représente cette fenêtre), on constate que l'erreur d'estimation de position est maintenant minimale au début et à la fin de la mission. C'est grâce au point GPS final, qui a maintenant pu être pris en compte grâce à la propagation *backward* sur l'équation d'évolution. L'enveloppe de la trajectoire correspondante dessinée dans la fenêtre *2D scene* (voir la figure 4.5 pour plus de détails sur ce que représente cette fenêtre) est maintenant globalement plus fine qu'avec la propagation *forward* seule. 93

4.14 Ajout d'une détection de mine sur une image sonar du *Redermor* avec GESMI. 94

4.15 Résultats d'une propagation forward/backward/mark sur les données de navigation du *Redermor* avec GESMI. Vu qu'on n'a qu'une seule détection d'un amer pour l'instant, l'estimation de la trajectoire du sous-marin reste inchangée après une nouvelle propagation *forward-backward* sur l'équation d'évolution cependant, la prise en compte de l'équation des amers permet d'obtenir la position de la mine dans l'eau et de la dessiner (carré bleu) dans la fenêtre *2D scene* (voir la figure 4.5 pour plus de détails sur ce que représente cette fenêtre). 95

4.16 Ajout d'une deuxième détection de la mine détectée manuellement auparavant, grâce à l'indication modélisée par une enveloppe verte dans la fenêtre principale de GESMI. 96

4.17 Amélioration de l'estimation de position du sous-marin grâce à 2 détections du même amer. Dans la fenêtre *Position error* (voir figure 4.7 pour plus de détails sur ce que représente cette fenêtre), on constate que l'erreur d'estimation de position a une sorte de seuil à un moment donné, qui doit correspondre au passage en face de l'amer détecté. L'enveloppe de la distance robot-amer est maintenant plus étroite au niveau de la deuxième détection d'amer dans la fenêtre principale (voir la figure 4.8 pour plus de détails sur ce que représente cette fenêtre) car cette détection a été rajoutée dans la liste des détections connues précisément à l'étape précédente (le fait qu'elle soit maintenant connue est indiqué par le petit carré rouge). 97

4.18 Trajectoire réelle du robot en vert et positions réelles des amers en bleu, en 2D. 99

4.19 Robot à un instant t voyant 2 amers i_1 et i_2. Cette situation permet d'écrire des équations inter-amers. 100

4.20 Robot voyant le même amer i à 2 instants t_1 et t_2. Cette situation permet d'écrire des équations intertemporelles. 101

4.21 Enveloppes de la trajectoire du robot trouvées par l'algorithme 1 pour le scénario 4 (rouge : $k = 4$, bleu : $k = 3.9$, vert : $k = 3.8$, noir : $k = 3.7$). 104

4.22 Enveloppes des positions des amers trouvées par l'algorithme 1 pour le scénario 4 (rouge : $k = 4$, bleu : $k = 3.9$, vert : $k = 3.8$, noir : $k = 3.7$). 105

4.23 Enveloppes de la trajectoire du robot trouvées pour $k = 4$ pour le scénario 4 (rouge : algorithme 1, noir : algorithme 4). Les résultats des 2 algorithmes sont presque confondus. 106

4.24 Enveloppes des positions des amers trouvées pour $k = 4$ pour le scénario 4 (rouge : algorithme 1, noir : pavés produits à la fin de l'algorithme 4, magenta : sous-pavage produit par les bissections de l'algorithme 4). Il faut noter que les pavés noirs sont (en général) les plus petits pavés englobant les sous-pavages magenta trouvés en faisant des bissections (en fait il est possible que les pavés noirs soient encore plus petits que le plus petit pavé englobant car pour un amer donné, le pavé noir trouvé peut avoir été amélioré après la génération de son sous-pavage magenta). On remarque que l'algorithme 4 réussit à améliorer principalement les grands pavés. On peut donc conclure que l'algorithme 1 n'est pas optimal, mais il n'en est pas trop loin pour les petits pavés. 107

4.25 Enveloppes de la trajectoire du robot trouvées pour le scénario 8 (rouge : algorithme 1, bleu : algorithme 2, vert : algorithme 3, noir : algorithme 4). Les résultats des différents algorithmes sont presque confondus : on voit que ni la prise en compte d'équations d'interdétections propres au problème (algorithme 2), ni l'ajout de bissections qu'elles soient sur des tranches de pavés (algorithme 3) ou complètes sur les positions des amers (algorithme 4) n'améliorent de manière significative l'estimation des positions x et y du robot. 109

4.26 Enveloppes des positions des amers trouvées pour le scénario 8 (rouge : algorithme 1, bleu : algorithme 2, vert : algorithme 3, noir : pavés produits à la fin de l'algorithme 4, magenta : sous-pavage produit par les bissections de l'algorithme 4). Il faut noter que les pavés noirs sont (en général) les plus petits pavés englobant les sous-pavages magenta trouvés en faisant des bissections (en fait il est possible que les pavés noirs soient encore plus petits que le plus petit pavé englobant car pour un amer donné, le pavé noir trouvé peut avoir été amélioré après la génération de son sous-pavage magenta). Cette figure est la plus importante : on voit que ni la prise en compte d'équations d'interdétections propres au problème (algorithme 2), ni l'ajout de bissections qu'elles soient sur des tranches de pavés (algorithme 3) ou complètes sur les positions des amers (algorithme 4) n'améliorent de manière significative l'estimation des positions x et y des amers. On peut alors en déduire que l'algorithme 1, qui est rapide et utilisable directement pour tout type de problème de SLAM est déjà presque optimal. 110

4.27 Enveloppes de la trajectoire du robot trouvées pour le scénario 12 (rouge : algorithme 1, noir : algorithme 4). Les résultats des 2 algorithmes sont aussi presque confondus. 111

4.28 Enveloppes des positions des amers trouvées pour le scénario 12 (rouge : algorithme 1, noir : pavés produits à la fin de l'algorithme 4, magenta : sous-pavage produit par les bissections de l'algorithme 4). Il faut noter que les pavés noirs sont (en général) les plus petits pavés englobant les sous-pavages magenta trouvés en faisant des bissections (en fait il est possible que les pavés noirs soient encore plus petits que le plus petit pavé englobant car pour un amer donné, le pavé noir trouvé peut avoir été amélioré après la génération de son sous-pavage magenta). Ces résultats confirment ce qui a été constaté avec le scénario 8 par exemple. . . . 112

5.1 Partie d'image sonar du *Redermor* affichée dans GESMI. La zone rose à gauche est considérée comme étant des données sonar non exploitables pour détecter un amer. En effet, elle correspond à la water column (voir 2.2.1). Les rectangles blancs sont les pavés récupérés suite à un seuillage de l'image sonar : il est possible qu'un amer s'y cache. La zone entourée de vert est l'indication qu'un amer devrait avoir été revu sur l'image sonar d'après les calculs de GESMI. En regardant l'intersection entre cette zone et les pavés résultants du seuillage, on voit qu'il serait possible de réduire la taille de la zone verte estimée par GESMI de manière automatique. 116

5.2 Les points fugaces (points noirs) sont supposés appartenir à la waterfall $\mathbb{W}(t)$. 117

5.3 Partie d'une image sonar venant du *Redermor*, où on voit des rides de sable et une mine. . . 118

5.4 A gauche, image produite par un télémètre laser rotatif dans une petite pièce. Le rouge correspond aux murs de la pièce (comme si c'était une vue du dessus), les zones bleues à des zones vides d'obstacles et le jaune à ce qui a changé depuis le dernier tour de laser (le laser était presque immobile). A droite, waterfall produite schématiquement à partir de l'image de gauche. Le rouge, bleu et jaune correspondent aux mêmes éléments qu'à gauche. On voit que les discontinuités correspondent aux coins de la pièce. Cette waterfall est obtenue en mettant les distances mesurées par le télémètre à mesure qu'il tourne les unes au-dessus des autres. . 119

5.5 Exemple de contraction avec la relation de visibilité. 124

5.6 Contraction du tube $[y](t)$ quand on sait (par le signe de v) qu'une donnée visible a été détectée. a) Sélection de tranches telles que $0 \in [v](k)$ pour tout $k \in [k_1+1, k_2-1]$ (étape 1). b) Illustration de l'intersection entre $[y](k)$ et $\mathbb{W}(k)$. c) Calculs des sous-tubes temporaires pour chaque $k \in [k_1+1, k_2-1]$ (étape 2). d) Tube $[y](t)$ résultant de l'union des sous-tubes calculés en c) suivie de l'intersection de cette union avec le tube initial $[y](t)$ (étape 3). . . . 125

5.7 Le robot est seulement équipé d'un télémètre laser pour se localiser et connait seulement la position de l'amer $\mathbf{m}$. Le carré et le triangle peuvent être détectés par le télémètre mais leur présence est inconnue du robot à l'avance. 126

5.8 Le robot à différents instants de la simulation. 129

5.9 Le robot sait seulement que les points fugaces $(t, g(\mathbf{x}(t)))$ sont toujours dans la waterfall $\mathbb{W}(t)$ (en gris) ou (de manière équivalente) qu'aucun point fugace n'est dans la zone blanche. 130

5.10 Enveloppe obtenue après une propagation intervalle sans utiliser les données du télémètre (à gauche) ; Enveloppe obtenue après une propagation intervalle en prenant en compte les données du télémètre (à droite). Le trait continu en gris moyen est la forme de la pièce dans laquelle le robot évolue. 131

5.11 Partie de la waterfall (zone noire) montrant une zone fugace (zone entourée de vert et délimitée par les 2 traits horizontaux blancs), à l'étape 1 de la contraction de $[y](t)$ par rapport à la contrainte de visibilité. Le temps est en ordonnées et la distance au robot en abscisses. 131

5.12 Sous-tube $[y](t)$ obtenu en supposant qu'un point fugace se trouve à l'instant indiqué par le trait horizontal orange. 132

5.13 Il ne peut pas avoir de point fugace à l'instant indiqué par le trait horizontal orange car la zone fugace connue jusqu'à maintenant (entourée de vert) n'intersecte pas la waterfall (zone noire). 132

5.14 Sous-tube $[y](t)$ obtenu en supposant qu'un point fugace se trouve à un autre instant, indiqué par le trait horizontal orange. 133

5.15 Résultat de l'étape 3 de la contraction de $[y](t)$ par rapport à la condition de visibilité. La zone entourée de vert est la zone fugace initiale et la zone jaune est le résultat de la contraction. 133

5.16 Ensemble des pavés dont l'union contient la carte de l'environnement (à gauche) ; Approximation de la carte en dessinant le centre de ces pavés (à droite). 134

5.17 Construction de la waterfall $\mathbb{W}$ sans la dimension liée à l'altitude (à droite) à partir d'une image de sonar latéral (à gauche). Elle est formée à partir des zones brillantes de l'image sonar. Il n'y a pas d'amers dans la zone blanche, mais il y en a peut être dans la zone grise. . 136

5.18 Illustration des fonctions de visibilité et d'observation en SLAM sous-marin. La fonction de visibilité d'un amer est calculée à partir des équations d'état, des données de navigation du sous-marin et d'au minimum une détection de l'amer. Elle modélise le passage de l'amer en face du sonar du robot. Elle peut être définie simplement par l'angle de vue de l'amer par rapport au sonar du sous-marin. On ne peut pas passer par une situation où l'amer est devant puis derrière sans avoir un moment où l'amer tombe en face du sonar du robot. La fonction d'observation d'un amer est calculée à partir des équations d'état, des données de navigation du sous-marin et d'au minimum une détection de l'amer. Elle peut être définie par la distance robot-amer. L'image sonar, à travers la waterfall et la fonction de visibilité vont permettre d'améliorer la connaissance de la fonction d'observation. Inversement, la waterfall et la fonction d'observation vont permettre d'améliorer la connaissance de la fonction de visibilité. L'amélioration de la connaissance de ces 2 fonctions aura pour conséquence d'améliorer l'estimation de la trajectoire du robot et de la position des amers (souvent des mines dans le cadre de la thèse). De plus, lorsqu'elles seront dessinées sur l'image sonar, ces fonctions vont aider l'opérateur humain dans sa recherche de mines. 137

5.19 Illustration sous forme de graphe des contraintes du processus de contraction et propagation pour un problème de SLAM avec données fugaces. Ce graphe est formé d'hyperarcs orientés vers les variables qui peuvent être contractées (quand on le sait). 138

Remerciements

Je tiens tout d'abord à remercier tous les membres de mon jury de thèse pour le temps qu'ils m'ont consacré ainsi que l'attention et l'intérêt qu'ils ont montrés lors de la relecture de mon manuscrit et ma présentation : leurs corrections, questions et remarques pertinentes et constructives m'ont permis d'améliorer ce manuscrit et ont valorisé mon travail. En particulier, je remercie Patrick RIVES d'avoir rapporté ma thèse et d'y avoir apporté un angle de vue probabiliste ainsi que Philippe BONNIFAIT pour tous ses conseils et l'intérêt qu'il a manifesté à travers son rapport, Pierre DE LOOR pour avoir accepté de présider ma soutenance de thèse, Eva CRUCK (et Jacques BLANC-TALON pour le début de ma thèse) pour avoir été mes correspondants DGA, Alain BERTHOLOM pour avoir apporté le support du GESMA (mise à disposition des données des sous-marins *Redermor* et *Daurade*...) et enfin Luc JAULIN, pour avoir été un directeur de thèse parfait du début à la fin de ma thèse, que je recommande à tous les futurs doctorants en robotique et calcul par intervalles, et sans qui il n'y aurait sans doute pas de robotique et calcul par intervalles à l'ENSTA Bretagne, et même peut-être ailleurs !

Je remercie également Jan SLIWKA (doctorant encadré par Luc JAULIN qui a commencé sa thèse en même temps que moi et avec qui j'ai beaucoup travaillé sur de nombreux projets) pour m'avoir aidé et supporté tout au long de la thèse dans toutes les situations, Aymeric BETHENCOURT (doctorant encadré par Luc JAULIN qui a commencé peu avant ma soutenance) notamment pour la relecture de mon manuscrit et ses contributions au développement de la robotique à l'ENSTA Bretagne, Olivier REYNET, Benoît CLEMENT, Yvon GALLOU, Benoît ZERR pour leur aide dans les différents projets robotique sur lesquels on a travaillé ensemble, les secrétaires du Pôle STIC de l'ENSTA Bretagne Annick BILLON-COAT et Michèle HOFMANN (que j'ai beaucoup sollicitées durant ma thèse notamment pour les commandes de matériel, les déplacements, la partie administrative de la thèse et en particulier l'organisation de la soutenance...), Philippe DHAUSSY (responsable du Pôle STIC), Yann DOUTRELEAU (directeur scientifique de l'ENSTA Bretagne), tous les collègues de bureau et du laboratoire ainsi que les différents personnels, élèves, stagiaires qui m'ont aidé ou avec qui j'ai participé aux activités et concours robotiques avec l'ENSTA Bretagne.

Je tiens aussi à remercier Gilles CHABERT pour son support pour l'utilisation de la bibliothèque de calcul par intervalles IBEX, et Cyril JOLY pour m'avoir aidé et permis de réutiliser ses données de simulation pour tester mes propres méthodes de SLAM.

Je remercie enfin l'UBO et l'ED SICMA, notamment Michèle KERLEROUX pour avoir suivi le bon déroulement de ma thèse, ainsi que la DGA et le CNRS pour l'avoir financée et administrée.

Chapitre 1

Introduction

1.1 Contexte

Un robot sous-marin connait en général sa position initiale (lorsqu'il est à la surface grâce à un *GPS*), son modèle de déplacement (très approximativement) et possède quelques capteurs l'aidant à estimer sa position (*écho-sondeur* pour mesurer sa distance au fond, *capteur de pression* pour mesurer sa profondeur, *loch Doppler* pour mesurer sa vitesse, *centrale inertielle* pour mesurer son orientation) et voir ce qui l'entoure (*sonar*). Pourtant, malgré tous ces capteurs, plus il avance, plus ses erreurs d'estimation de position s'accumulent : le robot se perd. L'idée du *SLAM* (Simultaneous Localization And Mapping) est d'essayer de limiter ce problème en essayant de trouver des points de repères (*amers*) et relever leur position lorsqu'on arrive bien à se localiser (*cartographie* à partir de la *localisation*), puis d'utiliser la position de ces points de repères quand on repasse devant et qu'on est perdu pour essayer de retrouver la trajectoire qu'on a faite (localisation grâce à la cartographie). Le SLAM est donc fortement lié à 2 problèmes : la localisation, qui correspond à chercher sa position par rapport à un repère qu'on s'est défini, et la cartographie, qui est le calcul de la position des éléments constituant son environnement.

Différentes méthodes existent pour traiter le problème du SLAM. Les mesures de capteurs ou variables utilisées pour décrire les robots étant souvent entachées d'erreurs, elles peuvent être représentées de différentes façons : distributions probabilistes, nuages de points, ensembles continus... En général, les données constructeur des capteurs ou actionneurs du robot nous indiquent des bornes (liées à la précision,...). On peut donc représenter ces valeurs sous forme d'*intervalles*. Les méthodes ensemblistes telles que l'analyse par intervalles permettent d'obtenir des résultats à partir d'équations ou inéquations sur des intervalles. Utiliser le calcul par intervalles a notamment ces avantages :

- Fonctionne dans les cas non-linéaires sans approximations particulières. En effet, les fonctions usuelles telles que cos, $\sqrt{}$, exp,... peuvent être utilisées dans des équations intervalles.
- Résultats garantis : aucune solution possible compte tenu des données et hypothèses faites lors de la modélisation du problème étudié n'est perdue.
- Validation, débogage : si un ensemble vide apparaît au cours des calculs, cela signifie qu'il y a une incohérence quelque part et nous permet donc de détecter des problèmes d'algorithmes, conception ou modélisation du robot.

– Permet de résoudre rapidement (quelques secondes) des problèmes mettant en jeu un grand nombre d'équations (correspondant par exemple à toute une mission faite par un robot), en utilisant un processus de propagation de contraintes.

1.2 Objectifs, hypothèses, contraintes, restrictions et contributions

Dans cette thèse, nous nous plaçons dans le cadre du SLAM *offline* (voir section 4.1) appliqué aux robots sous-marins (avec données réelles et aussi en simulation) en utilisant le calcul par intervalles et la *propagation de contraintes* (voir sous-section 3.4.1), sans *données aberrantes* et en supposant que l'*association des données* liées aux amers (amers immobiles et ponctuels) est connue (cependant nous verrons comment on peut faciliter le traitement du problème de distinction des amers en utilisant les équations d'état du sous-marin et ses données de navigation plutôt que l'image sonar seule dans la sous-section 4.3.2), et sans *bissections* (voir sous-section 3.4.2) sauf dans la partie comparaison (voir section 4.4). Nous verrons que le problème du SLAM peut être vu comme des problèmes d'estimation d'état et de paramètres, à erreur bornée dans le cas des intervalles.

Les robots mobiles étudiés seront modélisés avec la représentation d'état suivante :

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x},\mathbf{u}) \quad \text{(équation d'évolution)} \\ \mathbf{y} &= \mathbf{g}(\mathbf{x},\mathbf{u}) \quad \text{(équation d'observation)} \end{cases} \tag{1.1}$$

avec $\mathbf{f}$ fonction d'évolution, $\mathbf{g}$ fonction de mesure, $\mathbf{x}$ vecteur d'état, $\mathbf{u}$ vecteur de commande et $\mathbf{y}$ vecteur des mesures.

Dans les programmes et simulations faits sur ordinateur, la représentation utilisée correspond à celle des systèmes à temps discrets :

$$\begin{cases} \mathbf{x}_{k+1} &= \mathbf{f}(\mathbf{x}_k,\mathbf{u}_k) \quad \text{(équation d'évolution)} \\ \mathbf{y}_k &= \mathbf{g}(\mathbf{x}_k,\mathbf{u}_k) \quad \text{(équation d'observation)} \end{cases} \tag{1.2}$$

avec $t = k.dt$ où dt est le pas de temps.

Des variantes de ces représentations seront utilisées, notamment pour représenter les problèmes du SLAM (voir section 4.3) ou des *données fugaces* (voir section 5.2). Dans toutes les équations d'état, les coefficients éventuels seront (sauf indication contraire) considérés à 1 et donc omis pour simplifier l'écriture (équations d'état normalisées).

Pour essayer d'évaluer la qualité des résultats donnés par la méthode de SLAM sous-marin par intervalles utilisée, une comparaison entre différentes méthodes de traitement sur des données simulées a été faite (voir section 4.4). Une amélioration de la prise en compte des données sonar et une formalisation de la problématique des données fugaces qui en découle est proposée avec l'approche des *tubes*, pour essayer d'améliorer les résultats du SLAM (voir chapitre 5).

1.3 Annonce du plan

Le premier chapitre de cette thèse apportera les notions techniques nécessaires pour comprendre les robots étudiés et leurs similarités. Dans une première section, les capteurs les plus utilisés et les informations qu'on peut en retirer seront énumérés. La deuxième section de ce chapitre sera consacrée à la description détaillée des sous-marins autonomes utilisés au cours de la thèse, et notamment leur modélisation et le principe de fonctionnement de leurs capteurs les plus importants. Enfin, des robots terrestres ayant un principe de fonctionnement et localisation similaire à ceux des sous-marins seront présentés pour justifier la possibilité et l'intérêt d'adapter les méthodes utilisées dans la thèse à l'un ou l'autre des types de robots.

Dans un deuxième chapitre, les bases du calcul par intervalles et les concepts et outils relatifs utilisés dans le reste de la thèse seront décrits. La première section présentera de manière générale l'idée d'utiliser les intervalles, leurs avantages et inconvénients majeurs, ainsi que leurs possibilités d'applications déjà démontrées. La deuxième section présentera plus formellement les intervalles. La troisième section montrera des généralisations possibles des intervalles (pavés, tubes), particulièrement utiles en robotique. Les principales techniques de calcul par intervalles (contraction et bissection) seront décrites dans une quatrième section. La dernière section indiquera des outils disponibles pour pouvoir programmer efficacement avec des intervalles.

Le principe du SLAM, des méthodes pour le traiter et des comparaisons entre celles-ci seront ensuite présentés dans un troisième chapitre. Le SLAM et les problèmes sous-jacents seront introduits dans la première section. Les méthodes existantes pour traiter le problème du SLAM seront ensuite abordées dans une deuxième section. La troisième section sera consacrée au SLAM par intervalles par propagation de contraintes, avec notamment son application au cas du robot sous-marin autonome *Daurade*. Une comparaison prenant la suite de celle déjà faite dans [Joly, 2010] sera proposée dans la dernière section de ce chapitre.

Enfin, nous montrerons dans le dernier chapitre comment il est possible d'automatiser les résultats du SLAM sous-marin par intervalles déjà implémenté en prenant en compte les données sonar différemment. Le principe de cette idée sera présenté dans la première section. Ceci nous amènera à poser le problème des données fugaces et proposer une méthode permettant de le gérer en utilisant le formalisme des tubes, dans la deuxième section. La dernière section montrera comment appliquer le formalisme et l'algorithme de traitement correspondant au SLAM sous-marin.

Chapitre 2

Robotique et sous-marins

Dans ce chapitre, les capteurs et conventions utilisés pour décrire les robots, ainsi que les notions techniques liées aux capteurs les plus importants pour la suite seront présentés. Un parallèle entre des robots autonomes sous-marins et terrestres utilisés au cours de la thèse sera fait, pour expliquer leurs similarités et justifier l'utilisation des mêmes principes sur différents types de robots dans la suite du document (pour le SLAM, par exemple dans la section 4.4 et pour le problème des données fugaces, dans le chapitre 5).

La première section présentera les principaux capteurs qui peuvent être trouvés sur des robots terrestres et sous-marins. Les informations que l'on peut retirer des données de ces capteurs seront notamment indiquées.

La deuxième section présentera les principaux robots sous-marins utilisés au cours de la thèse. La *Daurade* et le *Redermor* du GESMA (Groupe d'Etudes Sous-Marines de l'Atlantique) sont deux exemples de robots sous-marins munis de nombreux capteurs très précis. *SAUC'ISSE* (de l'ENSTA Bretagne) est plutôt un exemple de mini-robot avec peu de capteurs et plutôt peu cher. Les données de la *Daurade* et du *Redermor* seront traitées dans la sous-section 4.3.2 et le robot *SAUC'ISSE* illustrera un exemple présenté dans la sous-section 3.4.3.

La troisième section de ce chapitre sera consacrée à des robots terrestres. Les robots JOGs seront utilisés comme exemples plusieurs fois (*e.g.* dans les sections 3.2, 3.4). Les robots BOURRICOTs seront notamment une illustration de l'exemple traité dans la sous-section 5.2.2.

La dernière section conclura le chapitre et fera le lien entre les différents robots présentés et la suite du document.

2.1 Présentation

La robotique est un champ vaste et interdisciplinaire. On peut définir un robot comme un système mécanique, électronique et informatique contrôlé ou programmé pour effectuer des tâches trop répétitives, dangereuses ou difficiles pour être faites directement par des êtres humains. Les robots peuvent être domestiques, industriels, de loisirs, de transport, et faits pour des applications civiles ou militaires. La catégorie de robots qui nous intéresse plus particulièrement est celle des robots mobiles. Un robot mobile est en général équipé de capteurs,

actionneurs, calculateurs et de systèmes de gestion d'énergie. Selon que le robot est autonome ou téléopéré (contrôlé à distance par un ou plusieurs opérateurs humains), ses parties calculateur (intelligence) et énergie sont plus ou moins importantes. Les capteurs d'un robot peuvent être divisés en 3 catégories :

- Les capteurs proprioceptifs : ils fournissent une information sur l'état interne actuel du robot, par rapport à une référence interne.
- Les capteurs extéroceptifs : ils fournissent une information sur l'état du robot relativement à son environnement.
- Les capteurs exproprioceptifs : combinaison de proprioceptif et extéroceptif.

Une classification de différents capteurs couramment utilisés en robotique mobile est proposée figure 2.4 et dans le tableau 2.1. Il faut noter qu'il est parfois difficile de classer certains capteurs dans l'une ou l'autre des catégories, les notions de proprioception et extéroception variant parfois selon les communautés.

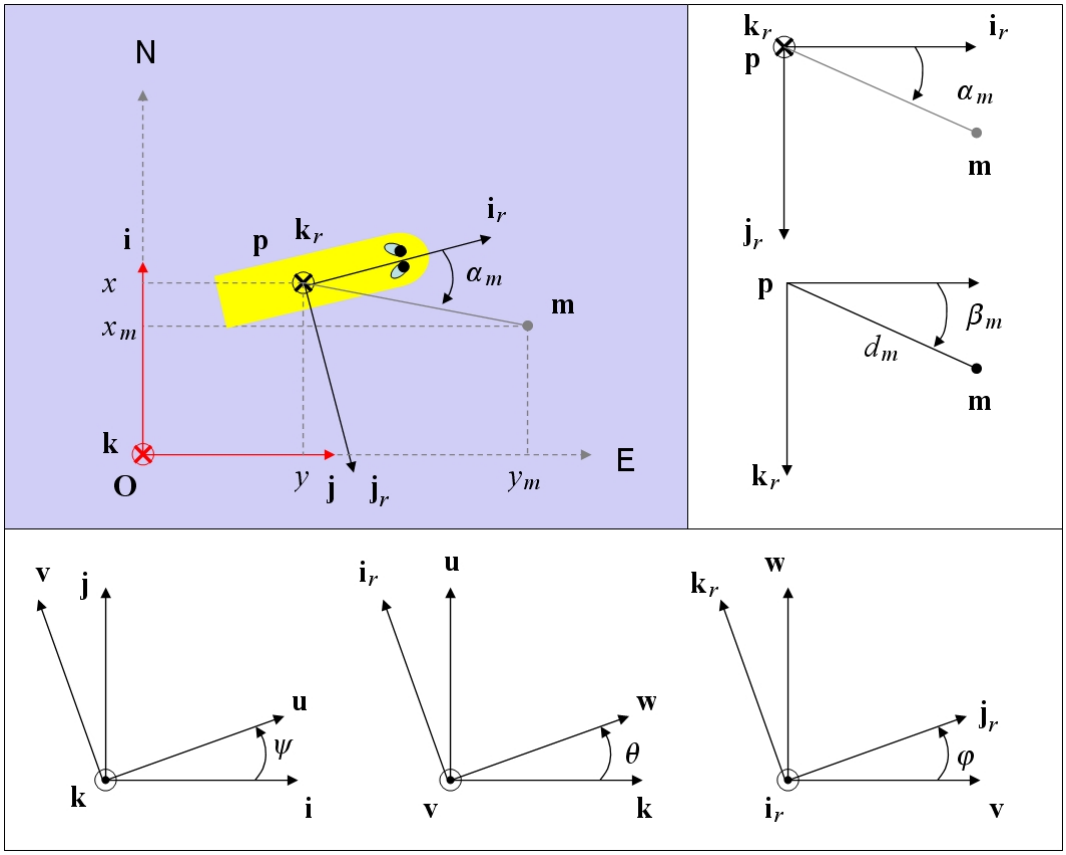


Figure 2.1 – Position $\mathbf{p} = (x, y, z)$ et orientation φ, θ, ψ (angles de roulis, tangage et lacet) d'un robot sous-marin auquel on associe le repère mobile $(\mathbf{p}, \mathbf{i}_r, \mathbf{j}_r, \mathbf{k}_r)$, par rapport au repère de référence $(\mathbf{O}, \mathbf{i}, \mathbf{j}, \mathbf{k})$ où $\mathbf{O}$ est la position du robot lors de sa mise à l'eau, le vecteur $\mathbf{i}$ indique le Nord, $\mathbf{j}$ indique l'Est et $\mathbf{k}$ est orienté vers le centre de la Terre (l'axe des z est souvent tourné vers le bas pour un sous-marin, plutôt vers le haut pour un robot aérien) avec mesure d'angle α_m, β_m et distance d_m à un amer $\mathbf{m} = (x_m, y_m, z_m)$.

Pour pouvoir effectuer sa mission, un robot mobile autonome (voir figures 2.1, 2.2 et 2.3) a bien souvent besoin de capteurs divers pour connaître sa position et attitude $(x, y, z, \varphi, \theta, \psi)$ à tout moment (localisation) et

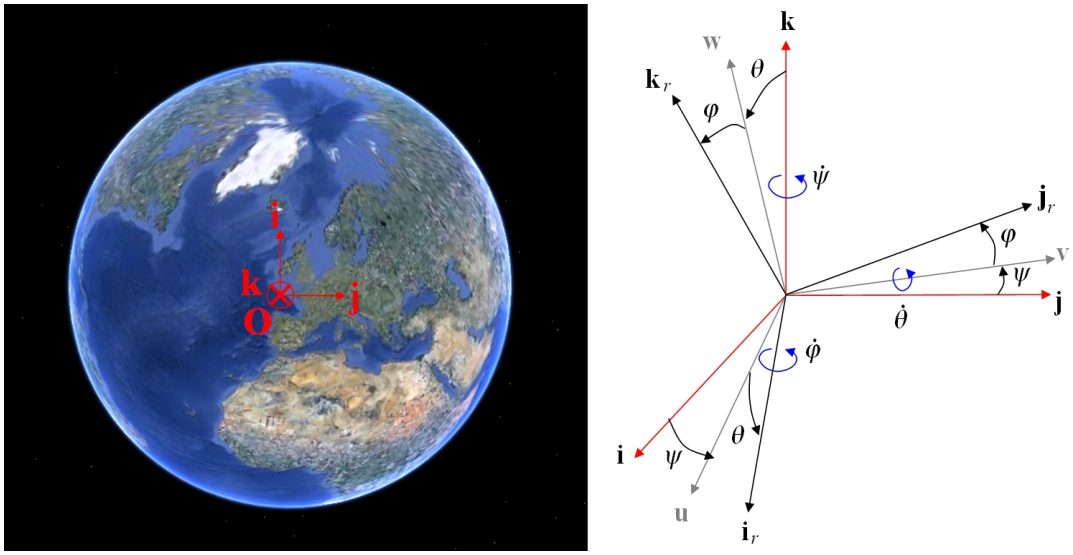


Figure 2.2 – Position et orientation du repère mobile par rapport au repère de référence.

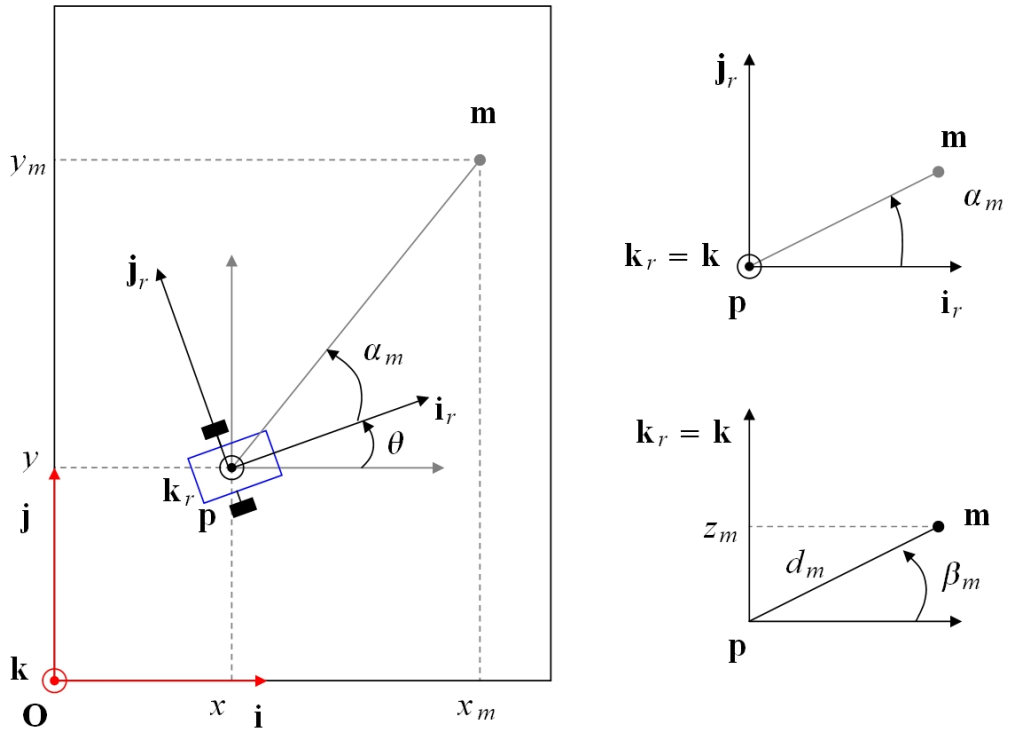


Figure 2.3 – Position et orientation x, y, θ d'un robot char se déplaçant en 2D (ici θ correspond plutôt au ψ du robot 3D) dans le plan dans une pièce rectangulaire coïncidant avec le repère de référence avec mesure d'angle α_m, β_m et distance d_m à un amer $\mathbf{m} = (x_m, y_m, z_m)$ (contexte de SLAM 2D/3D).

détecter la position d'amers (x_m, y_m, z_m), leur distance d_m, leur angle de gisement α_m, leur angle d'élévation β_m (cartographie). Voici des exemples de capteurs qu'on trouve couramment sur un robot mobile :

- **Boussole** (compass en anglais). Elle permet de mesurer l'angle par rapport au Nord magnétique. Elle aide donc à trouver l'angle θ d'un robot 2D. Parfois, 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 (comme décrit en 3.4.3) provoquent des perturbations magnétiques (à cause de pièces métalliques, courants électriques...).
- **Magnétomètre**. Sur le même principe que la boussole, un magnétomètre permet de mesurer le champ magnétique terrestre. Si on prend 3 magnétomètres placés sur les axes x, y, z, on peut obtenir les angles de roulis, tangage, lacet φ, θ, ψ du robot. Comme la boussole, les magnétomètres peuvent être perturbés par leur environnement. Par contre, il est parfois possible de prévoir ces perturbations, ou de les cartographier au préalable puis les prendre en compte lors de la mission du robot.
- **Inclinomètre**. Permet de mesurer l'angle par rapport à l'horizontale.
- **Gyromètre**. Un gyromètre permet de mesurer une vitesse instantanée de rotation. Si on prend 3 gyromètres placés sur les axes x, y, z, on peut obtenir les vitesses de rotation $\dot{\varphi}, \dot{\theta}, \dot{\psi}$ du robot, et par intégration φ, θ, ψ.
- **Accéléromètre**. Sur le même principe de mesure d'inertie qu'un gyromètre, un accéléromètre permet de mesurer une accélération instantanée en translation. Un accéléromètre 3 axes permet de mesurer les accélérations dans le repère du robot. On peut ensuite en déduire $\ddot{x}, \ddot{y}, \ddot{z}$ si on connait la matrice de rotation permettant de passer du repère du robot au repère de référence (grâce aux magnétomètres, inclinomètres ou gyromètres par exemple). Du fait qu'un accéléromètre mesure l'accélération de la gravité (qui en général ne nous intéresse pas et nous perturbe pour évaluer la position du robot), accéléromètres, gyromètres, inclinomètres et boussoles sont souvent regroupés dans un même capteur, ce qui aide à déterminer la composante à éliminer.
- **GPS** (Global Positioning System). Il permet d'obtenir latitude, longitude, altitude et vitesses (par rapport au repère WGS84) grâce à des satellites (voir *e.g.* [Drevelle and Bonnifait, 2010]). On peut donc en déduire $x, y, z, \dot{x}, \dot{y}, \dot{z}$. Des variantes telles que le DGPS (Differential Global Positioning System, utilisant des stations au sol en plus des satellites comme référence) ou l'AGPS (Assisted Global Positioning System, utilisant par exemple Internet via la 3G ou le Wifi pour obtenir des informations supplémentaires, couramment utilisé dans les smartphones) permettent d'améliorer la précision et/ou la rapidité de mise à jour des données lorsque la communication avec les satellites est perturbée (obstacles cachant le ciel, interférences...). Il faut noter que le signal GPS ne passe pas sous l'eau et ne peut donc être utilisé par un sous-marin que lorsque celui-ci est en surface.
- **IMU** (Inertial Measurement Unit). C'est une combinaison d'accéléromètres, gyromètres, inclinomètres, magnétomètres permettant d'obtenir $\ddot{x}, \ddot{y}, \ddot{z}, \dot{\varphi}, \dot{\theta}, \dot{\psi}, \varphi, \theta, \psi$. On parle d'AHRS (Attitude and Heading Reference System) si le capteur fait un traitement numérique global des données, et d'INS (Inertial Navigation System) lorsqu'elle retourne en plus $x, y, z, \dot{x}, \dot{y}, \dot{z}$. Si un GPS y est intégré, on obtient alors $x, y, z, \dot{x}, \dot{y}, \dot{z}$ de manière beaucoup plus fiable (obtenir ces valeurs à partir des données inertielles n'est pas toujours suffisamment précis pour être exploitable pendant une longue durée, à cause de l'accumulation des erreurs dues aux intégrations).
- **Odomètre** (encoder en anglais). Compte le nombre de tours d'une roue. On peut en général déduire les vitesses $\dot{x}, \dot{y}, \dot{\theta}$ d'un robot roulant 2D. Bien que ce soit parfois fait, les valeurs sont souvent trop imprécises pour en déduire x, y, θ de manière suffisamment précise, notamment à cause des glissements.

- **Altimètre barométrique** (le terme altimètre désigne en général un capteur permettant de mesurer une altitude) ou baromètre. Mesure la pression dans l'air. On peut en déduire z.
- **Capteur de pression**. Mesure la pression dans l'eau. On peut en déduire z. Il est aussi parfois possible de déduire les vitesses $\dot{x}, \dot{y}, \dot{z}$ selon le positionnement des capteurs et leur type (capteur de pression statique pour mesurer la profondeur, dynamique pour les vitesses) mais ce n'est pas toujours précis, notamment à cause des courants ou turbulences dans l'eau.
- **Loch**. Permet de mesurer une vitesse dans l'eau, par rapport à l'eau, la surface ou le fond de l'eau. On peut avoir des lochs à hélice, à ultrasons (loch Doppler ou DVL pour Doppler Velocity Log), à capteur de pression dynamique... On peut donc en déduire $\dot{x}, \dot{y}, \dot{z}$ et même parfois z ou l'altitude (pour les lochs ultrasons).
- **Caméra**. Beaucoup d'informations peuvent être théoriquement obtenues à partir des images d'une caméra (voir *e.g.* [Mei and Rives, 2007]), mais les traitements nécessaires peuvent être compliqués. On peut notamment mesurer son angle ou sa distance par rapport à des amers, sa vitesse (en calculant le flux optique, voir aussi [Rodriguez et al., 2009]). Pour améliorer les résultats ou simplifier les traitements, on utilise parfois plusieurs caméras (stéréovision si 2 caméras observent la même direction...), des miroirs devant une caméra (miroir sphérique pour faire une caméra omnidirectionnelle [Joly and Rives, 2009]...). On peut donc selon les situations déduire $x, y, z, \dot{x}, \dot{y}, \dot{z}, \varphi, \theta, \psi, \dot{\varphi}, \dot{\theta}, \dot{\psi}, x_m, y_m, z_m, \alpha_m, \beta_m, d_m$.
- **Sonar** (Sound Navigation And Ranging), sondeur ou échosondeur. Permet de mesurer une distance à un objet par acoustique (surtout utilisé dans l'eau mais parfois aussi dans l'air). On peut par exemple en déduire z quand on est en mer (en mesurant la distance par rapport au fond) ou même x, y si on est dans une piscine ou marina (en mesurant la distance par rapport aux bords). On parle plus de sonar lorsqu'on a accès au détail du signal acoustique au lieu d'une simple distance au premier obstacle (télémètre). Il est dit actif s'il émet un signal, passif s'il ne fait qu'écouter. On parle de sonar latéral (ou sidescan sonar) lorsqu'un faisceau part du côté bâbord et/ou tribord et est dirigé vers le fond de l'eau (cartographie du fond), frontal s'il est dirigé vers l'avant (évitement d'obstacles) ou rotatif lorsque le faisceau tourne et est dirigé vers les bords d'une piscine par exemple. Dans ce dernier cas, on peut alors aussi déduire θ. Les données sonar sont souvent présentées sous forme d'images montrant les intensités des échos reçus en fonction du temps et de la distance. On peut y appliquer des traitements similaires à ceux d'images vidéos : détection d'amers puis calcul d'angle ou distance et position (on a α_m, d_m et éventuellement x_m, y_m, z_m). L'utilité des sonars pour améliorer la localisation et même le SLAM a déjà été prouvée dans de nombreuses applications (*e.g.* [Leonard and Durrant-Whyte, 1992], [Leblond, 2006], [Ribas et al., 2010]).
- **Télémètre ultrason/infrarouge/laser** (le terme télémètre désigne en général un capteur permettant de mesurer une distance à un objet). Permet de mesurer une distance à un objet dans l'air. On peut donc en déduire x, y, z selon l'environnement du robot et le positionnement du capteur. Pour améliorer les résultats et/ou faciliter les traitements, ils peuvent parfois être rotatifs (il est même possible d'obtenir des résultats en 3D avec 2 axes de rotation). On peut aussi traduire les données en images de la même façon que pour les sonars. On peut donc selon les situations déduire x, y, θ (et même $x, y, z, \varphi, \theta, \psi$ avec un télémètre laser 3D) et α_m, d_m (et β_m avec un télémètre laser 3D) et éventuellement x_m, y_m, z_m pour les amers.
- **Radar** (RAdio Detection And Ranging). On peut le voir comme un équivalent du sonar, mais utilisant les ondes électromagnétiques (voir par exemple [Reynet and Jaulin, 2010]).
- **Lidar** (LIght Detection And Ranging). Même principe que le radar mais avec de la lumière (voir *e.g.* [Guyonneau et al., 2011]).
- **Récepteurs HF, GSM, 3G, Wifi**... En plus de permettre la communication, ces dispositifs peuvent par-

fois permettre aussi de se localiser en mesurant par exemple les puissances de réception ([Ferris et al., 2007])...

Note sur les équivalents eau/air : Certains capteurs peuvent être nommés différemment selon le milieu dans lequel ils sont utilisés et selon le principe qu'ils utilisent, bien qu'ils soient utilisés dans le même but et produisent des données similaires (radar, sonar, télémètre ultrason/infrarouge/laser...). Cela vient en général du fait que les caractéristiques de propagation (vitesse, temps de réponse, portée, atténuation, réflexion...) des signaux ne sont pas les mêmes dans l'eau ou dans l'air. Ainsi, les signaux ultrasons (acoustique) sont en général les plus utilisés dans l'eau puisque l'infrarouge, le laser ou les ondes électromagnétiques s'y propagent en général très mal. Dans l'air, l'acoustique n'est utilisée que pour mesurer de faibles distances (quelques centimètres en général), l'infrarouge ou le laser pour des distances de quelques mètres et les ondes électromagnétiques pour les grandes distances (kilomètres).

capteur	**sous-marin**	**aérien**	**proprioceptif**	**extéroceptif**	**exproprioceptif**
boussole	+	+	+		
magnétomètre	+	+	+		
inclinomètre	+	+	+		
gyromètre	+	+	+		
accéléromètre	+	+	+		
GPS		+		+	
IMU/AHRS	+	+	+		
INS	+	+			+
odomètre		+	+		
altimètre barométrique		+	+		
capteur de pression	+		+		
loch hélice/pression	+	+	+		
DVL	+			+	
caméra	+	+		+	
sonar	+			+	
télémètre ultrason		+		+	
télémètre infrarouge		+		+	
télémètre laser		+		+	
radar		+		+	
lidar		+		+	
récepteur radios		+		+	

(2.1)

2.2 Exemples de robots sous-marins autonomes

Des données venant de plusieurs sous-marins ont été traitées au cours de la thèse (voir sous-sections 4.3.2 pour la *Daurade* et le *Redermor* et 3.4.3 pour *SAUC'ISSE*). Cette section présentera les spécificités de ces AUVs (Autonomous Underwater Vehicles).

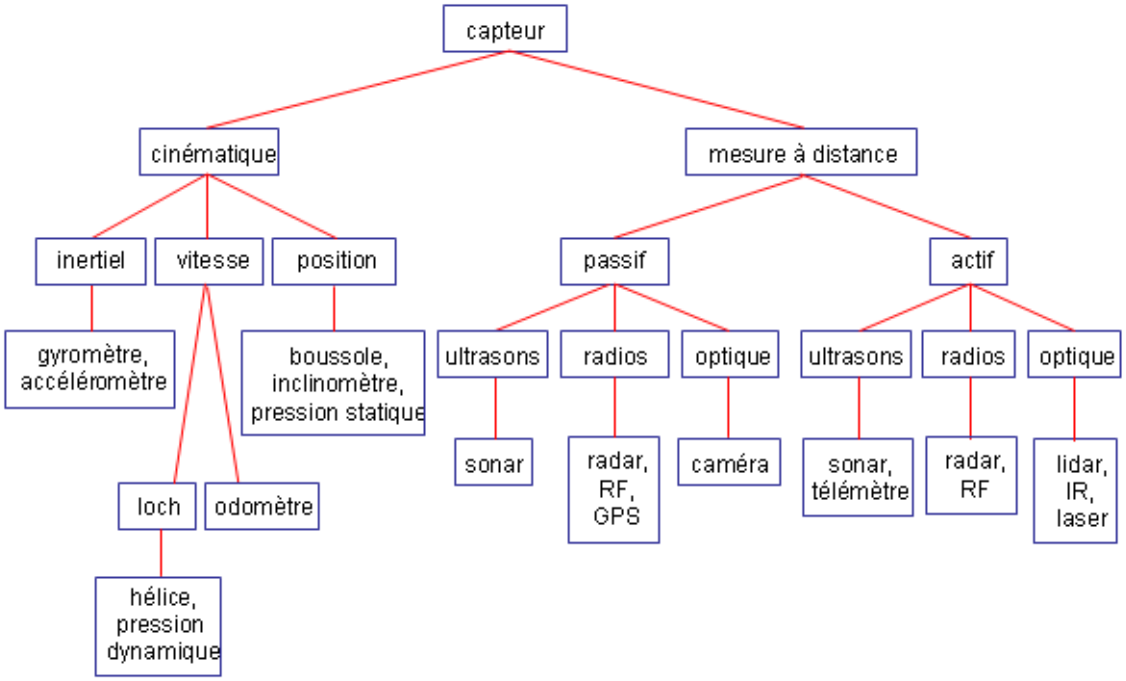


Figure 2.4 – Classification possible de capteurs utilisés couramment sur des robots mobiles.

2.2.1 Le *Redermor* et la *Daurade*, AUVs du GESMA munis d'un sonar latéral

La *Daurade* est un robot sous-marin autonome multi-capteurs construit par ECA et exploité par le GESMA (Groupe d'Etudes Sous-Marines de l'Atlantique) et le SHOM (Service Hydrographique et Océanographique de la Marine). Ce robot se présente sous la forme d'un tube d'une longueur de 5 m pour un diamètre de 0.70 m et pèse environ 1 tonne (voir [Vrignaud and Meyrat, 2009] pour plus d'informations techniques sur ce sous-marin). Il peut naviguer de manière autonome pendant une dizaine d'heures, à une vitesse de 4 nœuds (max 8) et jusqu'à une profondeur de 300 m. Son algorithme d'estimation de position est supposé avoir une erreur inférieure à 50 m après 10 km de mission grâce à ses capteurs très précis et son filtre de Kalman embarqué. De plus, des traitements de données embarqués supplémentaires et des missions demandant une adaptation à l'environnement (évitement d'obstacles,...) peuvent être faites. Ce sous-marin est l'un des successeurs du *Redermor* (*Coureur des mers* en Breton), aux caractéristiques similaires (comme décrit dans [Jaulin et al., 2006]). Les applications de ces AUVs sont principalement militaires : recherche de mines sous-marines, exploration de zones inconnues...

De nombreux capteurs sont intégrés dans la *Daurade*. Voici les principaux :

– DGPS (Differential Global Positioning System). Vu que les ondes électromagnétiques (ici à 1.57542 GHz) ne se propagent pas à travers l'eau, le GPS est uniquement opérationnel lorsque le sous-marin est en surface. Il retourne la latitude l et la longitude L du sous-marin (en degrés). Nous pouvons en déduire les 2 premières coordonnées du robot dans le repère de référence (voir figure 2.1) en utilisant la relation

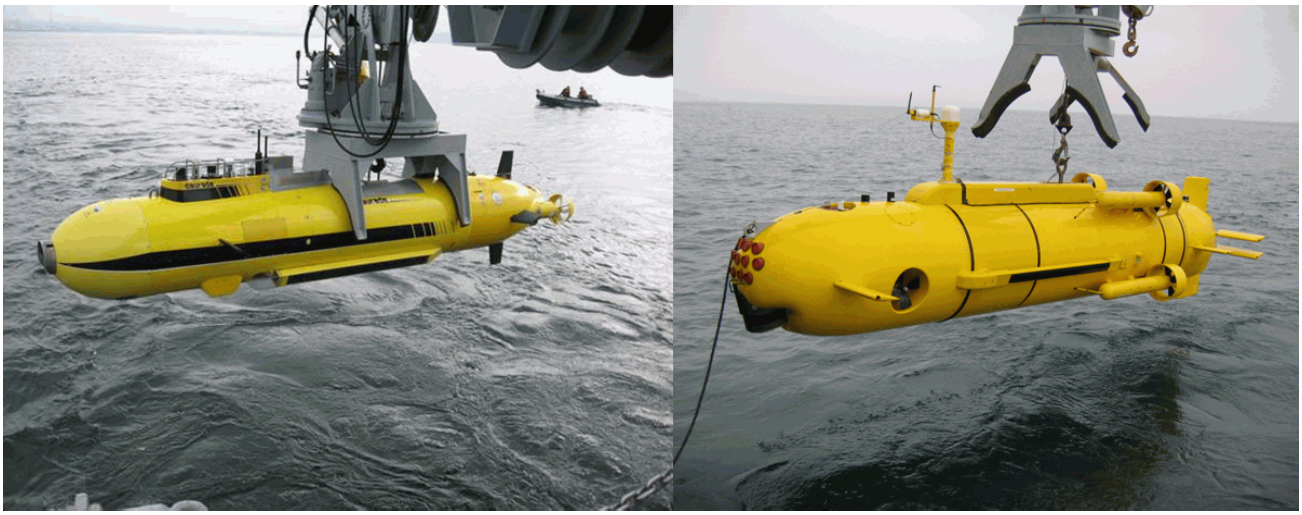


Figure 2.5 – La *Daurade* et le *Redermor* (photos DGA/GESMA).

suivante :

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{\pi}{180}.r_{Earth}.\begin{pmatrix} 0 & 1 \\ \cos\left(\frac{\pi}{180}.l\right) & 0 \end{pmatrix}\begin{pmatrix} L-L^0 \\ l-l^0 \end{pmatrix} \tag{2.2}$$

avec l^0 et L^0 latitude et longitude correspondant à l'origine du repère de référence et $r_{Earth} = 6.371.10^6$ m le rayon de la Terre (supposée sphérique). L'erreur d'estimation de position d'un GPS (de bonne qualité) est en général inférieure à 3 m.

– Sonar latéral Klein 5500 (voir [L-3 Klein Associates, Inc, 2011]). Il permet d'obtenir une image du fond de l'océan (souvent appelée *waterfall* [Glynn, 2007]) avec le logiciel SonarPro (fourni par le constructeur). Cette image a une taille correspondant à environ 75 m de large (ce paramètre est souvent appelé portée ou rangescale du sonar) par plusieurs km de hauteur (ceci correspond à la distance parcourue par le robot pendant toute sa mission). La figure 2.6 montre un sous-marin muni d'un sonar latéral bâbord et tribord en train d'avancer en ligne droite. Un sonar latéral envoie (tous les dixièmes de seconde environ) une onde ultrasonore (souvent appelée *ping*) sur un plan perpendiculaire à l'axe principal du robot. Lorsque cette onde est réfléchie par quelque chose, elle revient vers le sonar avec un retard et une amplitude atténuée, qui permettent d'en déduire la distance de l'objet et une valeur d'intensité (qui peut par exemple donner des informations sur sa constitution...). Lorsque le sous-marin avance, on peut mettre bout à bout tous les pings, pour former une image similaire à la figure 2.6. Les 2 bandes noires verticales (souvent appelées *water column*) correspondent à l'eau entre le sous-marin et le fond. On peut donc en déduire l'altitude du sous-marin (mais cette mesure est en général moins précise que celle fournie par le DVL, voir point suivant). La petite tache noire sur la partie tribord de l'image sonar sur la figure 2.6 est un petit rocher, et les zones ombrées à bâbord correspondent à une épave de bateau et des rides de sable. En bas à gauche, on distingue aussi quelque chose qui ressemble à un câble, au bout duquel il y avait probablement l'ancre du bateau. On peut imaginer que les sonars du robot (en fait il n'y a qu'un sonar, mais qui scanne à droite et à gauche) sont comme des lampes qui éclairent le sol la nuit : on a les mêmes phénomènes d'ombres (sur la partie droite de l'image, les ombres sont à droite des objets et sur la partie gauche, les ombres sont à gauche des objets). La figure 2.7 montre une capture d'image du logiciel SonarPro : on y voit le signal brut reçu par le sonar lorsque le robot est au-dessus de l'épave. Bien que de nombreux traitements

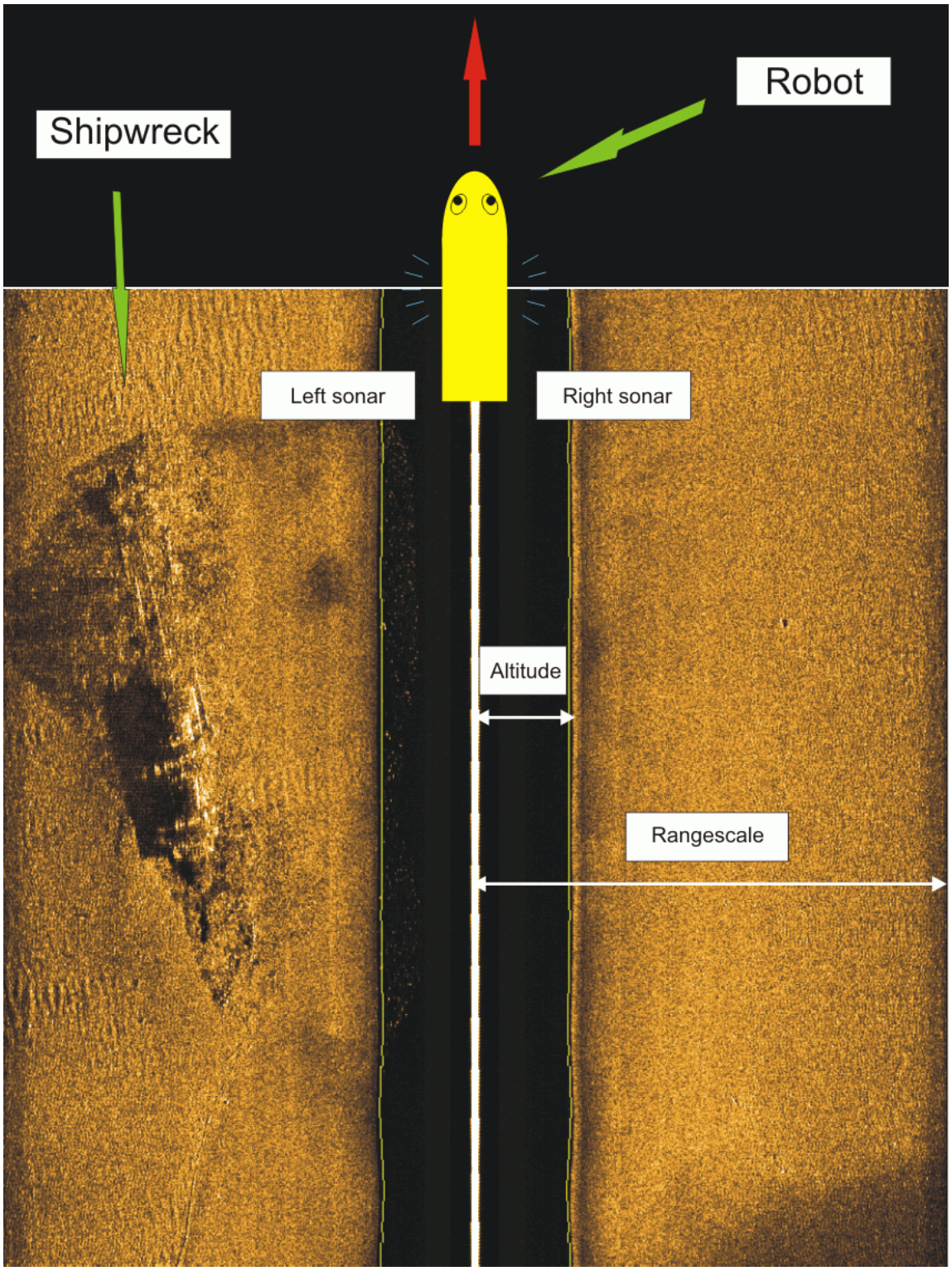


Figure 2.6 – Sous-marin muni d'un sonar latéral bâbord et tribord : au fur et à mesure que le sous-marin avance, on peut générer une image sonar qui correspond à une vue du dessus du fond marin.

soient faits avant de pouvoir obtenir l'image sonar en-dessous, on peut constater une correspondance entre les zones d'ombres dues à la présence de l'épave et une absence d'écho réfléchi. En parcourant l'image,

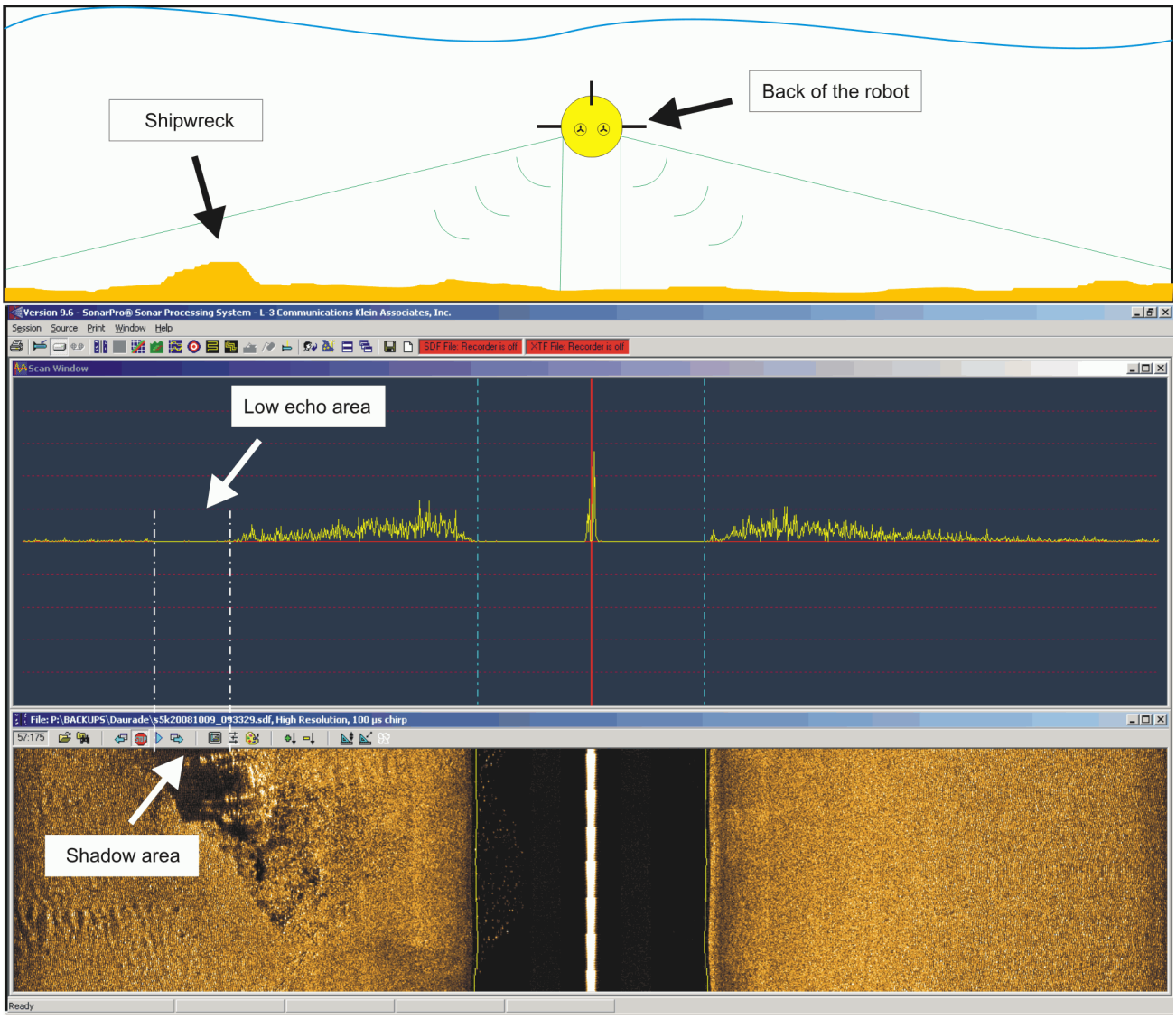


Figure 2.7 – Vue arrière du sous-marin et capture d'écran des résultats correspondants, fournis par le logiciel SonarPro (signal brut du sonar montrant l'intensité de l'écho reçu en fonction de la distance et image).

on peut donc repérer des temps t et distance d_m correspondant à des moments et endroits où un amer i (rocher, épave,...) est visible. Les mêmes amers peuvent être vus plusieurs fois selon la trajectoire que le robot a faite (et c'est d'ailleurs dans ces situations que le SLAM prend tout son intérêt).

– DVL (Doppler Velocity Loch) 1200 kHz type Navigator (voir [Teledyne RD Instruments, 2011]). Ce capteur permet de calculer le vecteur vitesse $\mathbf{v}_r$ du robot dans le repère mobile qui lui est associé. Il émet des ultrasons dirigés vers le fond de l'océan. Comme le fond est immobile, le capteur est capable de calculer une estimation de sa vitesse grâce à l'effet Doppler. Il calcule aussi la distance entre le robot et le fond de l'océan (altitude a).
– IXSEA U-PHINS INS (voir [IXSEA, 2011]). Il retourne les angles d'orientation φ (roll, bank ou roulis),

θ (pitch, elevation, tilt ou tangage),ψ (yaw, heading, pan, azimut ou lacet, cap) et vitesses de rotation $\dot{\varphi}, \dot{\theta}, \dot{\psi}$ du robot.
- Capteur de pression. Il permet d'obtenir la profondeur du robot (*i.e.* la distance entre le robot et la surface de l'océan, qui correspond en général à z).
- Caméra.
- Sonar frontal pour l'évitement d'obstacles.

Le mouvement du robot peut être décrit par l'équation d'état (voir [Jaulin, 2009a] pour plus de détails sur la mise en équation)

$$\dot{\mathbf{p}} = \mathbf{R}(\varphi, \theta, \psi).\mathbf{v}_r \tag{2.3}$$

où :

- $\mathbf{p} = (x, y, z)$ est la position du robot exprimée dans le repère de référence.
- $\mathbf{R}$ est la matrice de rotation formée à partir des angles retournés par l'INS (correspondent à l'orientation du repère mobile par rapport au repère de référence) :

$$\mathbf{R}(\varphi, \theta, \psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

- $\mathbf{v}_r$ représente la vitesse du robot mesurée par le DVL (exprimée dans le repère mobile, d'où la nécessité de la multiplier par $\mathbf{R}(\varphi, \theta, \psi)$ pour l'exprimer dans le repère de référence).

Du fait de l'emplacement du sonar sur le robot, qu'on suppose que les amers sont au fond de l'eau et que pendant la cartographie, le robot subit peu de roulis et tangage, lorsqu'un amer $\mathbf{m}$ est détecté à un instant t sur une image sonar, il est forcément sous le robot et dans un plan qui lui est perpendiculaire et on a :

- $r = \|\mathbf{e}\| = d_m$ la distance robot-amer sur l'image sonar à l'instant t (voir figure 2.8), avec $\mathbf{e} = \mathbf{p} - \mathbf{m}$.

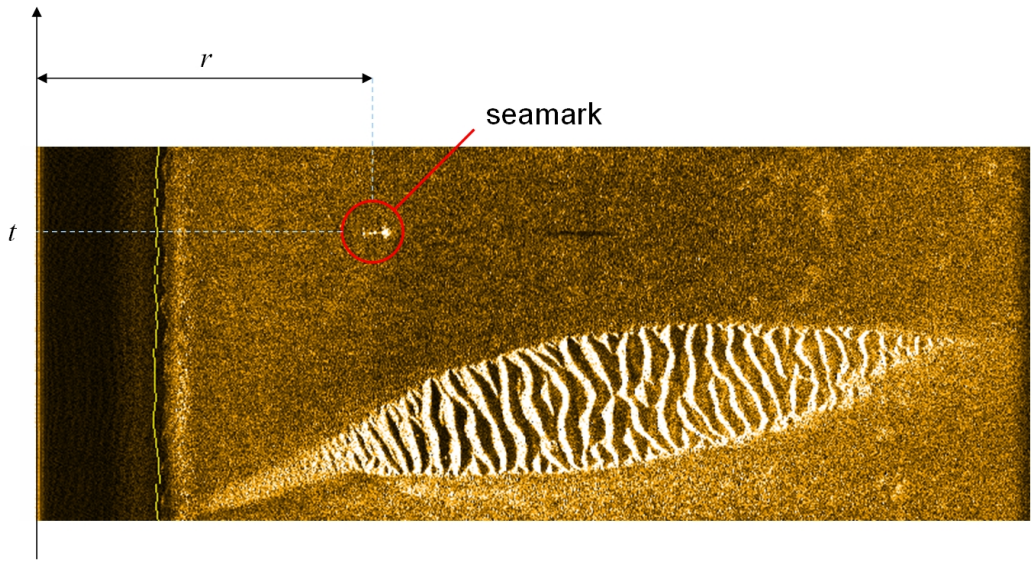


Figure 2.8 – Une mine sous-marine sur une image de sonar latéral tribord, et des rides de sable en dessous.

- $\alpha_m = \pi/2$ si l'amer est détecté sur l'image sonar tribord, $\alpha_m = -\pi/2$ si c'est sur bâbord.
- $\mathbf{R}^{\mathrm{T}}.\mathbf{e} = \left(-\cos(\alpha_m)\sqrt{d_m^2 - a^2}, -\sin(\alpha_m)\sqrt{d_m^2 - a^2}, -a\right)$ en utilisant le théorème de Pythagore (voir figure 2.9) avec $\cos(\alpha_m) = 0$ et $\sin(\alpha_m) = 1$ ou -1 selon que l'amer est à tribord ou bâbord (amer forcément sous le robot et dans un plan perpendiculaire à l'axe principal du robot, $\mathbf{R}^{\mathrm{T}}.\mathbf{e}$ étant l'expression de $\mathbf{e}$ dans le repère mobile associé au robot et a représente la distance entre le robot et le fond mesurée par le DVL, suivant la direction $\mathbf{k}$).

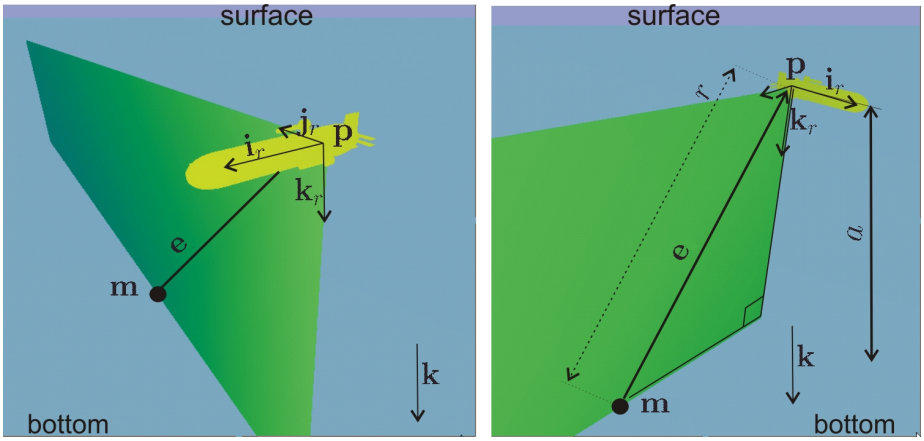


Figure 2.9 – Détection d'une mine sous-marine par le sonar latéral tribord d'un sous-marin.

- $z_m = z + a$ (amer supposé au fond de l'eau, qu'on suppose plat localement).

2.2.2 *SAUC'ISSE*, AUV de l'ENSTA Bretagne muni d'un sonar rotatif

SAUC'ISSE (SAUC-E Interval Super Submarine of ENSTA Bretagne, voir figure 2.10) est un robot sous-marin autonome construit en 2007 par des étudiants de l'ENSTA Bretagne pour le concours SAUC-E (Student Autonomous Underwater Challenge - Europe). Cette compétition, organisée chaque année depuis 2006 dans différents pays d'Europe (Angleterre en 2006, 2007 et 2009, France en 2008 et Italie en 2010, l'AUVSI organise des concours similaires aux Etats-Unis) a pour but de faire réaliser par des étudiants des robots sous-marins capables d'effectuer des successions de missions de localisation et détection d'objets dans une piscine ou marina, de manière complètement autonome. En 2010, les missions du concours étaient les suivantes (voir figure 2.13) :

- Partir de Start 1 ou Start2 et passer dans le cadre Gate 1. Des points supplémentaires étaient attribués si on partait de Start 1. Un sous-marin incapable de passer à travers Gate 1 était disqualifié.
- Faire demi-tour en repassant à travers Gate 1, suivre le pipeline jaune (voir figure 2.11) au fond de l'eau en restant à moins de 50 cm de lui puis passer à travers Gate 2.
- Couper le fil maintenant une bouée orange (voir figure 2.12) dans l'eau.
- Suivre le mur du coin de la piscine.
- Tourner autour de Start 2, où se trouvait une balise acoustique
- Faire surface au-dessus de la balise.
- Produire un fichier log (appelé aussi fichier journal) ayant enregistré les temps, positions et actions pendant toute l'épreuve.

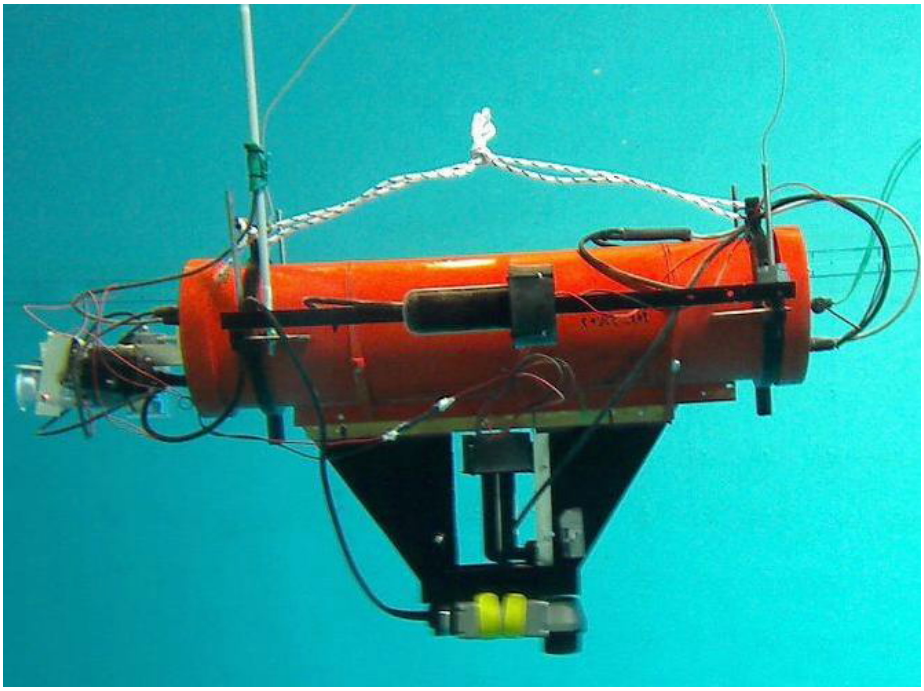


Figure 2.10 – *SAUC'ISSE* dans une piscine de l'IFREMER (Brest, France).



Figure 2.11 – Pipeline jaune au fond de la zone de compétition.



Figure 2.12 – Bouée orange dans la zone de compétition.



Figure 2.13 – Missions du concours SAUC-E 2010, à La Spezia en Italie.

Le maximum de points était attribué lorsque des missions avaient été faites les unes après les autres mais il était quand même possible de gagner des points en effectuant des missions individuellement. L'ENSTA Bretagne est arrivée $3^{ème}$ sur 9 au concours SAUC-E 2010, grâce à ses 2 robots *SAUC'ISSE* et *SARDINE* (ce dernier sous-marin est une version simplifiée de *SAUC'ISSE* avec des caractéristiques similaires, voir [Sliwka et al., 2011a]).

SAUC'ISSE [Le Bars et al., 2010b] est basé sur un tube en aluminium de 17 cm de diamètre par 70 cm de long. L'étanchéité du tube est réalisée par 2 tapes (sortes de couvercles) en aluminium munies de connecteurs étanches pour relier les différents actionneurs et capteurs extérieurs avec l'intérieur du tube (ces connecteurs étanches sont les éléments qui limitent la profondeur d'opération du sous-marin à 10 m). Les angles de roulis et tangage du sous-marin ne sont pas contrôlés car ils restent stables grâce à une quille lestée, qui sert aussi de support pour le sonar et le propulseur vertical. L'équilibrage du sous-marin pour atteindre la limite de flottaison (qui change selon les conditions d'utilisation) est ajusté en rajoutant des morceaux de plomb aux 4 coins du robot. En conséquence, le propulseur vertical n'a besoin de produire qu'une force faible pour que le sous-marin descende en profondeur et lorsqu'il est éteint, le robot remonte tout seul à la surface. A l'intérieur du tube (voir figure 2.14), des rails collés à l'intérieur du tube en aluminium permettent de faire glisser une plaque de Plexiglas qui sert de support principal pour les éléments intérieurs de *SAUC'ISSE*. Un tiroir monté sur la plaque contient les batteries. Ainsi, on peut facilement accéder aux batteries sans

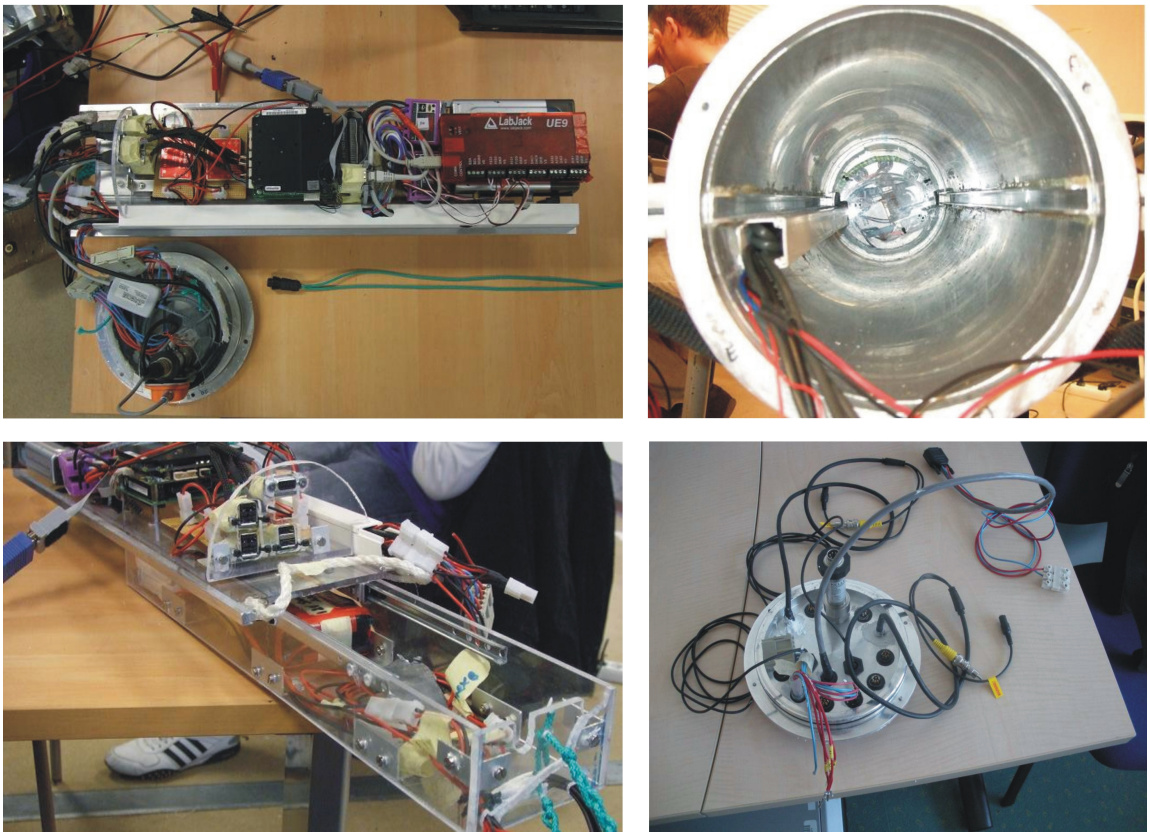


Figure 2.14 – Organisation intérieure de *SAUC'ISSE*.

avoir besoin de déplacer les autres éléments, qui sont fixés à l'aide de scratchs sur la plaque de Plexiglas

principale. L'organisation électronique du sous-marin est décrite sur la figure 2.15.

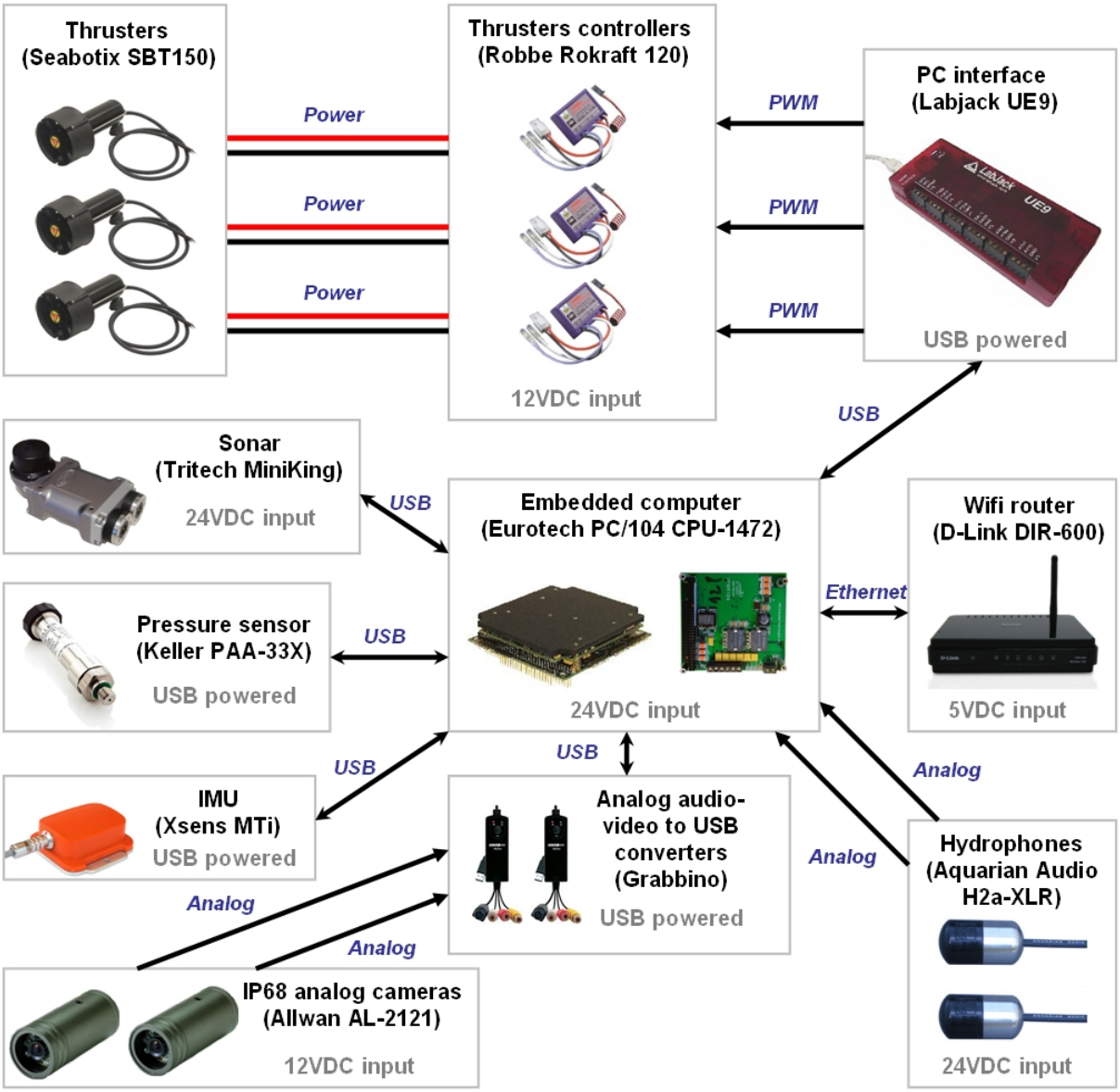


Figure 2.15 – Capteurs et actionneurs de *SAUC'ISSE*.

3 propulseurs DC BTD150 de SEABOTIX, un fabricant américain de ROVs (Remote Operated Vehicles) permettent au robot de se déplacer :

– 1 propulseur vertical ajuste la profondeur du véhicule.
– 2 propulseurs horizontaux contrôlent la vitesse et la direction.

Ces propulseurs sont étanches jusqu'à 150 m.

Pour contrôler la vitesse des propulseurs à partir de signaux électroniques, nous utilisons des variateurs de modélisme Robbe Rokraft. La puissance envoyée aux propulseurs (et donc leur vitesse) dépend de la forme du signal PWM (Pulse Width Modulation) que reçoivent les variateurs. Pour générer ces signaux PWM à partir de programmes fonctionnant sur l'ordinateur embarqué, nous avons besoin d'un module d'interface entre l'ordinateur et le variateur : le Labjack UE9. C'est un périphérique USB (ou Ethernet) fournissant de nombreuses entrées-sorties pouvant être facilement connectées à divers périphériques électroniques.

L'ordinateur embarqué dans *SAUC'ISSE* est un PC/104 d'EUROTECH avec un processeur Pentium M à 1.4 GHz et 512 Mo de RAM. Le système d'exploitation et les programmes sont stockés sur un disque dur 2.5 pouces de 320 Go. 8 ports USB, 1 port Ethernet, 2 ports RS232 et un port VGA permettent d'y connecter toutes sortes de périphériques et de communiquer avec l'ordinateur de manière classique. Il peut être directement alimenté à partir de batteries 12 ou 24 V grâce à un étage d'alimentation au format PC/104 qui fournit des tensions de +3.3, +5 et +12 V régulées.

Des caméras analogiques étanches (jusqu'à 50 m) ALLWAN AL-2121 sont connectées au PC/104 via des connecteurs étanches (elles sont placées à l'extérieur du tube, une orientée vers l'avant et une autre orientée vers le bas) et des convertisseurs Grabbino audio-vidéo analogique vers USB.

Pour obtenir la profondeur du sous-marin, un capteur de pression Keller PAA33X connecté au PC embarqué via un convertisseur RS485 vers USB est fixé sur la tape arrière.

Une centrale inertielle Xsens MTi prêtée par le GESMA (Groupe d'Etudes Sous Marines de l'Atlantique) permet d'obtenir l'orientation de *SAUC'ISSE* par rapport au Nord (le seul angle de rotation qui nous intéresse est le cap car les autres devraient rester stables du fait de la quille du sous-marin) de façon assez précise et robuste grâce à son filtre de fusion intégré qui utilise ses magnétomètres (boussole 3D) et ses gyromètres (capteurs inertiels) pour fournir une orientation correcte même en cas de perturbations magnétiques (cependant des corrections doivent parfois être apportées comme indiqué en 3.4.3). Elle est connectée au PC embarqué via un convertisseur RS422 vers USB. Une MTi-G (comme la MTi, mais avec un GPS en plus et un convertisseur RS232 vers USB) peut aussi être installée à la place (cependant cette dernière semble moins précise que la MTi lorsque le GPS n'est pas disponible, ce qui est le cas lorsque le sous-marin plonge sous l'eau).

Pour obtenir sa position par rapport aux bords de la zone où il évolue ou détecter des objets, *SAUC'ISSE* a un sonar Tritech MiniKing (prêté par d'autres personnes de L'ENSTA Bretagne) connecté au PC embarqué en RS232 via un connecteur étanche. Contrairement au sonar latéral de la *Daurade* qui envoie des pings à bâbord et tribord perpendiculairement à l'axe principal du robot, le sonar de *SAUC'ISSE* est rotatif : il envoie ses pings à différents angles α. Alors qu'une image de sonar latéral devient facilement interprétable lorsque le sous-marin avance en ligne droite à vitesse constante (les données sonar forment alors l'image du fond de l'eau), une image de sonar rotatif est plus facilement interprétable lorsque le sous-marin est immobile : on y voit alors se dessiner les bords de la piscine où évolue le sous-marin, ainsi que les objets qui sont entre 2 eaux (figure 2.16). Ce sonar a une portée de 100 m.

Un routeur Wifi D-Link DIR-600 associé à une antenne Wifi externe de 1 m nous permettent de communiquer avec *SAUC'ISSE* via un PC portable quand il est près de la surface de l'eau (*i.e.* lorsque l'antenne sort de l'eau). Si on doit communiquer avec le sous-marin à de plus grandes profondeurs, on utilise un déport d'antenne de 5 m (avec les connecteurs étanches SMB Bulgin Buccaneer et du câble RG174). De plus,

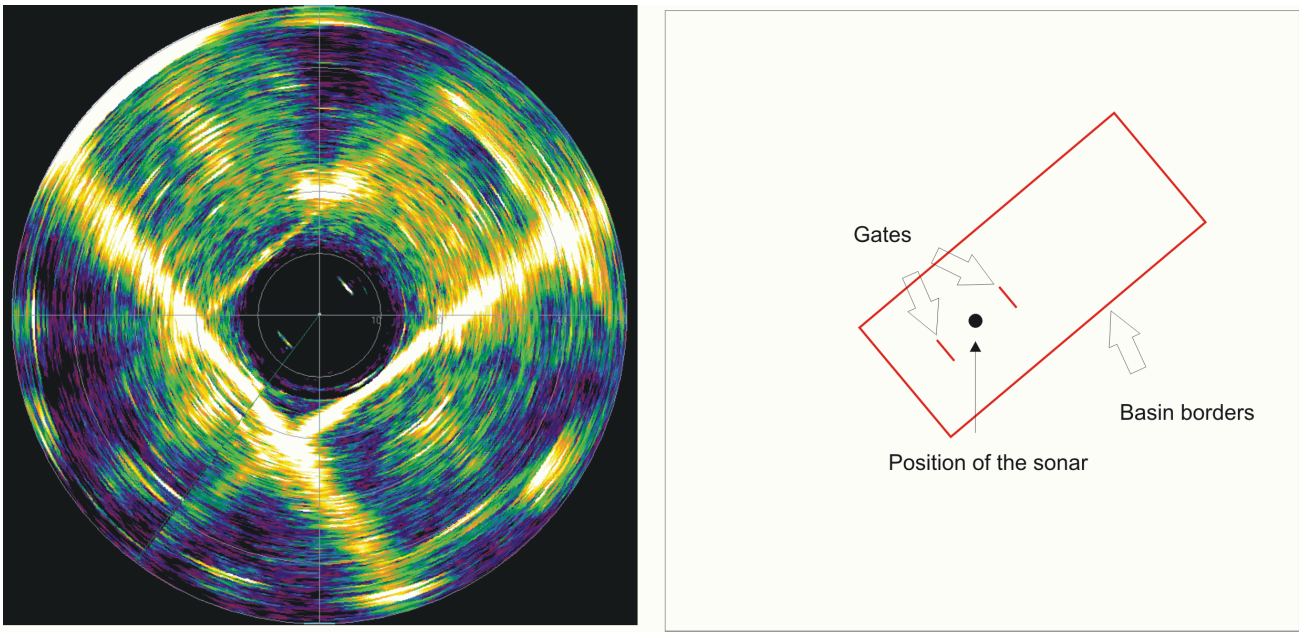


Figure 2.16 – Image sonar de la zone de compétition pendant SAUC-E 2009.

SAUC'ISSE a été équipé d'un port Ethernet étanche sur sa tape arrière (avec un connecteur étanche Switchcraft 8 pins) qui nous permet de brancher un câble Ethernet de 30 m relié à un PC portable sur terre ou une bouée flottante avec un point d'accès Wifi et une batterie. Nous utilisons la « Connexion au bureau à distance » intégrée dans Windows XP pour accéder au bureau du PC embarqué via Wifi ou Ethernet (figure 2.17). Une fois connecté, on peut alors lancer toutes les applications voulues sur le PC embarqué,

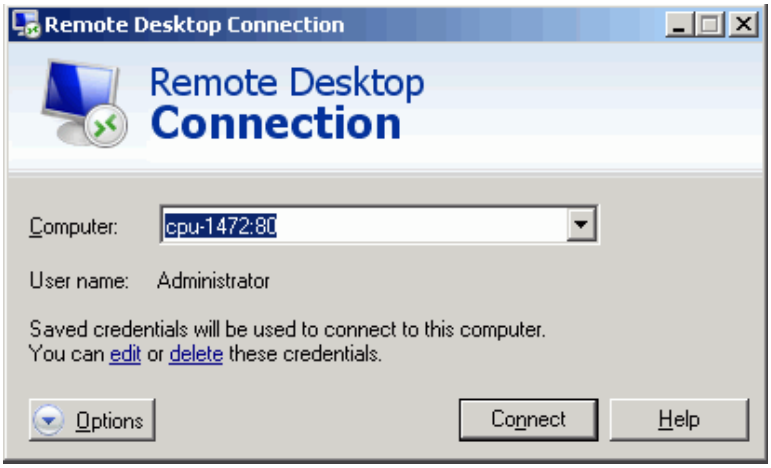


Figure 2.17 – Connexion au bureau à distance.

vérifier si tous les périphériques sont correctement détectés... comme sur n'importe quel PC. L'avantage est qu'il n'est pas nécessaire de gérer la partie réseau dans le programme embarqué dans le robot, ceci est déjà géré par Windows (ceci pourrait aussi être fait sous Linux, avec les protocoles RDP, VNC ou serveur X via

SSH...).

L'alimentation de *SAUC'ISSE* est divisée en 2 parties :

- Les propulseurs sont alimentés par une batterie de 12 V.
- Le PC/104, le sonar, le routeur Wifi (via le 5 V fourni par l'étage d'alimentation du PC/104) et les caméras (via le 12 V fourni par l'étage d'alimentation du PC/104) sont alimentés par une batterie de 24 V (ou 2 batteries de 12 V en série).

Tous les autres périphériques (capteur de pression, centrale inertielle, Labjack) sont alimentés par le 5 V fourni par les ports USB du PC/104. Il faut noter qu'actuellement (contrairement à ce qui pourrait être pensé au premier abord), c'est le PC embarqué et ses capteurs qui consomment le plus d'énergie (et non les actionneurs).

Des algorithmes de régulation en profondeur, cap, vitesse et distance classiques associés à d'autres algorithmes de détection, localisation et contrôle plus innovants forment l'intelligence du robot (figure 2.18).

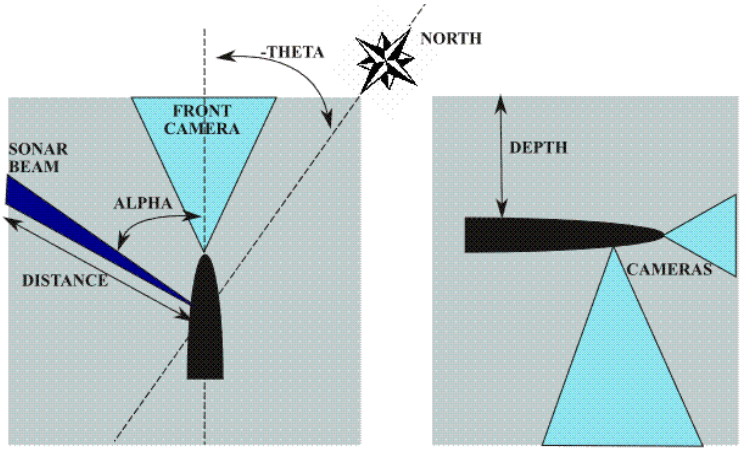


Figure 2.18 – Localisation et détection.

L'une des principales difficultés pour le robot est de pouvoir se localiser dynamiquement par rapport aux bords de la piscine où il évolue. Un observateur d'état utilisant le calcul par intervalles et prenant en compte des données aberrantes venant du sonar rotatif du sous-marin lui permet d'estimer sa position. De plus, des méthodes originales supplémentaires mettant en jeu un *contracteur sur l'image* (voir [Sliwka et al., 2011b]) et une notion de polynômes ensemblistes (voir [Sliwka, 2011]) viennent d'être élaborées et testées sur le sous-marin dans la piscine de l'ENSTA Bretagne. Le robot peut :

- Etre téléopéré de différentes façons en utilisant les algorithmes de régulation et contrôle voulus : pas de régulation (boucle ouverte), régulation en profondeur et régulation en cap et vitesse dans le plan...
- Effectuer des enchaînements de missions prédéfinies sous forme de scripts (fichiers texte avec quelques mots-clés prenant des paramètres) : successions et combinaisons entre régulations en profondeur, cap, vitesse ou distance dont les consignes changent au cours du temps ou appel à des fonctions particulières telles que l'activation du suivi de waypoints par localisation dynamique, mur, pipeline jaune ou boule orange...

Dans les simulations, les équations d'état du robot sous-marin ont été décrites de cette façon :

$$\begin{cases} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= u_2 - u_1 \\ \dot{v} &= u_1 + u_2 - v \end{cases} \tag{2.5}$$

où (x, y) sont les coordonnées du robot, θ, son orientation et v sa vitesse. Les entrées u_1 et u_2 sont les forces de propulsion fournies par les propulseurs droite et gauche. Ce modèle normalisé correspond à un robot sous-marin à une profondeur constante (la régulation en profondeur du robot est indépendante et est résolue par une commande dite bang-bang : si le sous-marin est trop profond, on le laisse remonter naturellement ou on actionne son propulseur vertical pour qu'il remonte s'il est loin de la profondeur voulue, autrement on actionne son propulseur vertical pour le faire descendre) sans roulis ni tangage et contrôlé directement via la force de poussée de ses 2 propulseurs horizontaux, avec prise en compte des forces de frottement fluides dues à l'eau (proportionnelles à la vitesse du sous-marin et le ralentissant). De cette manière, notre sous-marin peut être considéré comme un robot évoluant en 2 dimensions. Pour prendre en compte la présence de données aberrantes, nous considérons qu'une pose du robot est valide si elle est consistante avec toutes les mesures faites sauf q d'entre elles (voir *q-intersection* dans [Jaulin, 2009b] et [Sliwka et al., 2009]).

Le principe du contrôleur utilisé pour le suivi de waypoints est décrit dans la figure 2.19. D'abord, le

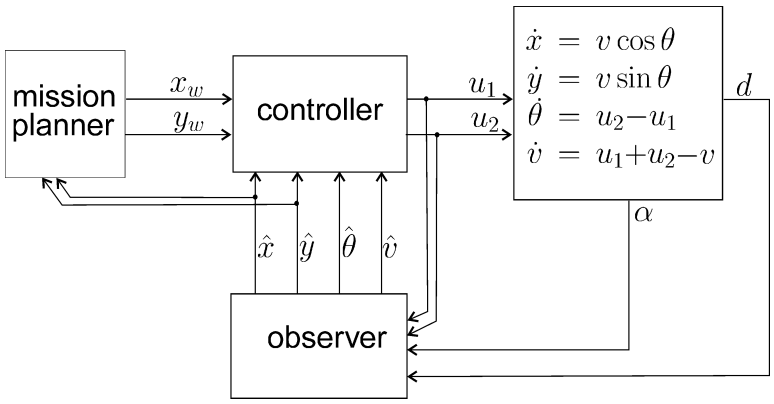


Figure 2.19 – Principe du contrôle du robot sous-marin.

planificateur de mission envoie au contrôleur un waypoint (x_w, y_w) à atteindre. Lorsque le waypoint courant est atteint avec une précision prédéfinie (*i.e.*, $(\hat{x} - x_w)^2 + (\hat{y} - y_w)^2 \leq \varepsilon$), le planificateur passe au waypoint suivant. Le contrôleur choisi est donné par l'expression suivante :

$$\mathbf{u} = \begin{pmatrix} 1-\omega \\ 1+\omega \end{pmatrix}, \text{ où } \omega = \text{sign}\left(\det\begin{pmatrix} \cos\hat{\theta} & x_w - \hat{x} \\ \sin\hat{\theta} & y_w - \hat{y} \end{pmatrix}\right). \tag{2.6}$$

Le principal avantage de ce contrôleur est sa simplicité. La direction que le robot doit suivre est donnée par le vecteur $\mathbf{e} = (x_w - \hat{x},\ y_w - \hat{y})$. L'orientation estimée du robot est donnée par le vecteur $\mathbf{v} = \left(\cos\hat{\theta},\ \sin\hat{\theta}\right)^{\mathrm{T}}$. Si $\mathbf{v}$ est à la droite de $\mathbf{e}$ (*i.e.* $\det(\mathbf{v}, \mathbf{e}) < 0$), le robot tourne à droite ($\omega = 1$) autrement il tourne à gauche ($\omega = -1$).

Les actions de haut niveau telles que se déplacer à un point de coordonnées connues, suivre une trajectoire, rechercher un objet, suivre un bord de la piscine... peuvent être effectuées en exécutant les algorithmes de traitement suivis des algorithmes de déplacement de base selon leurs résultats. Ces actions sont implémentées dans le programme du robot en C/C++. Par contre, l'enchaînement de ces actions est spécifié sous forme de script (figure 2.20). Cela permet de changer rapidement et facilement les missions que doit effectuer le

```
% this is a commentary. It is ignored by the program
% Example 1: go to depth -3m
%% step1: start depth regulation
depthreg -3
%% step2: wait 15s for the robot to actually get there
wait 15
% Example 2: some random movements at -3m
heading 1
wait 5
thrust 1
wait 10
stop
heading 1.57
wait 10
% Example 3: high level action
configureWallFollowing 12.5 1.2 -1.57 0 0.3 0.2 0.5
startWallFollowing
wait 20
stopWallFollowing
generalstop
depthreg 0
```

Figure 2.20 – Exemple de script de mission autonome.

robot et peut de plus être compris et pris en main par un non-spécialiste.

Les caméras et le sonar peuvent fournir des images pouvant être traitées pour détecter des objets, obtenir des informations de mouvement et de position... Différents algorithmes ont été prévus :

– Détection d'objet selon la couleur [Bazeille, 2008] (figure 2.21). Certains objets peuvent être caractérisés

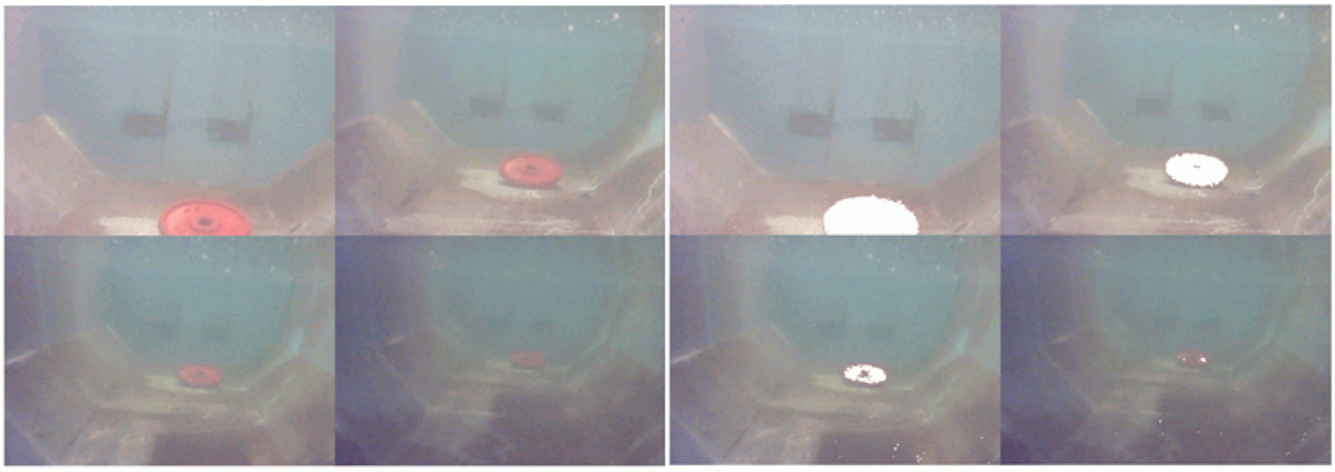


Figure 2.21 – Détection d'objets selon la couleur en prenant en compte l'absorption des couleurs dans l'eau selon la distance.

par leur couleur. Cependant, les couleurs sont altérées dans l'eau : le rouge est plus absorbé que le bleu

lorsqu'on s'éloigne d'un objet. Ces modifications peuvent être modélisées par une formule relativement simple, qui donne la couleur sous laquelle apparaitra un objet dans l'eau selon la distance qui nous sépare de lui, connaissant sa couleur dans l'air. Des coefficients dans la formule (coefficients d'absorption du rouge, du vert et du bleu) doivent être déterminés auparavant par des expériences pour s'adapter à la piscine et aux conditions de luminosité.
- Détection de formes simples (lignes, cercles, rectangles,) avec par exemple la transformée de Hough classique et son équivalent par intervalles [Jaulin and Bazeille, 2009].
- Détection de mouvement : si le sous-marin est immobile et que l'on sait que l'objet cible bouge, on peut alors comparer des images successives pour voir ce qui a changé.
- Détection de points d'intérêts (lignes, objets, parties d'objet ou formes quelconques singulières et détectables dans plusieurs images consécutives) pour obtenir la vitesse du robot en observant le défilement du sol (flux optique) avec une caméra regardant vers le bas et connaissant son altitude.

2.3 Exemples de robots terrestres, sortes d'équivalents terrestres des sous-marins

Des résultats de simulations de robots terrestres seront présentés en 4.4 et 5.2. Voici des exemples de robots utilisés au cours de la thèse qui correspondent aux modèles utilisés. Cette section permettra aussi de mettre en évidence les équivalences entre certains capteurs sous-marins et aériens, ainsi que de montrer rapidement que les méthodes décrites dans ce document (SLAM par intervalles, formalisation avec données fugaces...) sont applicables à des robots différents.

2.3.1 Les JOGs de l'ENSTA Bretagne, munis de télémètres sonar et infrarouges

Les JOGs ([Reynet et al., 2010], voir figure 2.22) sont des robots roulants destinés à l'enseignement des systèmes embarqués à base de processeur ARM (fonctionnant sous Linux et avec Java et Python) et FPGA (Field Programmable Gate Array). Ces robots sont munis d'une carte opensource Armadeus APF27 (incluant le processeur ARM et le FPGA) qui permet de connecter de nombreux capteurs et actionneurs (figure 2.23) :

- 2 moteurs. Le robot possède 2 roues motrices (et une roue folle).
- 2 odomètres (1 pour chaque roue motrice).
- 5 télémètres sonar I2C SRF02 qui ont une portée entre 20 et 200 cm pour détecter les murs ou obstacles. Ces télémètres se comportent de manière similaire à un sonar latéral, même s'ils sont moins élaborés.
- 4 télémètres infrarouges analogiques GP2D120, d'une portée entre 0 et 40 cm, utilisés principalement pour détecter des obstacles en complément des télémètres sonar.
- Une boussole I2C CMP03 pour obtenir son orientation.
- Un micro pour se localiser par acoustique dans une pièce avec 4 haut-parleurs (un dans chaque coin).

Le modèle d'évolution d'un JOG peut être considéré comme celui d'un char :

$$\begin{cases} \dot{x} &= u_1 \cos\theta \\ \dot{y} &= u_1 \sin\theta \\ \dot{\theta} &= u_2 \end{cases} \tag{2.7}$$

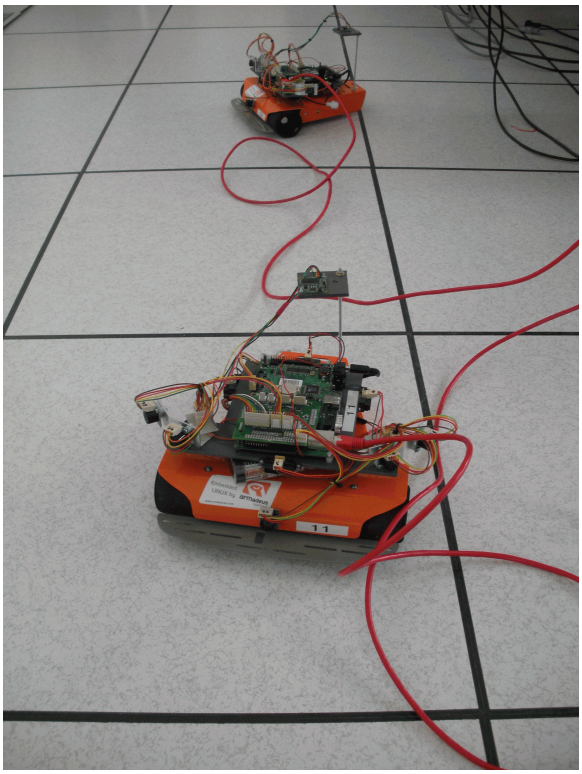


Figure 2.22 – Robots JOGs.

avec u_1 la vitesse dans l'axe principal du robot et u_2 sa vitesse de rotation.

2.3.2 Les robots BOURRICOTs de l'ENSTA Bretagne, munis d'un télémètre laser rotatif

Pour le concours CAROTTE (CArtographie par RObot d'un TErritoire) 2010, une meute de robots roulants terrestres (figure 2.24) a été réalisée à l'ENSTA Bretagne (projet BOURRICOT pour BOUnding Robust and Reliable Interval approach for CarOTte). L'objectif était d'explorer et cartographier un bâtiment inconnu à l'aide de robots autonomes (figure 2.25). Pour mener à bien cette mission, nos robots étaient constitués des éléments suivants (figure 2.26) :

- Plateforme mécanique Lynxmotion A4WD1 à 4 roues munies de 4 moteurs DC 12 V, dont un avec un odomètre. Cet odomètre envoie 2 signaux PWM déphasés, dont le Labjack U3 peut compter le nombre de front montants, qui correspond au nombre de tours de roue. Dans la configuration où nous étions, il n'était possible de récupérer qu'un seul signal, ce qui nous empêchait de connaître le sens dans lequel tournaient les roues (il pouvait cependant être connu en prenant le signe de la commande envoyée).
- 2 cartes de puissance Robbe Rokraft 120, qui contrôlent les moteurs selon les signaux PWM qu'elles reçoivent.
- PC de la classe des ultraportables ASUS EeePC T91.
- Webcam Logitech Quickcam/Webcam Pro 9000.

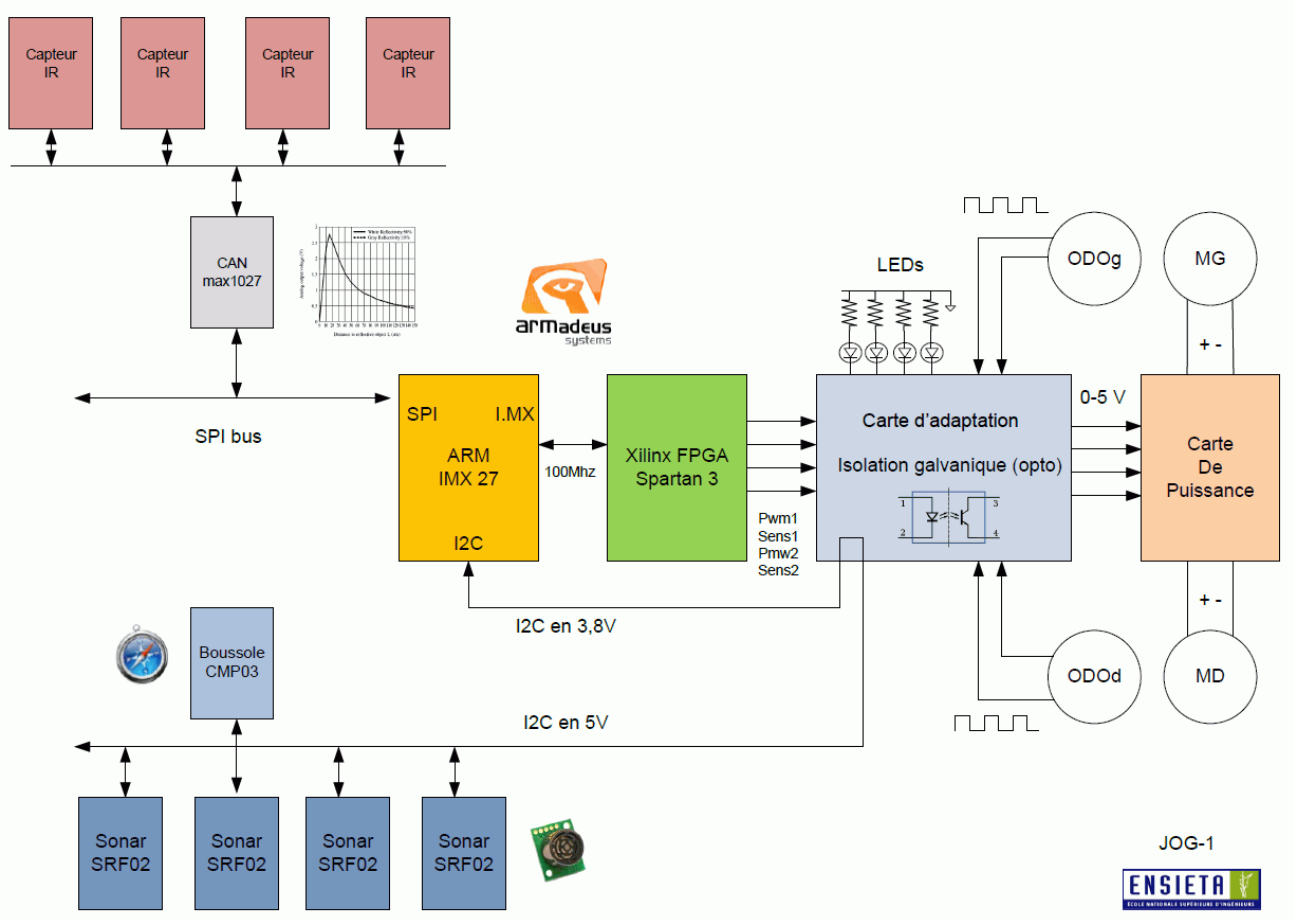


Figure 2.23 – Capteurs et actionneurs des JOGs.



Figure 2.24 – BOURRICOT, la meute de robots de l'ENSTA Bretagne pour le concours CAROTTE 2010.

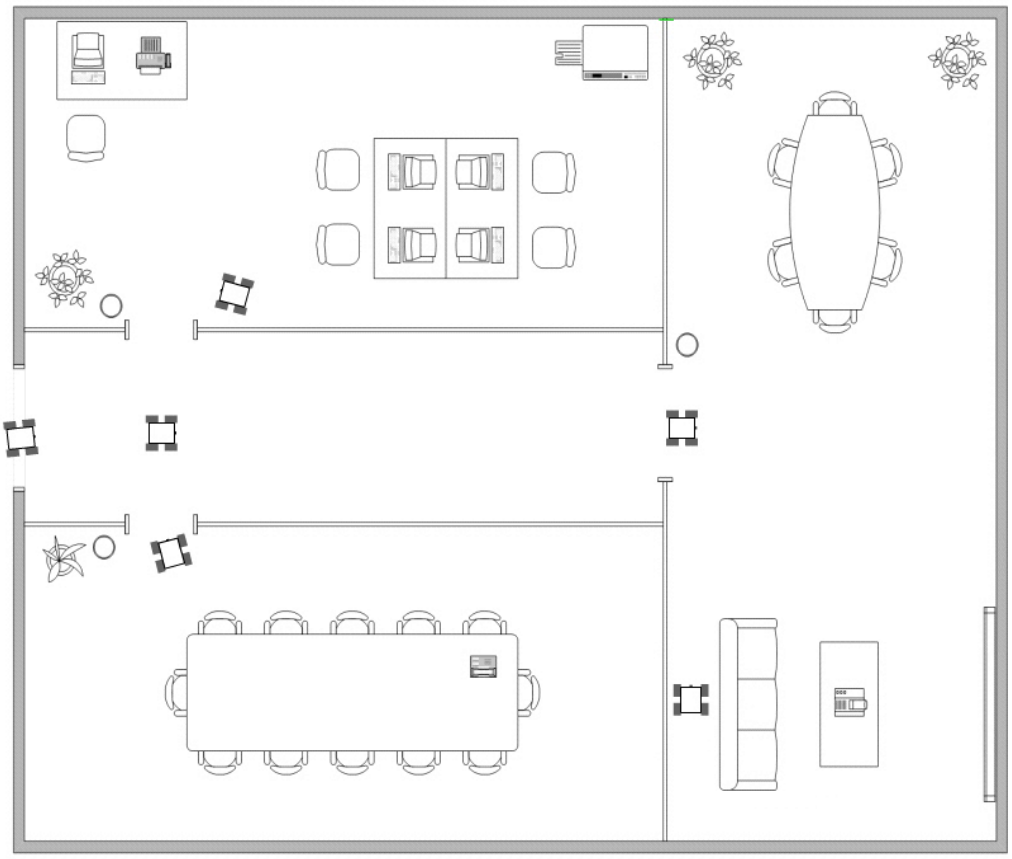


Figure 2.25 – Meute de robots cartographiant un bâtiment.

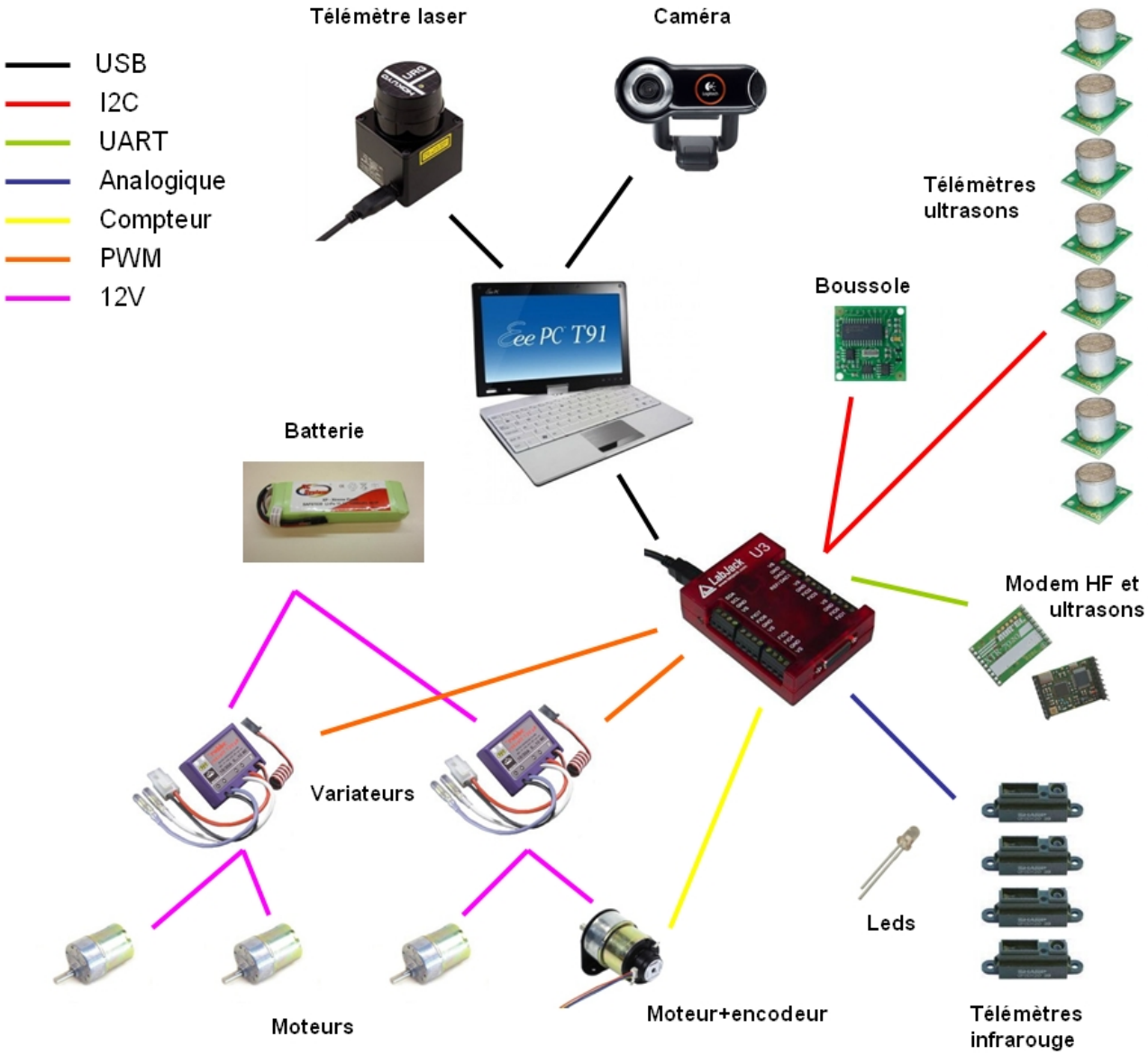


Figure 2.26 – Capteurs et actionneurs d'un robot BOURRICOT.

– Télémètre laser Hokuyo URG-04-LX-UG01. C'est un télémètre rotatif bon marché avec une ouverture angulaire de 240 °, une résolution de 0.35 ° et une portée de 5.6 m. Il permet d'obtenir une image représentant les murs de la pièce (ainsi que les objets) où il se trouve. Sur la figure 2.27, on voit en rouge les murs d'une petite pièce presque rectangulaire (environ 4.5 m de large), les zones bleues indiquant les données n'ayant pas changées depuis le dernier scan, les jaunes indiquant celles qui ont changées (il y a du jaune car le télémètre vibrait un peu).

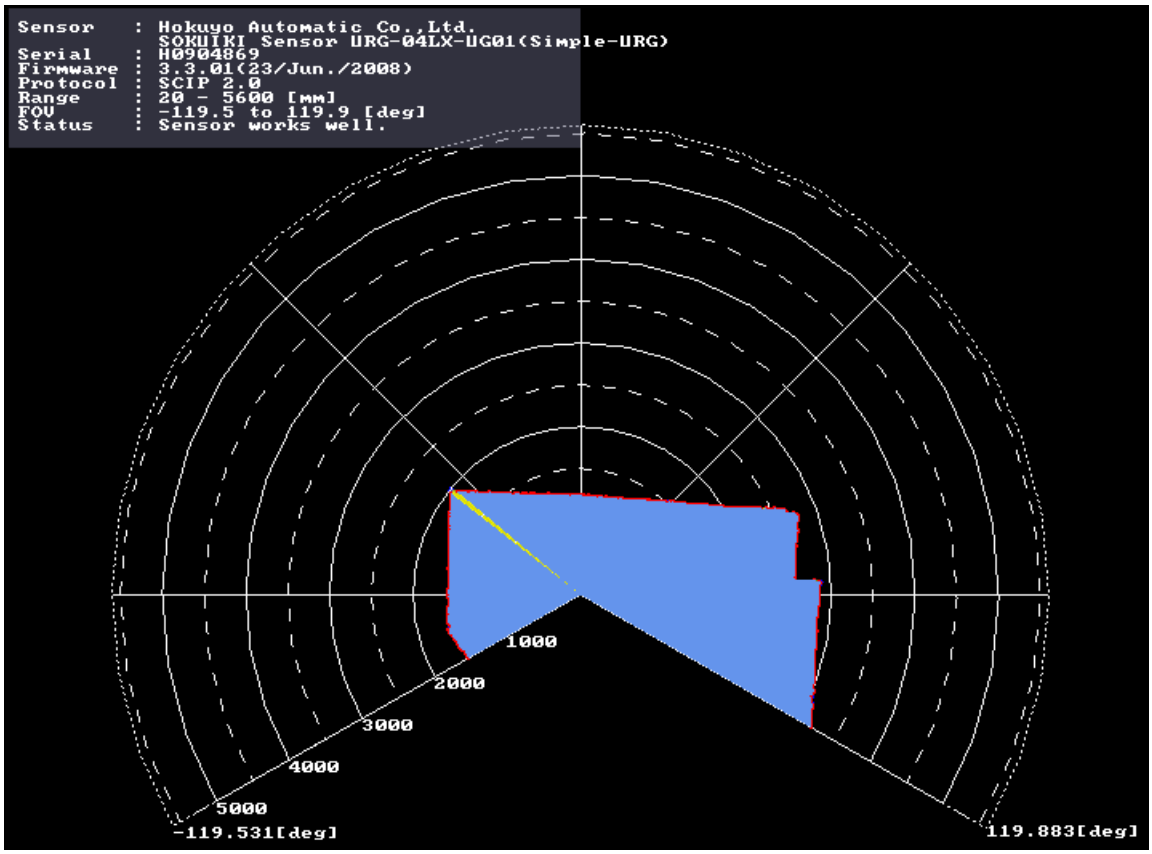


Figure 2.27 – Image fournie par un télémètre laser rotatif.

– Boitier d'interface Labjack U3, qui permet au PC de communiquer avec les périphériques non USB (cartes de puissance, odomètre, boussole, télémètres infrarouges et ultrasons, LEDs, modules de communication ultrasons et HF).
– Boussole CMP03, pour obtenir l'orientation du robot.
– 4 télémètres infrarouge Sharp GP2D120, utilisés pour la détection d'obstacles proches, d'une portée de 4 à 30 cm.
– 8 télémètres ultrason MSU02, utilisés pour la détection d'obstacles entre 15 cm et 6 m. Ces télémètres peuvent compléter les informations renvoyées par le télémètre laser (qui pourtant a une gamme de portée similaire) car certains obstacles, comme les vitres par exemple, peuvent ne pas être détectés par le télémètre laser (le laser ne se réfléchit pas sur la vitre) alors que les télémètres ultrasons le détecteraient correctement.
– Module de communication RF et ultrason (fonctionnant à une fréquence différente de celle des télémètres).
– 3 LEDs clignotantes de couleurs différentes (bleu en haut, vert en bas à droite et rouge en bas à gauche).

La méthode de déplacement et localisation distribuée (voir [Reynet, 2011]) était basée sur les principes suivants :

- Pendant 10 s, tous les robots se déplacent puis s'arrêtent.
- A tour de rôle, chaque robot émet simultanément un signal RF et ultrason, fait clignoter ses LEDs et prend des données avec son télémètre laser et ses télémètres ultrasons. Pendant ce temps-là, les autres robots sont en position d'attente des signaux RF et ultrason et utilisent leurs caméras pour essayer de voir les LEDs clignotantes du robot en mode émission (en supposant qu'ils se soient orientés vers l'endroit où ils estiment que ce robot devrait être compte tenu des informations qu'ils ont actuellement, notamment grâce à sa boussole et son odomètre).
- Les données et équations, calculs et résultats des calculs sont partagés par Wifi entre les robots (en réseau adhoc amélioré grâce au protocole OLSR [olsrd, 2011], qui permet notamment à chaque robot de servir de relais entre 2 robots qui seraient trop éloignés pour pouvoir communiquer directement).

Ainsi, plusieurs méthodes de localisation étaient utilisées et combinées :

- L'odomètre et la boussole pouvaient donner une idée du déplacement d'un robot et pouvait donc déjà donner une position approximative.
- Vision. Lorsqu'un robot voyait avec sa caméra les 3 LEDs clignotantes d'un autre, il pouvait évaluer la distance qui les séparait (grâce à la distance entre les LEDs du bas et la LED du haut sur le robot observé), l'angle sous lequel il le voyait et l'orientation du robot qu'il voyait (grâce à la distance entre les 2 LEDs du bas). La méthode de traitement d'images utilisée était basée sur la soustraction d'images consécutives : après soustractions, les zones restantes correspondent aux différences entre les 2 images, qui devaient être les LEDs qui ont clignotées, en supposant qu'il n'y ait pas d'objet en mouvement. Les éventuelles ambiguités pouvaient être levées par une sélection des zones restantes selon leurs couleurs (et même éventuellement leur forme, les LEDs formant des demi-sphères lumineuses, voir figure 2.28). Il faut noter que le problème d'identification du robot vu était résolu par le fait que les robots clignotent seulement à tour de rôle.



Figure 2.28 – Agencement des LEDs sur les robots, pour pouvoir déterminer la distance et l'angle sous lequel on voit un robot, ainsi que son orientation.

– Dans la même idée, un dispositif de communication ultrason commandé par une impulsion RF afin de mesurer la distance entre des robots avait été développé. Un robot émet une impulsion RF pour signaler qu'il émet simultanément un ping ultrasonore. Comme la vitesse de propagation de l'onde électromagnétique est bien supérieure à celle de l'onde sonore, l'impulsion RF permet de déclencher la mesure du temps de propagation de l'onde ultrasonore, qui peut donner une information sur la distance séparant les 2 robots. Une réflexion du signal ultrasonore sur un ou plusieurs murs peut être détectée en constatant une intensité du signal anormalement faible compte tenu du temps de propagation. Cependant, à cause des réflexions multiples et de l'éventuelle absence de trajet direct du signal acoustique lorsqu'on est dans des pièces différentes dans un bâtiment, il y avait de nombreuses données aberrantes difficiles à détecter.
– Une mesure de distance similaire avait aussi été envisagée en utilisant la qualité de réception du signal Wifi.

Une fois que chaque robot est localisé et a pris des données avec ses télémètres, il est alors simple (en principe...) de reconstituer la carte des zones du bâtiment explorées. Il fallait aussi être capable de détecter et localiser des objets connus dans un environnement inconnu. En effet, pour le concours CAROTTE, des objets dont on connaissait les caractéristiques et/ou qui nous étaient montrés quelques jours avant les épreuves devaient être retrouvés dans le bâtiment inconnu. Des algorithmes de calibration et détection par l'image avaient donc aussi été réalisés.

Le modèle de déplacement d'un de ces robots est celui d'un char. Les équations d'état (simplifiées) d'un robot muni de son télémètre laser peuvent être décrites de cette façon :

$$\begin{cases} \dot{x} &= \cos\theta \\ \dot{y} &= \sin\theta \\ \dot{\theta} &= u \\ \alpha &= \omega \end{cases} \tag{2.8}$$

où u est la commande et ω la vitesse de rotation du télémètre laser. Ce modèle sera réutilisé par la suite en 5.2.

2.4 Conclusion du chapitre

Dans ce chapitre, les principaux robots sous-marins et terrestres utilisés au cours de la thèse et des détails techniques expliquant leur fonctionnement et celui de leurs principaux capteurs ont été présentés. Ces robots seront utilisés dans divers exemples tout au long de ce document (applications de méthodes par intervalles dans le chapitre 3, SLAM dans le chapitre 4, localisation avec données fugaces dans le chapitre 5). Nous pouvons remarquer des similitudes entre les différents robots sous-marins et terrestres étudiés :

– Le fonctionnement et les données que l'on peut récupérer du sonar latéral du *Redermor* et de la *Daurade* sont à mettre en parallèle avec les télémètres infrarouges ou ultrasons latéraux des JOGs.
– De même, le sonar rotatif de *SAUC'ISSE* produit des données similaires à celles des robots CAROTTE avec leur télémètre laser.
– Enfin, ces types de capteurs ont pour point commun de ne fournir des mesures de détection d'amers qu'à des instants bien précis, lorsque le robot est dans une configuration (position, orientation...) bien précise (des détections d'amers ne sont pas toujours disponibles pour chaque instant t).

Ainsi, des résultats obtenus pour des robots terrestres peuvent être parfois assez directement applicables à des robots sous-marins. C'est ce qui sera fait notamment dans les sections 4.4 et 5.3.

Chapitre 3

Calcul par intervalles

Ce chapitre aura pour but de présenter le calcul par intervalles, qui est l'outil central de la thèse. Les principes des algorithmes essentiels associés seront décrits à travers des exemples concrets liés à la robotique.

La première section rappellera les idées qui ont amenées à penser à utiliser des intervalles au lieu des nombres réels et probabilités pour résoudre des problèmes courants, notamment en robotique. Les principaux avantages et exemples de problèmes déjà résolus par des méthodes par intervalles seront cités.

La deuxième section présentera plus en détail l'arithmétique des intervalles, avec des exemples d'utilisation appliqués sur les robots présentés dans le chapitre précédent.

La troisième section montrera que l'arithmétique des intervalles ne se limite pas aux intervalles de réels. Les intervalles de vecteurs, très couramment utilisés en robotique, ainsi que les intervalles de trajectoires seront présentés. Les tubes (intervalles de trajectoires) sont notamment une nouveauté et des algorithmes les utilisant seront notamment présentés pour la première fois dans cette thèse (e.g. dans les sections 3.4.1 et 5.2.2).

Dans une quatrième section, les algorithmes de base (propagation de contraintes et bissection) pour le calcul par intervalles seront rappelés. Les méthodes de résolution des problèmes de SLAM (chapitre 4) et localisation avec données fugaces (chapitre 5) sont fondées sur ceux-ci.

La cinquième section indiquera les outils de programmation existants (et utilisés au cours de la thèse) pour faire du calcul par intervalles.

La dernière section conclura le chapitre et fera le lien avec la suite du document.

3.1 Présentation

Les mesures de capteurs ou variables utilisées pour décrire les robots étant souvent entachées d'erreurs, elles peuvent être représentées de différentes façons : distributions probabilistes, nuages de points, ensembles continus... En général, les données constructeur des capteurs ou actionneurs de robot nous indiquent des bornes (liées à la précision...). On peut donc représenter ces valeurs sous forme d'intervalles.

Les méthodes ensemblistes telles que l'analyse par intervalles permettent d'obtenir des résultats à partir d'équations ou inéquations sur des intervalles. D'autres types d'ensembles peuvent aussi être utilisés : ellipsoïdes (voir *e.g.* [Becis-Aubry et al., 2011]), zonotopes (*e.g.* [Raka and Combastel, 2011])... Avec une approche par intervalles, une variable aléatoire x de $\mathbb{R}$ est représentée par un intervalle $[x]$ qui contient le support de sa densité de probabilité. Même si cette représentation est plus pauvre que celle fournie par sa fonction de densité de probabilité, utiliser le calcul par intervalles peut avoir plusieurs avantages :

- Une arithmétique par intervalles simple peut être définie (voir section 3.2).
- Quand les variables aléatoires sont reliées par des contraintes (*i.e.* équations ou inégalités) un processus de propagation (expliqué plus en détail en 3.4.1) permet d'obtenir des algorithmes polynomiaux efficaces pour calculer des intervalles précis qui sont sûrs de contenir toutes les valeurs possibles pour les variables aléatoires.
- Approximation. Les méthodes de propagation par intervalles ne nécessitent pas de linéariser les équations (elles marchent avec des problèmes non-linéaires, *e.g.* [Meizel et al., 2002] pour résoudre la localisation d'un robot), contrairement au Kalman smoother/lisseur ; elles ne supposent pas que le bruit est additif ; elles n'approximent pas les fonctions de densité de probabilité, contrairement à la plupart des smoothers/lisseurs Bayésiens ; elles ne supposent pas que l'ordinateur calcule avec des nombres réels au lieu de nombres flottants. En revanche, elles considèrent que le bruit est borné, que des bornes sûres sont connues, que le modèle est correct et qu'il n'y a pas de données aberrantes (sauf pour les méthodes les gérant), mais ces suppositions sont déjà contenues dans la définition du problème. La propagation par intervalles ne demande pas d'approximations supplémentaires comme cela peut être le cas avec d'autres méthodes de SLAM non linéaires par exemple.
- Résultats garantis : aucune solution consistante avec les données et hypothèses ou suppositions (sur le modèle et les bornes des erreurs) n'est perdue. Ce n'est pas le cas pour les lisseurs Bayesiens. Les méthodes basées sur un filtre de Kalman ne fournissent en général pas de bornes sur les erreurs de linéarisation et les méthodes particulaires ne fournissent pas de quantification de l'erreur à cause du nombre infini de particules. Seules les méthodes de propagation par intervalles sont capables de fournir une enveloppe contenant toutes les trajectoires possibles du robot dans un contexte non linéaire.
- Efficace pour résoudre des problèmes mettant en jeu un grand nombre d'équations (*e.g.* [Jaulin, 2009a] dans le cas du SLAM sous-marin).
- Robustesse par rapport aux données aberrantes, inconsistances. Lorsque le modèle n'est plus valide, lorsque des données aberrantes apparaissent dans les données (par exemple, si l'erreur sur les données est en fait gaussienne et donc non bornée), les méthodes de propagation par intervalles retournent en général un ensemble vide. Une inconsistance traduit souvent des anomalies pendant la mission du robot et devrait donc être identifiée pour obtenir un système de navigation sûr. Détecter des inconsistances peut donc être utile en pratique pour détecter des bugs de conceptions qui affectent le comportement du robot. Mais il est parfois indispensable de pouvoir les prendre en compte (*e.g.* [Jaulin, 2009b], [Sliwka et al., 2009] pour la localisation et cartographie robuste en robotique).
- Validation. D'un côté, si certains capteurs ne sont pas sûrs ou si le modèle n'est pas suffisamment précis, le système de propagation par intervalles détecte habituellement une inconsistance. D'un autre côté, si la propagation par intervalles fournit une enveloppe consistante et fine autour de la trajectoire du robot et des amers, le robot et ses capteurs peuvent être considérés comme sûrs (ce n'est pas le cas pour le filtre de Kalman, qui fournit souvent une enveloppe fine même si certaines suppositions sur les capteurs sont fausses). Par conséquent, la propagation par intervalles peut être considérée comme une manière sûre de

valider la qualité du système de navigation.
- Algorithmes parallélisables (*e.g.* [Flórez, 2008] pour la gestion d'effets de lumière en 3D sur GPU (Graphical Processor Unit) avec les intervalles, [Piskorski and Lacassagne, 2006] ou [Shettar et al., 2007] pour des implémentations de calculs par intervalles sur FPGA (Field Programmable Gate Array)).
- Les intervalles peuvent être utilisés pour représenter des variables aléatoires avec une fonction de densité de probabilité imprécise.

Les méthodes ensemblistes par intervalles ont aussi déjà prouvé leur intérêt dans divers domaines et applications liés à l'automatique et la robotique :

- Estimation d'état et de paramètres (*e.g.* [Raissi et al., 2004]).
- Contrôle (*e.g.* [Lydoire and Poignet, 2003], [Vinas et al., 2006]).
- Localisation de robots (*e.g.* [Meizel et al., 1996], [Halbwachs and Meizel, 1996], dans le cas linéaire, [Gning and Bonnifait, 2006] pour la localisation dynamique, [Caiti et al., 2002] pour un robot sous-marin).
- SLAM dans le cas de robots roulants terrestres (*e.g.* [Drocourt et al., 2005], [Porta, 2005], [Di Marco et al., 2004], [Di Marco et al., 2001]).
- Détection de boucles dans une trajectoire de robot sous-marin ([Aubry et al., 2011]).
- Analyse topologique d'espaces de configuration ([Delanoue et al., 2006]).

Cependant, les méthodes par intervalles peuvent aussi avoir des inconvénients : le principal est probablement la surestimation (ou conservatisme) des erreurs (*i.e.* le fait d'obtenir parfois des intervalles très larges qui au final donnent une information trop vague pour être réellement utilisable), à cause de l'effet de dépendance (existe pour toute opération ensembliste) et l'effet d'enveloppe. Ceci peut être limité en évitant la multi-occurrence des variables dans les équations [Jaulin et al., 2001b]. Un autre inconvénient peut être que le temps de calcul peut devenir exponentiel par rapport à la dimension des pavés (vecteurs d'intervalles) si des méthodes de bissection sont utilisées.

3.2 Intervalles

Un *intervalle* (voir par exemple [Moore, 1979], [Kearfott and Kreinovich, 1996], [Jaulin et al., 2001b]) est un sous-ensemble connexe fermé de $\mathbb{R}$. Par exemple, $[1,4]$, $\{2\}$, $[-\infty,8]$, $\mathbb{R}$ et $\emptyset$ sont des intervalles alors que $]1,3[$ et $[1,4]\cup[5,8]$ n'en sont pas. On note $\mathbb{IR}$ l'ensemble des intervalles de $\mathbb{R}$.

Si x est une variable réelle, on notera $[x]$ l'intervalle contenant cette variable :

$$[x] = \left[x^-, x^+\right] = \left\{x \in \mathbb{R}, x^- \leq x \leq x^+\right\} \tag{3.1}$$

x^- est la *borne inférieure* de l'intervalle et x^+ sa *borne supérieure*.

Le *centre* d'un intervalle non vide et borné est :

$$\text{mid}\left([x]\right) = \frac{x^- + x^+}{2}. \tag{3.2}$$

En robotique, la position x d'un robot sera souvent approximée (pour le dessin de la trajectoire par exemple) par le centre de l'intervalle $[x]$ calculé à partir d'algorithmes intervalles.

La *longueur* d'un intervalle non vide est définie par $w([x]) = x^+ - x^-$. Par convention, $w(\emptyset) = -\infty$. Si $w([x]) = 0$, $[x]$ est *dégénéré*. Dans ce cas, $[x]$ est un singleton de réels et sera noté $\{x\}$. Bien souvent, on fait correspondre l'erreur de mesure faite par un capteur à la demi-longueur de l'intervalle représentant la variable mesurée. Par exemple, si une boussole vendue comme ayant une précision de $2\,^\circ$ indique $45\,^\circ$, on représente cette donnée par l'intervalle $[\theta] = [45-2, 45+2] = [43, 47]$.

Considérons 2 intervalles $[x]$ et $[y]$ et un opérateur $\diamond \in \{+, -, *, /\}$, on définit $[x] \diamond [y]$ comme le plus petit intervalle contenant toutes les valeurs possibles pour $x \diamond y$, si $x \in [x]$ et $y \in [y]$ (voir [Jaulin et al., 2001b]). Ainsi :

$$\begin{array}{rcl} [x] + [y] & = & [x^- + y^-, x^+ + y^+] \\ [x] - [y] & = & [x^- - y^+, x^+ - y^-] \\ [x] * [y] & = & [\min\{x^-y^-, x^-y^+, x^+y^-, x^+y^+\}, \max\{x^-y^-, x^-y^+, x^+y^-, x^+y^+\}] \\ [x] / [y] & = & [x] * 1/[y] \end{array} \tag{3.3}$$

avec

$$\begin{array}{rcll} 1/[y] & = & \emptyset & \text{si } [y] = [0,0] \\ & = & [1/y^+, 1/y^-] & \text{si } 0 \notin [y] \\ & = & [1/y^+, \infty] & \text{si } y^- = 0 \text{ et } y^+ > 0 \\ & = & [-\infty, 1/y^+] & \text{si } y^- < 0 \text{ et } y^+ = 0 \\ & = & [-\infty, \infty] & \text{si } y^- < 0 \text{ et } y^+ > 0 \end{array} \tag{3.4}$$

De plus, si $\alpha \in \mathbb{R}$, on a :

$$\begin{array}{rcll} \alpha[x] & = & [\alpha x^-, \alpha x^+] & \text{si } \alpha \geq 0 \\ & = & [\alpha x^+, \alpha x^-] & \text{si } \alpha < 0. \end{array} \tag{3.5}$$

L'intersection, l'union et la restriction d'intervalles sont définis comme suit :

$$\begin{array}{rcl} [x] \cap [y] & = & [\max\{x^-, y^-\}, \min\{x^+, y^+\}] \text{ si } \max\{x^-, y^-\} \leq \min\{x^+, y^+\} \\ & = & \emptyset \text{ autrement} \\ [x] \sqcup [y] & = & [\min\{x^-, y^-\}, \max\{x^+, y^+\}] \\ [x] \,[\backslash]\, [y] & = & \emptyset \text{ si } [x] \subseteq [y] \\ & = & [x^-, y^-] \text{ si } x^- < y^- \text{ et } x^+ \leq y^+ \text{ et } x^+ > y^- \\ & = & [y^+, x^+] \text{ si } x^- \geq y^- \text{ et } x^+ > y^+ \text{ et } x^- < y^+ \\ & = & [x] \text{ autrement.} \end{array} \tag{3.6}$$

Il est à noter que $[x] \cup [y]$ (à différencier de $[x] \sqcup [y]$) ne sera pas considéré comme un intervalle dans le cadre de cette thèse, même si des travaux existent sur les unions d'intervalles (intervalles discontinus, [Hyvönen, 1992]).

Voici quelques exemples de calculs d'intervalles :

$$\begin{array}{rcl} [-1,4] + [2,3] & = & [1,7] \\ [-1,4] - [2,3] & = & [-4,2] \\ [-1,4] * [2,3] & = & [-3,12] \\ [-1,4]/[2,3] & = & [-\frac{1}{2}, 2] \\ 2[-1,4] & = & [-2,8] \\ [-1,3] \cap [2,4] & = & [2,3] \\ [-1,2] \sqcup [3,4] & = & [-1,4] \\ [-1,4] \,[\backslash]\, [2,3] & = & [-1,4]. \end{array} \tag{3.7}$$

Si f est une fonction élémentaire telle que sin,$\sqrt{}$,exp,... on définit $f([x])$ comme le plus petit intervalle contenant toutes les valeurs possibles pour $f(x)$, si $x \in [x]$. Par exemple, si $x \neq \emptyset$,

$$\exp([x]) = \left[\exp\left(x^-\right), \exp\left(x^+\right)\right]. \tag{3.8}$$

Pour des fonctions non monotones telles que sin, un algorithme plus compliqué doit être construit (voir par exemple [Jaulin et al., 2001b], [Revol, 2001]). Il est aussi important de noter qu'en général, $[x]^2 \neq [x] * [x]$ par exemple, à cause de l'effet de dépendance dû à la multi-occurrence de $[x]$.

La fonction $[f] : \mathbb{IR} \to \mathbb{IR}$ est une *fonction d'inclusion* de la fonction réelle $f : \mathbb{R} \to \mathbb{R}$ si et seulement si pour tout $[x] \in \mathbb{IR}$,

$$f([x]) \subset [f]([x]), \tag{3.9}$$

$[x] \to \mathbb{R}$ est une fonction d'inclusion pour toutes les fonctions f de $\mathbb{R}$ dans $\mathbb{R}$ qui est souvent utilisée à l'initialisation d'algorithmes cherchant à retrouver des trajectoires de robots par exemple.

Une fonction d'inclusion $[f]$ est *minimale* si pour tout $[x]$, $[f]([x])$ est le plus petit intervalle contenant $f([x])$. Dans ce cas, elle est unique et sera notée $[f]^*$. En pratique, le type de fonction d'inclusion qu'on utilise est la *fonction d'inclusion naturelle*. La fonction d'inclusion naturelle $[f]$ d'une fonction

$$\begin{array}{rcl} f : \mathbb{R}^n & \to & \mathbb{R} \\ (x_1, ..., x_n) & \longmapsto & f(x_1, ..., x_n), \end{array} \tag{3.10}$$

$n \in \mathbb{N}$ exprimée comme une composition finie des opérateurs $+, -, *, /$ et de fonctions élémentaires est obtenue en remplaçant chaque occurrence des variables réelles par l'intervalle correspondant et en utilisant la version intervalle des opérateurs et fonctions. Si les opérateurs et fonctions sont continus et que chaque variable n'apparaît au plus qu'une fois dans l'expression formelle, alors $[f]$ est minimale (voir [Jaulin et al., 2001b]). Il existe d'autres types de fonctions d'inclusions telles que les formes centrées ou de Taylor.

Typiquement, les équations discrétisées modélisant l'évolution, l'observation ou les mesures des robots peuvent être écrites sous la forme d'une composition finie des opérateurs $+, -, *, /$ et de fonctions élémentaires continues : on obtient donc immédiatement une fonction d'inclusion minimale. Par exemple, les équations du char (équations 2.7 du robot JOG) peuvent être discrétisées avec la méthode d'Euler sous la forme :

$$\left\{ \begin{array}{rcl} x_{k+1} & = & f_{x,k}(x_k, y_k, \theta_k, u_{1,k}, u_{2,k}) = x_k + dt * u_{1,k} * \cos\theta_k \\ y_{k+1} & = & f_{y,k}(x_k, y_k, \theta_k, u_{1,k}, u_{2,k}) = y_k + dt * u_{1,k} * \sin\theta_k \\ \theta_{k+1} & = & f_{\theta,k}(x_k, y_k, \theta_k, u_{1,k}, u_{2,k}) = \theta_k + dt * u_{2,k} \end{array} \right. \tag{3.11}$$

avec dt le pas de temps (il faut noter qu'un intervalle représentant le bruit d'état est souvent aussi à rajouter pour prendre en compte les erreurs de modèle et de discrétisation). Si par exemple on a $[x_0], [y_0], [\theta_0]$ (état initial du robot) et $[u_{1,k}], [u_{2,k}]$ (mesures des vitesses prises comme entrées) $\forall k \in \mathbb{N}$, on peut évaluer (par récurrence) :

$$\left\{ \begin{array}{rcl} [x_{k+1}] & = & [f_{x,k}]([x_k], [y_k], [\theta_k], [u_{1,k}], [u_{2,k}]) = [x_k] + dt * [u_{1,k}] * [\cos][\theta_k] \\ [y_{k+1}] & = & [f_{y,k}]([x_k], [y_k], [\theta_k], [u_{1,k}], [u_{2,k}]) = [y_k] + dt * [u_{1,k}] * [\sin][\theta_k] \\ [\theta_{k+1}] & = & [f_{\theta,k}]([x_k], [y_k], [\theta_k], [u_{1,k}], [u_{2,k}]) = [\theta_k] + dt * [u_{2,k}]. \end{array} \right. \tag{3.12}$$

En prenant le centre des intervalles $[x_{k+1}]$ et $[y_{k+1}]$ pour tout k, on pourra alors dessiner une approximation de la trajectoire du robot.

3.3 Généralisation des intervalles

3.3.1 Principe

La notion d'intervalles peut être généralisée à tout ensemble admettant une structure de treillis. Ainsi, on peut considérer des intervalles de réels, vecteurs, booléens, sous-pavages, fonctions... (voir [Chabert and Jaulin, 2009] [Delanoue, 2006], [Jaulin et al., 2001b] pour plus d'informations sur ces généralisations et les treillis). Dans la section précédente, nous étions par exemple dans le treillis des nombres réels.

3.3.2 Intervalles vectoriels

Dans cette sous-section, nous allons nous placer dans le treillis des vecteurs de $\mathbb{R}^n$.

Un *pavé*, ou *intervalle vectoriel* (ou encore boîte) $[\mathbf{x}]$ de $\mathbb{R}^n$ est un vecteur dont les composantes sont des intervalles :

$$[\mathbf{x}] = [x_1^-, x_1^+] \times ... \times [x_n^-, x_n^+] = [x_1] \times ... \times [x_n] = ([x_1], ..., [x_n])^T . \tag{3.13}$$

C'est un produit cartésien de n intervalles réels, $n \in \mathbb{N}$. $[x_i]$, $i \in \{1, ..., n\}$ est la $i^{ème}$ composante intervalle de $[\mathbf{x}]$ ou encore la projection de $[\mathbf{x}]$ sur le $i^{ème}$ axe. $\mathbb{R}^n$, $\emptyset \times ... \times \emptyset$ $[0, 1] \times [-1, 2] \times [-\infty, 4]$ sont des exemples de pavés mais $[0, 1] \times \emptyset$ n'a pas de sens par exemple. Si $n = 2$, un pavé peut être dessiné en 2D sous forme de rectangle parallèle aux axes du repère utilisé (voir figure 3.1).

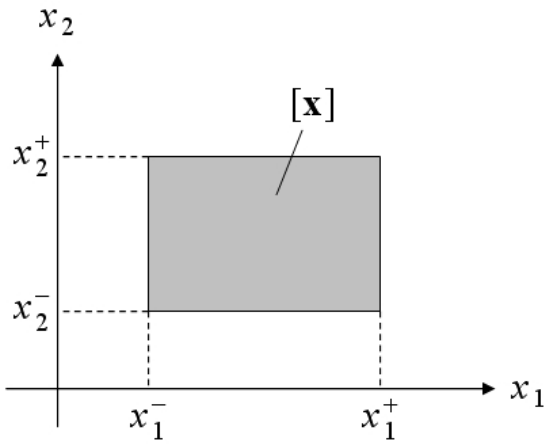


Figure 3.1 – Un pavé de $\mathbb{R}^2$.

$\mathbb{IR}^n$ est l'ensemble des pavés de $\mathbb{R}^n$. La *borne inférieure* et *borne supérieure* d'un pavé sont définis respectivement par :

$$\begin{array}{rcl} \mathbf{x}^- & = & \left(x_1^-, ..., x_n^-\right)^T \\ \mathbf{x}^+ & = & \left(x_1^+, ..., x_n^+\right)^T . \end{array} \tag{3.14}$$

Sa *longueur* $w([\mathbf{x}]) = \max_{1\leq i\leq n} w([x_i])$ est la longueur de son côté le plus long. Son *centre* est $\text{mid}([\mathbf{x}]) = (\text{mid}([x_1]), ..., \text{mid}([x_n]))^T$. Les opérations et fonctions élémentaires vues pour les intervalles ainsi que la notion de fonction d'inclusion se généralisent aussi aux pavés en appliquant les calculs intervalles à chaque composante [Jaulin et al., 2001b].

3.3.3 Tubes

Dans cette sous-section, nous allons nous placer dans le treillis des fonctions de $\mathbb{R} \rightarrow \mathbb{R}^n$.

Un *tube* $[\mathbf{x}](t)$, avec un pas de temps $\delta > 0$ est une fonction d'intervalles vectoriels constante dans $[k\delta, k\delta + \delta], k \in \mathbb{Z}$. Le pavé $[k\delta, k\delta + \delta] \times [\mathbf{x}](t_k)$, avec $t_k \in [k\delta, k\delta + \delta]$ est appelé la $k^{ième}$ *tranche* du tube $[\mathbf{x}](t)$ et sera noté $[\mathbf{x}](k)$ (voir figure 3.2). On peut donc voir un tube comme une union de tranches. Par abus de langage, on utilisera parfois $[\mathbf{x}](k)$ pour représenter $[\mathbf{x}](t_k)$. On définit la *fonction de correspondance d'index* κ comme suit :

$$\kappa([t_a, t_b]) = \{k \in \mathbb{Z}, \exists t \in [t_a, t_b], t \in [k\delta, k\delta + \delta]\}. \tag{3.15}$$

Un tube $[\mathbf{x}](t)$ contenant une fonction vectorielle (de $\mathbb{R} \rightarrow \mathbb{R}^n$) $t \longmapsto \mathbf{x}(t)$ a une borne inférieure $t \longmapsto \mathbf{x}^-(t)$ et

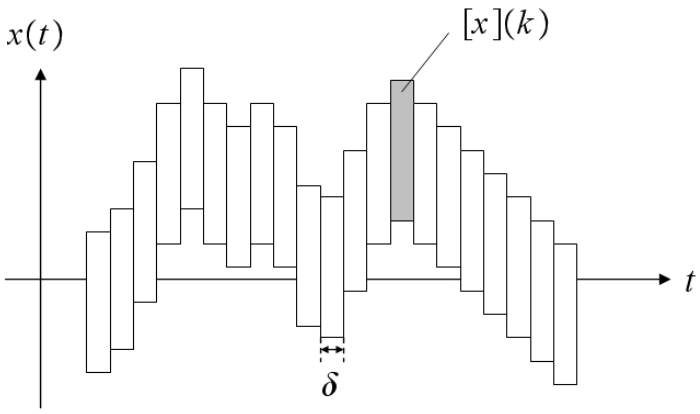


Figure 3.2 – Un tube $[x](t)$, avec un pas de temps δ. $[x](k)$ est sa $k^{ième}$ tranche.

une borne supérieure $t \longmapsto \mathbf{x}^+(t)$ qui sont des fonctions vectorielles en escalier constantes dans $[k\delta, k\delta + \delta], k \in \mathbb{Z}$ telles que $\forall t \in \mathbb{R}, \mathbf{x}(t) \in [\mathbf{x}^-(t), \mathbf{x}^+(t)] = [\mathbf{x}](t)$. Un tube est en fait un intervalle de fonctions de $\mathbb{R} \rightarrow \mathbb{R}^n$ (de la même façon qu'un intervalle contient un ensemble de réels et qu'un pavé contient un ensemble de vecteurs, un tube contient un ensemble de fonctions).

En robotique, on peut voir un tube comme un intervalle de trajectoires contenant la trajectoire réelle du robot par exemple (voir figure 3.3). Une trajectoire $\mathbf{x}(t)$ appartient au tube $[\mathbf{x}](t)$ si $\forall t, \mathbf{x}(t) \in [\mathbf{x}](t)$. La connaissance d'un tube $[\mathbf{x}](t)$ contenant une fonction inconnue $\mathbf{x}(t)$ peut nous donner des informations sur $\mathbf{x}(t)$. Par exemple, on voit sur la figure 3.4 que la fonction $x(t)$ dont on ne connaît que le tube $[x](t)$ s'annule au moins 4 fois (en regardant les signes des tranches grises) et que ces annulations ne peuvent se produire que dans les intervalles $[t_1], [t_2], [t_3], [t_4]$.

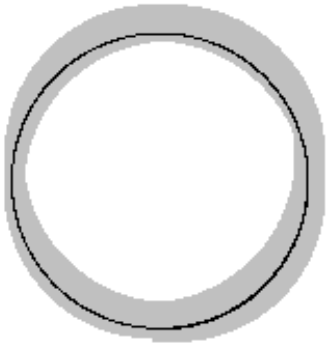


Figure 3.3 – Tube (formé ici par la superposition de pavés) contenant la trajectoire d'un robot char (dans le plan $(\mathbf{O}, \mathbf{i}, \mathbf{j})$). La trajectoire noire $\mathbf{x}(t)$ est la trajectoire réelle, générée par une simulation et l'enveloppe grise $[\mathbf{x}](t)$ est le tube obtenu après traitement des données et équations. On voit que la trajectoire réelle ne sort jamais du tube, comme les méthodes intervalles nous le garantissent.

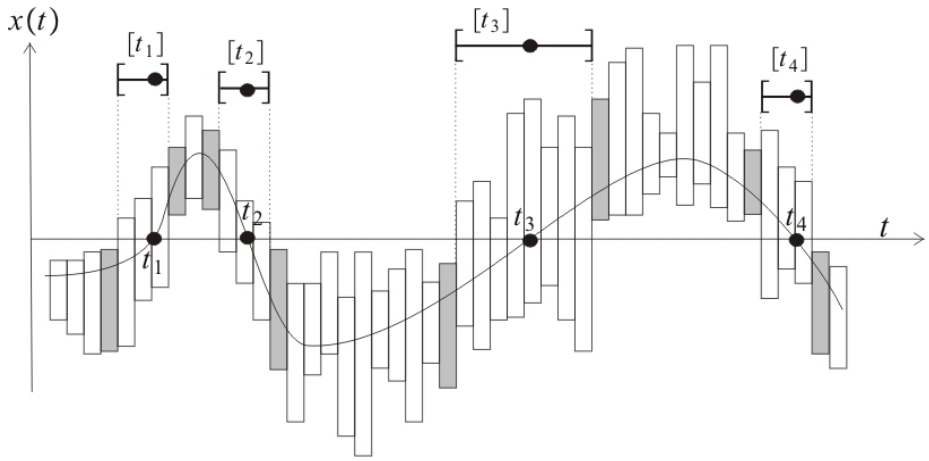


Figure 3.4 – Un tube permet d'obtenir des informations intéressantes sur la trajectoire qu'il représente. On peut par exemple voir ici que la fonction $x(t)$ s'annule au moins 4 fois.

Une notion de tube existe aussi dans des contextes d'erreur bornée plus généraux (voir [Kurzhanski and Valyi, 1997], [Milanese et al., 1996]).

Les opérations et fonctions élémentaires vues pour les intervalles ainsi que la notion de fonction d'inclusion se généralisent aussi aux tubes en appliquant les calculs intervalles pour tout t [Moore, 1979]. L'intégrale d'un tube peut être définie comme ceci :

$$\int_{t_0}^{t} [\mathbf{x}](\tau)\, d\tau = \left[\int_{t_0}^{t} \mathbf{x}^{-}(\tau)\, d\tau, \int_{t_0}^{t} \mathbf{x}^{+}(\tau)\, d\tau \right]. \tag{3.16}$$

On peut montrer (par le théorème de croissance pour des intégrales de fonctions vectorielles) que

$$\mathbf{x}(t) \in [\mathbf{x}](t) \Rightarrow \int_{t_0}^{t} \mathbf{x}(\tau)\, d\tau \in \int_{t_0}^{t} [\mathbf{x}](\tau)\, d\tau \tag{3.17}$$

et que $\int_{t_0}^{t} [\mathbf{x}](\tau)\, d\tau$ est un tube. La dérivée d'un tube n'est pas calculable dans le cas général. Cependant, en robotique, les équations d'état des robots nous donnent souvent une expression analytique des dérivées. Par exemple, les équations d'états du char (équations 2.7) permettent d'obtenir des tubes $[\dot{x}](t), [\dot{y}](t), \left[\dot{\theta}\right](t)$ contenant $\dot{x}(t), \dot{y}(t), \dot{\theta}(t)$ par les opérations suivantes :

$$\begin{array}{rcl} [\dot{x}](t) & = & [u_1](t)\,[\cos]\,[\theta](t) \\ [\dot{y}](t) & = & [u_1](t)\,[\sin]\,[\theta](t) \\ \left[\dot{\theta}\right](t) & = & [u_2](t) \end{array} \tag{3.18}$$

avec $[\cos], [\sin]$ les équivalents intervalles de cos et sin.

3.4 Algorithmes intervalles

Deux principaux types d'algorithmes sont fréquemment utilisés pour résoudre des problèmes mettant en jeu des intervalles : la propagation de contraintes et les bissections. Cette section présentera leur principe, avec quelques exemples d'applications liées à la robotique. Les algorithmes de SLAM et localisation avec données fugaces qui seront présentés dans les chapitres suivants sont basés sur ces techniques (surtout sur la propagation de contraintes, les bissections n'étant utilisées que dans la section 4.4).

3.4.1 Contraction et propagation

La propagation de contraintes intervalles (ICP pour Interval Constraint Propagation) est une technique (ou sorte d'algorithme) permettant de résoudre rapidement (le temps de calcul est en général assez faible comparé à d'autres méthodes) des équations mettant en jeu des intervalles. C'est la combinaison entre 2 notions plus générales : la propagation de contraintes et le calcul par intervalles [Jaulin and Walter, 2002].

De nombreux problèmes peuvent être écrits sous la forme d'un CSP (Constraint Satisfaction Problem, comme défini dans [Jaulin, 2002b], [Sam-Haroud, 1995] ou encore [Walter and Pronzato, 1997]). Un CSP est constitué

1. d'un ensemble de variables $\mathcal{V} = \{x_1, \ldots, x_n\}$

2. d'un ensemble de domaines $\{[x_1], \ldots, [x_n]\}$ qui sont censés contenir les variables $x_1, ..., x_n$

3. et un ensemble de contraintes reliant entre elles ces variables

Un CSP peut avoir pour variables x_i des booléens (CSP booléens), entiers (CSP discrets), réels, vecteurs, fonctions de $\mathbb{R}$ dans $\mathbb{R}^n$ (CSP continus), des contraintes (la notion de contrainte est assez générale) qui peuvent être des équations ou inéquations entre des variables réelles ($x_3 = x_1 + \exp(x_2)$ par exemple), sur des trajectoires (équations différentielles, équations de retards/délais...), et les domaines peuvent être des intervalles (*e.g.* [van Emden, 1999]), pavés (*e.g.* [Jaulin et al., 2001b]), zonotopes (*e.g.* [Combastel, 2005]), ellipsoïdes (*e.g.* [Durieu et al., 1996]) ou des tubes ([Le Bars et al., 2011a]). On dit qu'un domaine est consistant localement (consistance locale) s'il est consistant avec toutes les contraintes prises indépendamment. Le plus petit domaine contenant toutes les solutions est dit globalement consistant (comme expliqué dans [Jaulin, 2009a]). Trouver ce domaine est un problème NP-complet (on peut utiliser des méthodes de bissections, voir sous-section 3.4.2).

Par exemple, considérons le CSP mettant en jeu 3 variables réelles x, y, z liées par l'équation $z = x + y$, et supposons qu'initialement $x \in [x], y \in [y], z \in [z]$ avec $[x] = [-\infty, 2], [y] = [-\infty, 4]$ et $[z] = [1, \infty]$. Alors on a :

$$[z] = [x] + [y] = [-\infty, 2] + [-\infty, 4] = [-\infty, 6]. \tag{3.19}$$

Or, on sait au départ que $z \in [1, \infty]$, donc :

$$z \in [1, \infty] \cap [-\infty, 6] = [1, 6]. \tag{3.20}$$

On s'aperçoit qu'on a considérablement amélioré notre connaissance de z par cette procédure. Si ensuite on écrit

$$\begin{array}{rll} x = z - y \Rightarrow & x \in & [-\infty, 2] \cap ([1, 6] - [-\infty, 4]) \\ & & = [-\infty, 2] \cap [-3, \infty] = [-3, 2] \\ y = z - x \Rightarrow & y \in & [-\infty, 4] \cap ([1, 6] - [-3, 2]) \\ & & = [-\infty, 4] \cap [-1, 9] = [-1, 4], \end{array} \tag{3.21}$$

on peut donc conclure que $[x] = [-3, 2], [y] = [-1, 4]$ et $[z] = [1, 6]$.

On peut aussi faire face en robotique à des CSP mettant en jeu des tubes. Par exemple, si on cherche à calculer la position x d'un robot dont on sait qu'il s'éloigne toujours de l'origine de son repère de référence pendant une durée connue, on a une contrainte de croissance sur le tube englobant la fonction décrivant sa position x en fonction du temps (voir figure 3.5).

Le but des techniques de propagation est de contracter (réduire) au maximum la taille des domaines contenant les variables recherchées, sans perdre de solutions. Ceci peut se faire en utilisant la notion de *contracteur* (définie dans [Jaulin and Chabert, 2008] et [Chabert and Jaulin, 2009] notamment). L'opérateur $\mathcal{C} : \mathbb{IR}^n \rightarrow \mathbb{IR}^n$ est un contracteur si (on peut aussi généraliser la notion de contracteur pour d'autres types d'intervalles, par exemple l'ensemble des fonctions de $\mathbb{R} \rightarrow \mathbb{R}^n$ pour les tubes)

$$\begin{array}{lll} (i) & \forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}([\mathbf{x}]) \subseteq [\mathbf{x}] & \text{(contractance)} \\ (ii) & (\mathbf{x} \in [\mathbf{x}], \mathcal{C}(\{\mathbf{x}\}) = \{\mathbf{x}\}) \Longrightarrow \mathbf{x} \in \mathcal{C}([\mathbf{x}]) & \text{(consistance)} \\ (iii) & \mathcal{C}(\{\mathbf{x}\}) = \emptyset \Leftrightarrow (\exists \varepsilon > 0, \forall [\mathbf{x}] \subseteq B(\mathbf{x}, \varepsilon), \mathcal{C}([\mathbf{x}]) = \emptyset) & \text{(convergence)} \end{array} \tag{3.22}$$

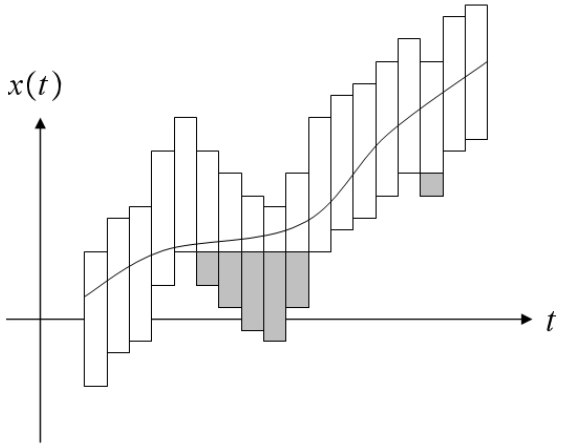


Figure 3.5 – Une contrainte de croissance sur $x\,(t)$ permet d'éliminer les parties grises du tube initial.

où $B\,(\mathbf{x},\varepsilon)$ désigne la boule de centre $\mathbf{x}$ et de rayon ε. La propriété (i) nous assure que par l'action d'un contracteur, un pavé ne peut que se contracter. La propriété (ii) nous dit que tout pavé gardera, après contraction tous ses points $\mathbf{x}$ insensibles à ce même contracteur. Enfin, (iii) nous assure entre autre que l'ensemble des $\mathbf{x}$ sensibles à $\mathcal{C}$ est un ensemble ouvert, et donc que l'ensemble des $\mathbf{x}$ insensibles est fermé (cette dernière propriété est rarement utilisée). Un pavé est dit insensible au contracteur $\mathcal{C}$ si

$$\mathcal{C}\,([\mathbf{x}]) = [\mathbf{x}]\,. \tag{3.23}$$

L'ensemble associé à un contracteur $\mathcal{C}$ (noté set$(\mathcal{C})$) est l'ensemble formé par l'union des singletons insensibles à $\mathcal{C}$, c'est-à-dire

$$\text{set}\,(\mathcal{C}) = \{\mathbf{x} \in \mathbb{R}^n, \mathcal{C}\,(\{\mathbf{x}\}) = \{\mathbf{x}\}\}\,. \tag{3.24}$$

On peut définir la contraction comme l'action d'un contracteur, le fait de réduire la taille du domaine contenant une variable. Un contracteur peut être vu comme un moyen de représenter un sous-ensemble de $\mathbb{R}^n$. Il correspond à un algorithme (implémentable sur ordinateur par exemple) et contient toute l'information sur l'ensemble qu'il représente. Les manipulations sur les ensembles (intersections, union...), pourront se faire à travers leur contracteur. Un contracteur est un algorithme en général construit à partir d'une contrainte pour contracter les domaines des variables impliquées dans cette contrainte (cependant, un contracteur n'est pas forcément associé à une contrainte [Chabert and Jaulin, 2009] mais nous ne nous intéresserons pas à cette nuance).

Opérations sur les contracteurs :

$$\begin{array}{ll}
\text{intersection} & (\mathcal{C}_1 \cap \mathcal{C}_2)\,([\mathbf{x}]) = \mathcal{C}_1\,([\mathbf{x}]) \cap \mathcal{C}_2\,([\mathbf{x}]) \\
\text{union} & (\mathcal{C}_1 \sqcup \mathcal{C}_2)\,([\mathbf{x}]) = \mathcal{C}_1\,([\mathbf{x}]) \sqcup \mathcal{C}_2\,([\mathbf{x}]) \\
\text{composition} & (\mathcal{C}_1 \circ \mathcal{C}_2)\,([\mathbf{x}]) = \mathcal{C}_1\,(\mathcal{C}_2\,([\mathbf{x}])) \\
\text{répétition} & \mathcal{C}_1^{\infty} = \mathcal{C}_1 \circ \mathcal{C}_1 \circ \mathcal{C}_1 \circ \ldots \\
\text{intersection répétée} & \mathcal{C}_1 \sqcap \mathcal{C}_2 = (\mathcal{C}_1 \cap \mathcal{C}_2)^{\infty}
\end{array} \tag{3.25}$$

Inclusion entre contracteurs :

$$\mathcal{C}_1 \subset \mathcal{C}_2 \Leftrightarrow \forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}_1([\mathbf{x}]) \subset \mathcal{C}_2([\mathbf{x}]) \tag{3.26}$$

On peut définir la propagation comme un ensemble de compositions et/ou répétitions de contracteurs. De plus, on dit que :

$$\begin{array}{ll} \mathcal{C} \text{ est } \textit{monotone} \text{ si} & [\mathbf{x}] \subset [\mathbf{y}] \Rightarrow \mathcal{C}([\mathbf{x}]) \subset \mathcal{C}([\mathbf{y}]) \\ \mathcal{C} \text{ est } \textit{minimal} \text{ si} & \forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}([\mathbf{x}]) = [[\mathbf{x}] \cap \text{set}(\mathcal{C})] \\ \mathcal{C} \text{ est } \textit{idempotent} \text{ si} & \forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}(\mathcal{C}([\mathbf{x}])) = \mathcal{C}([\mathbf{x}]). \end{array} \tag{3.27}$$

On peut montrer qu'on a toujours $[\mathbf{x}] \cap \text{set}(\mathcal{C}) \subset \mathcal{C}([\mathbf{x}])$ et que tous les contracteurs minimaux sont idempotents. Dans les applications, on dit parfois qu'un contracteur est *optimal* s'il est minimal.

Il existe de nombreuses méthodes pour mettre au point des contracteurs adaptés à des contraintes complexes. Certaines utilisent le calcul formel avec des intervalles (par exemple en calculant les dérivées des fonctions impliquées dans les équations formant les contraintes), d'autres peuvent être spécifiques au problème (pour des contraintes très utilisées en robotique comme les contraintes de distance et d'angle). Lorsque le problème comporte plus d'une contrainte (ou quand la contrainte peut donner des résultats différents selon son écriture), les contractions doivent être effectuées les unes après les autres plusieurs fois, jusqu'à obtention d'un point fixe (les contractions n'ont plus d'effets significatifs). Malheureusement, il n'y a pas de critère connu pour prévoir la précision et la rapidité d'obtention de ce point fixe (il est cependant possible de faire une analyse d'erreur a priori comme décrit dans [Chabert and Jaulin, 2011], où une propagation de contraintes est effectuée sur des intervalles contenant les erreurs prévues pour toutes les mesures, sans avoir les mesures elles-mêmes, par contre le temps de calcul est le même que si on faisait le calcul final avec les vraies mesures, ce qui limite l'intérêt de cette méthode aux cas où on n'a pas de données complètes à traiter). L'optimalité des contracteurs utilisés peut éventuellement donner une information relative sur le temps de calcul et la précision (mais la composition de contracteurs minimaux n'est pas forcément minimale), mais cela n'exclut pas qu'il existe d'autres contracteurs plus rapides et plus précis pour le problème traité. La méthode de propagation converge vers un domaine (pas forcément le plus petit possible, notamment à cause des effets d'enveloppe et de dépendance locale) qui contient l'ensemble de toutes les solutions de l'ensemble de contraintes de notre CSP. Si ce domaine est vide, on a démontré que le problème n'avait aucune solution. Par contre, on montre que le point fixe (qu'on pourrait appeler aussi domaine fixe) ne dépend pas de l'ordre dans lequel les contracteurs sont appelés [Jaulin et al., 2001b], mais le temps de calcul y est très sensible. Il n'y a pas d'ordre optimal en général mais en pratique, l'un des plus efficace est la propagation *forward-backward* [Jaulin, 2009a] (aussi connu sous le nom de *hull consitency* [Jaulin et al., 2001b], *2B-coherence* [Trombettoni and Chabert, 2007a] ou *HC4Revise* [Benhamou et al., 1999]). Cette méthode nécessite d'écrire l'ensemble des contraintes du CSP sous la forme $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ où $\mathbf{x}$ et $\mathbf{y}$ correspondent à des quantités mesurées (*i.e.* on a des domaines initiaux pour ces variables). Ensuite, en utilisant l'arithmétique des intervalles, les intervalles sont propagés de $\mathbf{x}$ vers $\mathbf{y}$ dans un premier temps (*forward propagation*, voir les calculs 3.19 et 3.20) puis de $\mathbf{y}$ vers $\mathbf{x}$ dans un second temps (*backward propagation*, voir les calculs 3.21). Comme pour toute propagation, ce processus est à répéter jusqu'à ce que les contractions n'aient plus d'effets significatifs.

Il y a aussi des techniques particulières pour contracter des domaines $[\mathbf{x}](t)$ par rapport à des équations diffé-

rentielles telles que $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ [Jaulin, 2002a][Jaulin, 2009a][Berz et al., 1996] [Berz and Makino, 1998][Raissi et al., 2004] (elles sont utiles pour le problème du SLAM) ou des contraintes linéaires (*e.g.* Gauss-Seidel, voir [Jaulin et al., 2001b]). Par exemple, les calculs 3.12 correspondaient déjà presque à une propagation *forward* par rapport au temps (de la condition initiale jusqu'au temps final) pour une équation différentielle : connaissant $[x_0], [y_0], [\theta_0]$ (état initial du robot) et $[u_{1,k}], [u_{2,k}]$ (mesures des vitesses prises comme entrées), $\forall k \in [0, n]$ (n correspondant au temps final), on peut évaluer $[x_k], [y_k], [\theta_k]$ en faisant

$$\begin{cases} [x_{k+1}] &= [x_{k+1}] \cap ([x_k] + dt * [u_{1,k}] * [\cos]([\theta_k])) \\ [y_{k+1}] &= [y_{k+1}] \cap ([y_k] + dt * [u_{1,k}] * [\sin]([\theta_k])) \\ [\theta_{k+1}] &= [\theta_{k+1}] \cap ([\theta_k] + dt * [u_{2,k}]) . \end{cases} \tag{3.28}$$

Si de plus on connait par exemple $[x_n], [y_n], [\theta_n]$ (état final), une propagation *backward* (de l'instant final à l'instant initial) améliorera l'évaluation de la trajectoire du char :

$$\begin{cases} [\theta_k] &= [\theta_k] \cap ([\theta_{k+1}] - dt * [u_{2,k}]) \\ [y_k] &= [y_k] \cap ([y_{k+1}] - dt * [u_{1,k}] * [\sin]([\theta_k])) \\ [\theta_k] &= [\theta_k] \cap \left(\left[\sin^{-1}\right] \left(\frac{[y_{k+1}]-[y_k]}{dt*[u_{1,k}]} \right) \right) \\ [x_k] &= [x_k] \cap ([x_{k+1}] - dt * [u_{1,k}] * [\cos]([\theta_k])) \\ [\theta_k] &= [\theta_k] \cap \left(\left[\cos^{-1}\right] \left(\frac{[x_{k+1}]-[x_k]}{dt*[u_{1,k}]} \right) \right) . \end{cases} \tag{3.29}$$

Il faut noter qu'en général $\left[\cos^{-1}\right] \neq [\arccos]$ et $\left[\sin^{-1}\right] \neq [\arcsin]$ à cause des problèmes de modulo 2π.

3.4.2 Algorithmes de bissection

Le principe d'un algorithme de *bissection* [Chabert, 2007b] peut être décrit comme ceci : il consiste à effectuer un arbre de recherche où chaque nœud est un intervalle représentant le domaine courant des variables. Un test est effectué sur cet intervalle via l'arithmétique des intervalles. Si ce test est faux, alors le test sera faux si on le fait sur tous les points de l'intervalle, donc il peut être supprimé. Inversement, s'il est vrai, alors le test sera vrai si on le fait sur tous les points de l'intervalle, donc on peut le ranger dans la liste des solutions. Sinon (cas où il y a des points de l'intervalle où le test est vrai et d'autres où le test est faux), il est coupé en deux sous-intervalles sur lesquels la recherche est poursuivie récursivement. On parle de bissection (*branch*) pour le découpage en sous-intervalles, et d'évaluation (*bound*) pour le test. Enfin, un intervalle dont la longueur devient trop petite et que le test n'a pas permis d'exclure, est retiré du processus de recherche et en général stocké dans une liste à part. Typiquement, ce type d'algorithme donne lieu à des dessins en 2D de sous-pavages de solutions, de sous-pavages non solutions et des sous-pavages indéterminés.

SIVIA est un exemple d'algorithme de bissection (voir [Jaulin and Walter, 1993] pour l'algorithme de base, [Le Bars et al., 2009] par exemple pour une variante faisant un parallèle entre le problème du lancé de rayon et l'analyse de stabilité d'un système paramétrique par le calcul de la plus petite racine d'une fonction).

Plusieurs autres variantes d'algorithmes de bissections plus ou moins optimisés et utilisés habituellement en combinaison avec des contracteurs existent : *e.g.* 3BCID (qui peut être appliqué à n'importe quel contracteur) [Trombettoni and Chabert, 2007b], STRANGLE (que l'on peut appliquer sur les positions du robot et des amers dans un contexte de SLAM par exemple) [Jaulin, 2002a].

3.4.3 Combinaison de contraction, propagation et bissection

Les procédures de contraction et propagation sont habituellement appelées plusieurs fois jusqu'à ce qu'on constate qu'elles n'ont plus d'effets significatifs. Cependant, lorsqu'un point fixe est atteint, les domaines solutions peuvent contenir encore un grand nombre de valeurs inconsistantes. On peut alors faire des bissections pour les éliminer.

Voici par exemple un problème de calibration magnétique de centrale inertielle de robot sous-marin rencontré sur l'AUV *SAUC'ISSE* qu'une propagation de contraintes par intervalles suivie de bissections a permis de résoudre rapidement (d'autres exemples de robotique similaires existent, par exemple pour la calibration de caméras [Telle, 2003]). La situation est la suivante : on veut que le sous-marin fasse des allers-retours dans une piscine (grâce notamment à une régulation en cap utilisant la centrale inertielle du robot). Le problème est que lorsqu'on donne le cap θ pour un aller et le cap $\theta + \pi$ pour le retour, on constate que le robot suit un cap $\theta + \pi + \alpha$, avec α suffisamment grand pour être gênant (de l'ordre d'une vingtaine de degrés). Ce problème est dû aux perturbations magnétiques liées aux éléments du sous-marin lui-même. En effet, le champ magnétique dû aux éléments perturbateurs s'additionne au champ magnétique terrestre et dévie donc de manière différente la direction du champ magnétique mesuré par la centrale inertielle selon l'orientation par rapport au Nord. Pour obtenir une évaluation de cette perturbation et pouvoir la compenser, on peut prendre des mesures $\theta_{m,i}$ de l'angle mesuré par la centrale à des angles connus (du robot par rapport au Nord magnétique) θ_i :

θ_i (deg)	$\theta_{m,i}$ (deg)	θ_i (rad)	$\theta_{m,i}$ (rad)	$[\theta_i]$ (rad)	$[\theta_{m,i}]$ (rad)
0	-164.5	0.00	-2.87	$[-0.05, 0.05]$	$[-2.92, -2.82]$
45	-129.3	0.79	-2.26	$[0.74, 0.84]$	$[-2.31, -2.21]$
90	-94	1.57	-1.64	$[1.52, 1.62]$	$[-1.69, -1.59]$
135	-56.5	2.36	-0.99	$[2.31, 2.41]$	$[-1.04, -0.94]$
180	-9	3.14	-0.16	$[3.09, 3.19]$	$[-0.21, -0.11]$
225	43.5	3.93	0.76	$[3.88, 3.98]$	$[0.71, 0.81]$
270	106.5	4.71	1.86	$[4.66, 4.76]$	$[1.81, 1.91]$
315	156	5.50	2.72	$[5.45, 5.55]$	$[2.67, 2.77]$
360	195.5	6.28	3.41	$[6.23, 6.33]$	$[3.36, 3.46]$

(3.30)

Il faut noter que :

- La centrale inertielle du robot est tournée d'environ π par rapport au robot.
- Les angles connus ont été trouvés par rapport aux murs du bâtiment où les tests étaient faits.
- On sait que le Nord géographique se trouve à environ -2 ° (-0.03 rad) du Nord magnétique à notre position [Magnetic Declination, 2011].
- Grâce à Google Maps, on peut connaître l'angle des bâtiments par rapport au Nord géographique (6 ° ou 0.10 rad dans notre cas, d'où 4 ° ou 0.07 rad par rapport au Nord magnétique).
- L'erreur sur les angles est supposée inférieure à 3 ° *i.e.* 0.05 rad (pour obtenir les intervalles $[\theta_i]$ et $[\theta_{m,i}]$).

En traçant la courbe (voir figure 3.6) des valeurs $\theta_i - \theta_{m,i} - \pi$ en fonction des $\theta_{m,i}$ (correspond à l'évolution de l'erreur d'estimation de l'orientation du robot en fonction des mesures indiquées par sa centrale inertielle) avec Scilab par exemple, on s'aperçoit qu'on peut considérer que

$$\theta = \theta_m + p_3 + p_1 \cos(\theta_m + p_2) \tag{3.31}$$

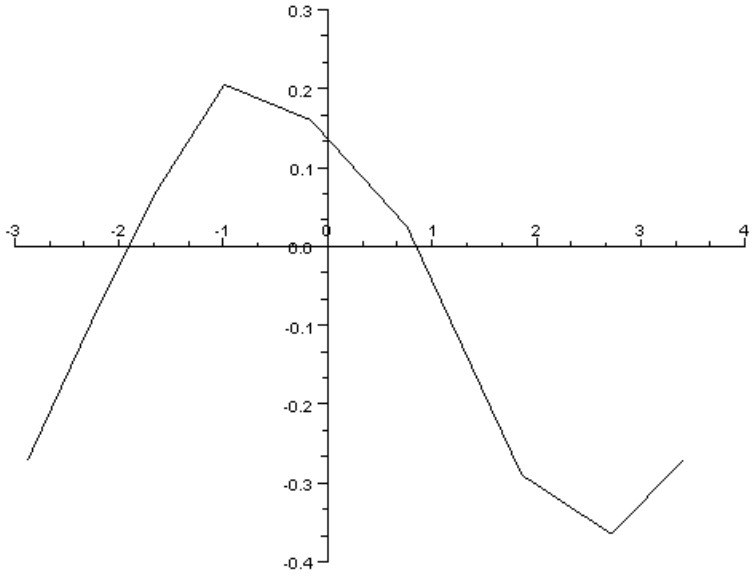


Figure 3.6 – Evolution de l'erreur d'estimation de l'orientation du robot en fonction des mesures indiquées par sa centrale inertielle ($\theta_{m,i}$ en abscisses et $\theta_i - \theta_{m,i} - \pi$ en ordonnées).

avec p_3 proche de π (on sait que la centrale inertielle est tournée d'environ π par rapport au robot). On a donc à résoudre un problème d'estimation de paramètres. Ceci peut être fait par intervalles en utilisant le logiciel Proj2D [Dao et al., 2004]. Le problème est rentré sous la forme d'un CSP (voir figure 3.7), où les variables que l'on cherche à obtenir précisément sont p_1 et p_2 (on suppose $p_3 = \pi$ dans un premier temps). On constate que malgré la grande marge d'erreur qu'on s'est donné (on considère une erreur de $0.05\,^\circ$ sur toutes les mesures qu'on a prises), l'ensemble solution est très petit (voir figure 3.8) et devient rapidement vide si on diminue la longueur des intervalles en entrée. Pourtant, la courbe formée par les mesures et la courbe avec les valeurs de paramètres trouvées ($p_1 = 0.27, p_2 = 0.63$) sont relativement éloignées (voir figure 3.9). Ceci est dû au fait qu'on ait supposé p_3 parfaitement connu pour simplifier, car Proj2D est fait pour afficher des résultats pour des problèmes à 2 dimensions. Pour des résolutions numériques de problèmes d'estimation de paramètres en plus grande dimension, on peut utiliser qSolve (voir section 3.5). En considérant p_3 variable, on peut alors arriver à un résultat plus exploitable (voir figure 3.11) en écrivant un script simple en langage QUIMPER (voir figure 3.10) et en appelant qSolve comme indiqué dans la figure 3.12. On peut alors choisir une ligne (n'importe laquelle théoriquement) du fichier de résultats figure 3.13, par exemple p_1 =mid($[0.285350502328302, 0.289007541394344]$), p_2 =mid($[0.566253458491941, 0.569544098712093]$), p_3 =mid($[3.06449103439268, 3.06587111868666]$). Le tracé de la courbe nous permet de vérifier que c'est un bon choix.

3.5 Bibliothèques et outils pour faire du calcul par intervalles

Plusieurs bibliothèques logicielles et programmes permettent d'utiliser le calcul par intervalles pour résoudre divers problèmes. Proj2D [Dao et al., 2004] (utilisé dans l'un des exemples précédent) est l'un des

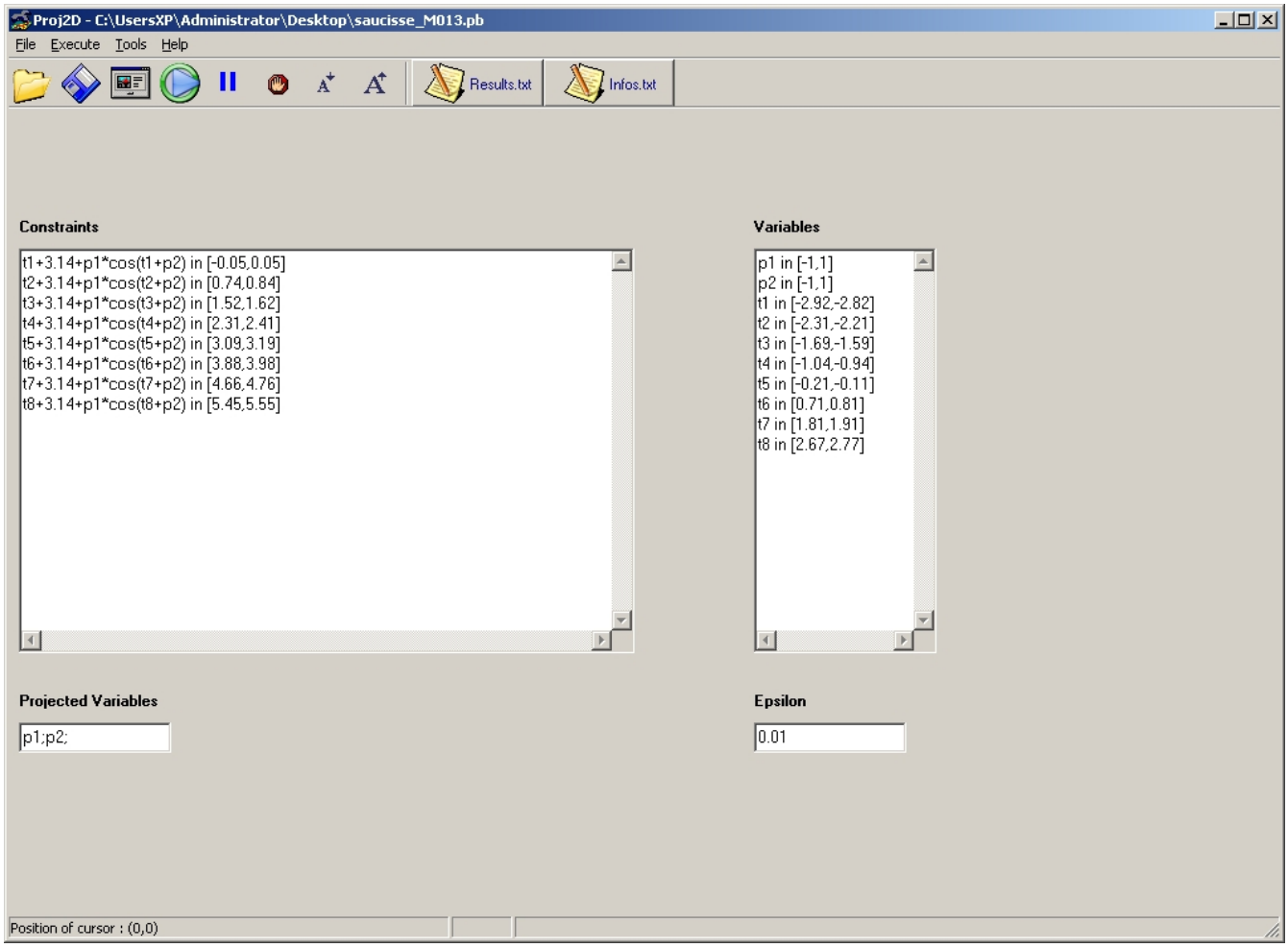


Figure 3.7 – CSP rentré dans Proj2D.

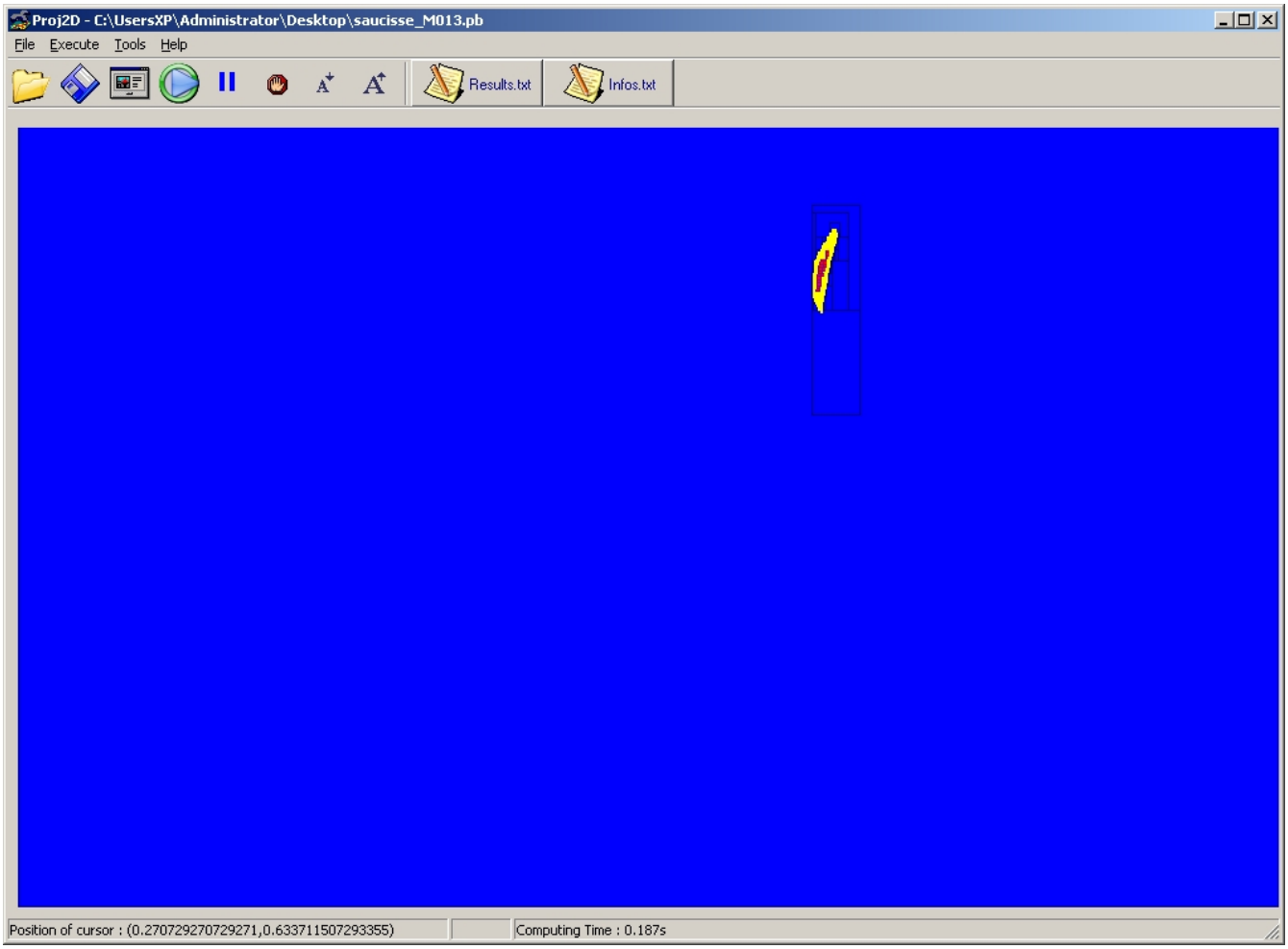


Figure 3.8 – Ensemble solution dans l'espace des paramètres (p_1, p_2) trouvé par PROJ2D (le temps de calcul est inférieur à la seconde sur un ordinateur à base de processeur Intel Core 2 Duo).

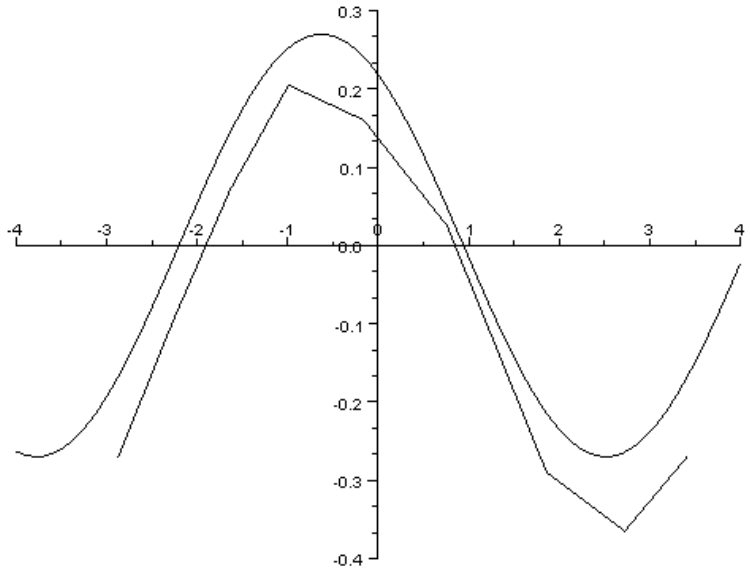


Figure 3.9 – Superposition du modèle théorique (en prenant p_1, p_2 inconnus avec PROJ2D) et de la courbe des points de mesures.

programmes les plus simple à utiliser pour résoudre des problèmes d'estimation de paramètre (limité à 2 paramètres pour la partie graphique). Une démonstration de problèmes classiques résolus par calcul par intervalles est disponible avec SETDEMO [Baffet and Jaulin, 2011]. Une petite bibliothèque intervalles C++ est disponible dans le code source de ces programmes. Un langage spécial dédié à la programmation par contraintes et des programmes (QPAVE, QTRAJ, QSOLVE) affichant les résultats graphiquement existent (voir l'exemple de la section précédente et 4.4) : QUIMPER [Chabert, 2007a] [Chabert and Jaulin, 2009] et la bibliothèque C++ IBEX (bibliothèque C++ sous-jacente de QUIMPER), elle-même basée sur la bibliothèque de calcul par intervalles garantie (gère correctement les arrondis liés aux calculs flottants) PROFIL/BIAS (Programmer's Runtime Optimized Fast Interval Library / Basic Interval Arithmetic Subroutines) [Knüppel, 1993]. Toutes les fonctions mathématiques et opérateurs de base sont implémentés pour les nombres réels, les intervalles et les pavés. De plus, des algorithmes couramment utilisés tels que la propagation *forward-backward* (nommé HC4REVISE dans IBEX, voir sous-sections 3.4.1, 4.3.1, 5.2.2...), 3BCID (voir sous-sections 3.4.2, 4.4...)... sont directement disponibles pour le programmeur.

3.6 Conclusion du chapitre

Les intervalles, leurs généralisations aux vecteurs et trajectoires et les principes de propagation, contraction et bissection ont été présentés dans ce chapitre. Les tubes (intervalles de trajectoires) sont une nouveauté qui sera particulièrement utile en 5.2.

Dans ce chapitre, nous avons rappelé l'intérêt des méthodes de calcul par intervalles, et les idées qui ont conduit à leur mise au point. Une arithmétique simple et peu contraignante existe pour manipuler des intervalles et de nombreux exemples, notamment en robotique, ont déjà montré qu'elle pouvait être utilisée pour

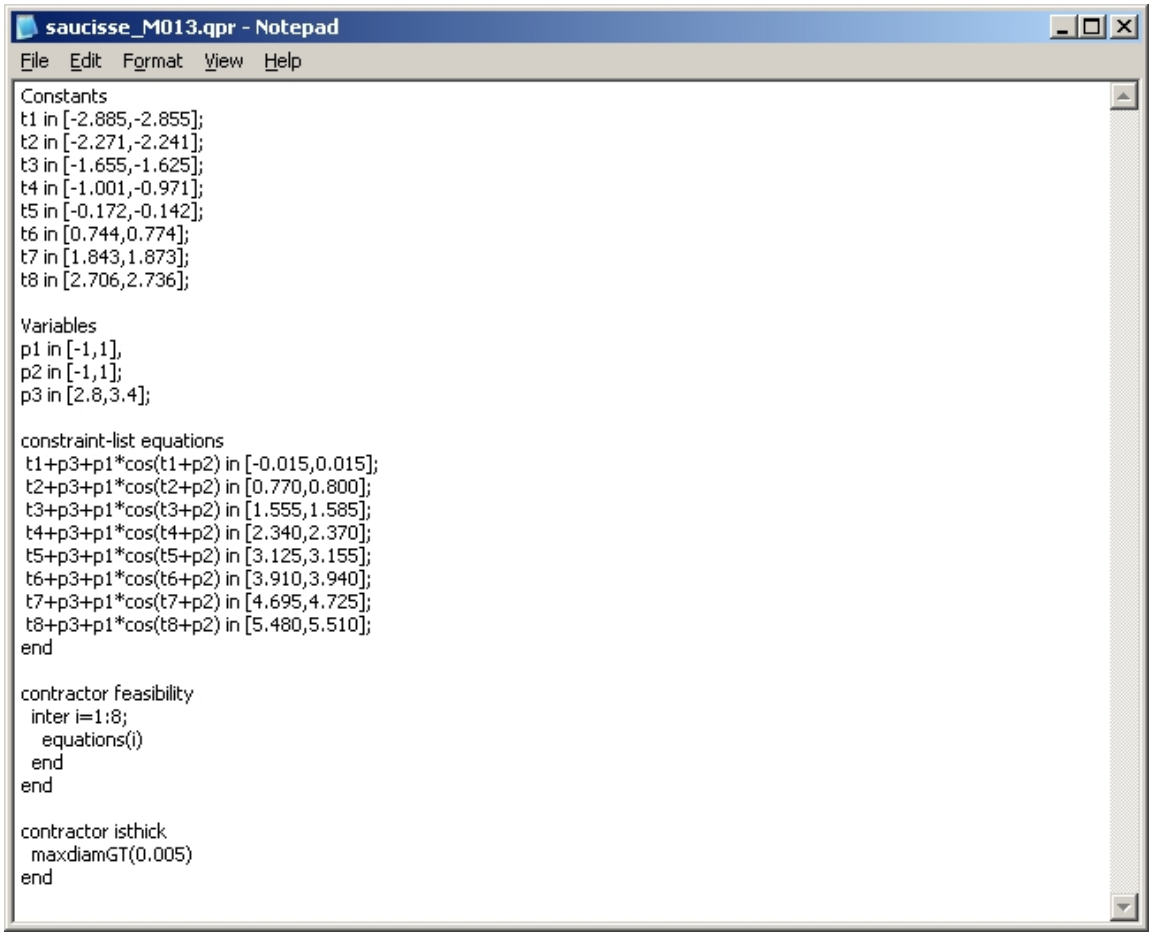


Figure 3.10 – Script en langage QUIMPER utilisé avec QSOLVE.

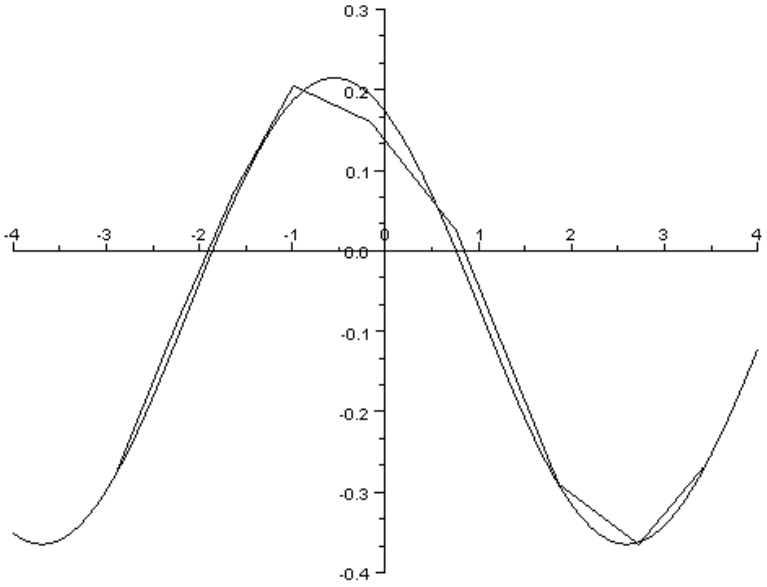


Figure 3.11 – Superposition du modèle théorique (en prenant p_1, p_2, p_3 inconnus avec QSOLVE) et de la courbe des points de mesures.

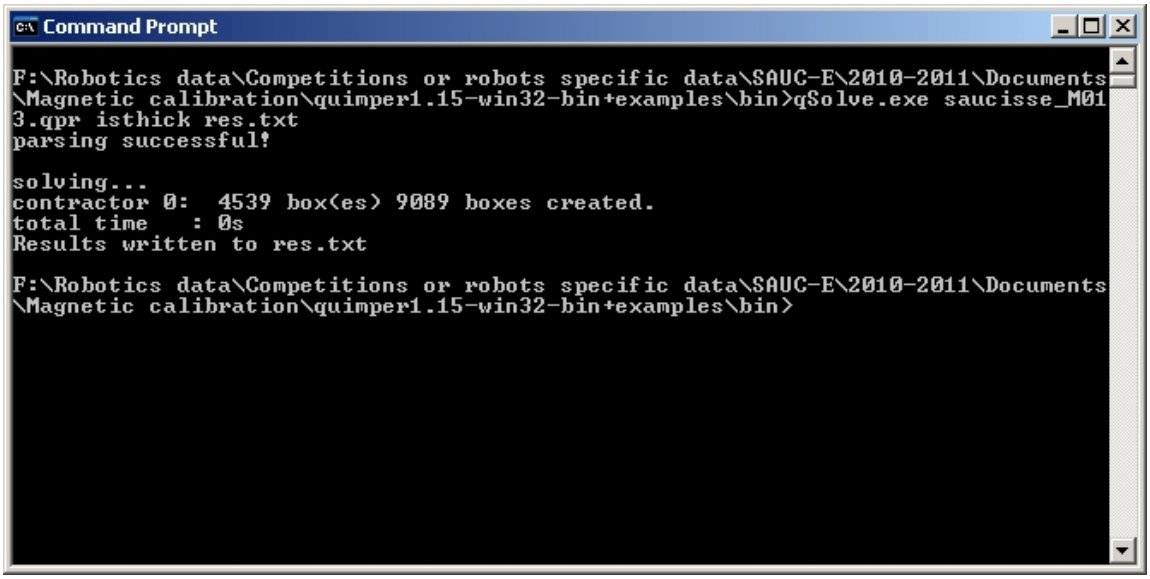


Figure 3.12 – Commande utilisée avec QSOLVE.

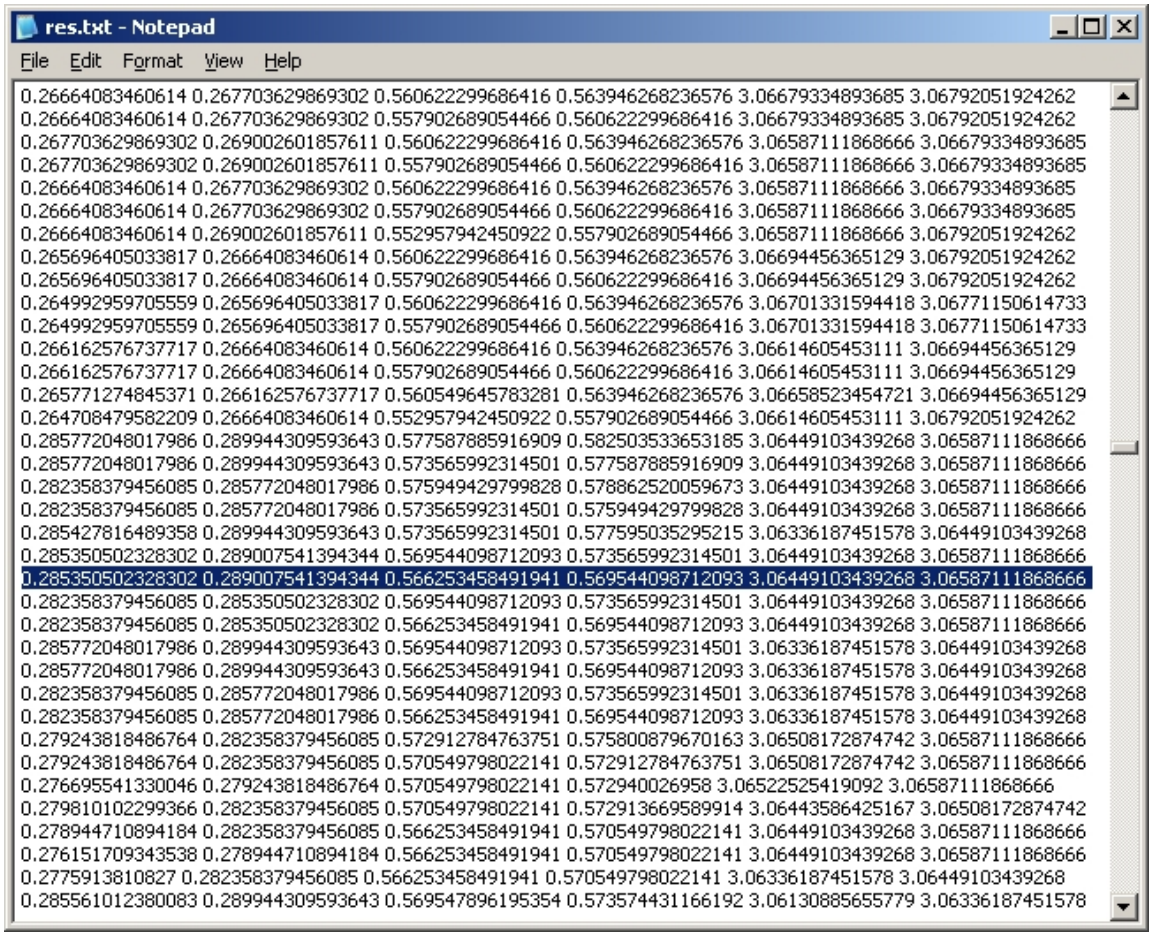


Figure 3.13 – Fichier de résultats généré par QSolve. Chaque ligne correspond à des intervalles $\left[p_1^-, p_1^+\right]$, $\left[p_2^-, p_2^+\right]$, $\left[p_3^-, p_3^+\right]$ solutions du problème.

résoudre des problèmes concrets. Deux types d'algorithmes complémentaires (propagation de contraintes et bissection) sont utilisés pour résoudre ces différents problèmes. Ceux-ci sont la base des algorithmes présentés dans la suite. De plus, les outils de programmation utilisés au cours de la thèse (et qui peuvent être utiles pour toute personne voulant résoudre des problèmes de manière garantie, comme les méthodes par intervalles le permettent) ont été indiqués, avec quelques exemples d'utilisation.

Chapitre 4

SLAM

Le principe du SLAM (Simultaneous Localization And Mapping), des algorithmes de résolution ainsi qu'une comparaison entre ceux-ci vont être présentés dans ce chapitre. Un programme permettant de faire du SLAM offline avec des données réelles de sous-marin sera notamment décrit.

La première section rappellera le principe du SLAM, et notamment son intérêt dans le milieu sous-marin et définira les conditions spécifiques qui seront considérées dans ce document.

La deuxième section présentera succinctement le principe des méthodes principales existantes n'utilisant pas les intervalles pour résoudre le problème du SLAM.

La troisième section sera consacrée au SLAM par intervalles par propagation de contraintes. Cette méthode, décrite et testée pour la première fois sur des données réelles de sous-marin dans [Jaulin, 2009a] a été appliquée au cours de la thèse sur un sous-marin supplémentaire, la *Daurade* (décrite dans la sous-section 2.2.1 et voir [Le Bars et al., 2010a]). Ceci a conduit à l'amélioration de GESMI, un logiciel créé à l'origine pour traiter le problème de SLAM offline du *Redermor* (pour de la recherche de mines sous-marines). En plus de la prise en compte des données de la *Daurade*, j'ai essayé de le rendre plus pratique pour qu'il corresponde un peu plus à ce qui pourrait être utilisé dans un contexte opérationnel.

Dans la quatrième section, une étude de l'optimalité de la méthode de SLAM offline par propagation de contraintes par intervalles sera faite à travers la comparaison de résultats obtenus par différents algorithmes sur une simulation de robot char (avec un modèle de déplacement similaire à celui des robots présentés dans la sous-section 2.3.1) présentée dans [Joly, 2010]. Ceci montrera d'une part que l'algorithme utilisé pour résoudre le SLAM sous-marin est réutilisable dans un contexte terrestre et que bien qu'il ne soit pas optimal, ses résultats sont un bon rapport entre temps de calcul, précision, simplicité d'adaptation...

La dernière section conclura le chapitre et fera le lien avec la suite du document.

4.1 Principe

Le problème du SLAM (voir [Leonard and Durrant-Whyte, 1992] pour l'une des premières formulations, [Durrant-Whyte and Bailey, 2006a] et [Durrant-Whyte and Bailey, 2006b] pour un état de l'art, aussi parfois appelé sous l'acronyme CML pour Concurrent Mapping and Localization ou encore Cooperative Mapping and Localization) pour un robot autonome se déplaçant dans un environnement inconnu est de construire une carte de cet environnement tout en utilisant cette carte simultanément pour calculer sa position. Un robot sous-marin autonome connait en général sa position initiale (lorsqu'il est à la surface grâce à un GPS), son modèle de déplacement (très approximativement) et possède quelques capteurs l'aidant à estimer sa position (écho-sondeur pour mesurer sa distance au fond, capteur de pression pour mesurer sa profondeur, loch Doppler pour mesurer sa vitesse, centrale inertielle pour mesurer son orientation) et voir ce qui l'entoure (sonar). Pourtant, malgré tous ces capteurs, plus il avance, plus ses erreurs d'estimation de position s'accumulent : le robot se perd. L'idée du SLAM (Simultaneous Localization And Mapping) est d'essayer de limiter ce problème en essayant de trouver des points de repères (amers) et relever leur position lorsqu'on arrive bien à se localiser (cartographie à partir de la localisation), puis d'utiliser la position de ces points de repères quand on repasse devant et qu'on est perdu pour essayer de retrouver la trajectoire qu'on a faite (localisation grâce à la cartographie). Le SLAM est donc fortement lié à 2 problèmes : la localisation, qui correspond à chercher sa position par rapport à un repère qu'on s'est défini, et la cartographie, qui est le calcul de la position des éléments constituant son environnement. Le SLAM est une sorte de localisation avec des repères mal connus ou même pas connus au départ.

Il y a plusieurs façons de faire du SLAM :

- Online : le robot embarque l'algorithme de SLAM et l'utilise pour effectuer sa mission. Ceci implique d'avoir mis au point un algorithme temps-réel.
- Offline : l'algorithme est utilisé à la fin de l'expérience, avec toutes les données. Dans ce cas de figure, les calculs n'ont pas besoin d'être immédiats (il n'est pas nécessaire d'être temps-réel). Par contre, un problème de mémoire (quantité de données à prendre en compte trop importante) peut se poser si on travaille sur toutes les données d'une longue expérience (cas traités dans cette thèse et [Jaulin, 2009a]).

De plus, d'autres conditions sont à définir :

- Association des données : lorsqu'un robot rencontre plusieurs fois le même point de repère, il n'est pas toujours facile pour lui de déterminer si c'est bien le même amer qu'il voit ou plusieurs amers identiques. Il arrive en effet parfois qu'on connaisse le nombre total d'amers et des instants où on sait qu'on a détecté un amer, sans savoir de quel amer il s'agissait (ce cas est traité dans [Chabert et al., 2009]). On dira que l'association des données est connue lorsqu'on sait quelle détection correspond à quel amer (c'est ce que nous supposerons dans cette thèse).
- Bearing-only, range-only, range and bearing (voir *e.g.* [Rives, 2009]) : les mesures liées aux amers peuvent parfois être seulement des angles (bearing-only, *e.g.* [Joly, 2010]), des distances (range-only, *e.g.* [Jaulin, 2011]) ou les 2.(des cas range and bearing et bearing-only seront abordés dans cette thèse).
- Amers mobiles, forme des amers : on pourrait imaginer que le robot ait besoin de localiser des objets mobiles (voir *e.g.* [Wang et al., 2007]), et qu'il se serve aussi de ces objets comme amers. Ce cas peut se produire si on travaille avec une meute de robots, où chaque robot pourrait servir à la localisation des autres (ce cas ne sera pas étudié dans cette thèse, les amers seront considérés ponctuels et immobiles).

– Prise en compte de données aberrantes : il est courant que des données capteurs soient totalement fausses, ce qui a souvent pour conséquence de fausser les positions trouvées (cas des méthodes non garanties) ou de ne pas trouver de positions possibles (cas des méthodes garanties). Il est possible de supposer que ce problème a déjà été résolu (ce que nous ferons dans les cas décrits dans cette thèse) ou que l'algorithme de SLAM le prend en compte automatiquement (il n'y a pas actuellement à ma connaissance d'algorithmes de SLAM faisant cela).

Nous verrons que le problème du SLAM peut être vu comme des problèmes d'estimation d'état et de paramètres, à erreur bornée dans le cas des intervalles (voir *e.g.* [Joly, 2010], [Drocourt et al., 2005], [Porta, 2005], [Di Marco et al., 2004], [Di Marco et al., 2001] pour le SLAM par intervalles dans le cas de robots roulants terrestres).

4.2 Méthodes existantes

Différentes méthodes existent pour traiter le problème du SLAM. La plupart des approches transforment le problème en estimation d'état pour un système dynamique, en incluant les positions des amers parmi les variables d'état à estimer ([Castellanos and Tardos, 1999], [Dissanayake et al., 2001], [Montemerlo et al., 2003], [Dufourd, 2005], [Joly, 2010]). Voir aussi [Williams et al., 2001], [Ruiz et al., 2004], [Eustice et al., 2005], [Ribas et al., 2010] pour le cas des robots sous-marins autonomes. Certaines solutions proposées sont basées sur des techniques d'estimation probabiliste (à base de filtre de Kalman, estimation Bayésienne, filtre de particules, voir [Thrun et al., 2005]), qui cherchent à mélanger les données avec les équations d'état du robot.

Les plupart des méthodes représentent tout d'abord les équations d'état du robot de cette manière [Chabert and Jaulin, 2011] :

$$\left\{ \begin{array}{rcl} \dot{\mathbf{x}} & = & \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} & = & \mathbf{g}(\mathbf{x}, \mathbf{u}). \end{array} \right. \tag{4.1}$$

Les équations liées aux détections d'amers $i \in \{1, \ldots, n\}$ peuvent ensuite s'écrire :

$$\mathbf{z}_i = \mathbf{h}_i(\mathbf{x}, \mathbf{u}, \mathbf{m}_i) \tag{4.2}$$

où $\mathbf{x}$ est le vecteur d'état du robot, $\mathbf{f}$ sa fonction d'évolution, $\mathbf{u}$ est le vecteur des commandes/entrées du système, $\mathbf{g}$ fonction d'observation/des mesures/sorties liées à l'état du robot et ses entrées, $\mathbf{y}$ le vecteur de ces mesures, $\mathbf{m}_i$ est le vecteur de position de l'amer i (par exemple $\mathbf{m}_i = (x_{mi}, y_{mi}, z_{mi})$), $\mathbf{h}_i$ la fonction des détections/mesures/sorties liées à l'état du robot, les entrées et la position de l'amer i (par exemple $\mathbf{h}_i(x, y, \theta, x_{mi}, y_{mi}, z_{mi}) = \begin{pmatrix} \arg\left(x_{mi} - x, y_{mi} - y\right) - \theta \\ \arg\left(\sqrt{(x_{mi} - x)^2 + (y_{mi} - y)^2}, z_{mi}\right) \\ \sqrt{(x_{mi} - x)^2 + (y_{mi} - y)^2 + z_{mi}^2} \end{pmatrix}$) et $\mathbf{z}_i$ le vecteur contenant ces mesures (par exemple $\mathbf{z}_i = (\alpha_{mi}, \beta_{mi}, d_{mi})$, voir figure 2.1).

En définissant l'état étendu $\mathbf{x}_e = (\mathbf{x}, \mathbf{m}_1, \ldots, \mathbf{m}_n)$, la sortie étendue $\mathbf{y}_e = (\mathbf{y}, \mathbf{z}_1, \ldots, \mathbf{z}_n)$ et les fonctions $\mathbf{f}_e$ et $\mathbf{g}_e$ correspondantes, on peut réécrire les équations sous la forme d'un problème d'estimation d'état

classique :

$$\begin{cases} \dot{\mathbf{x}}_e &= \mathbf{f}_e(\mathbf{x}_e, \mathbf{u}) \\ \mathbf{y}_e &= \mathbf{g}_e(\mathbf{x}_e, \mathbf{u}). \end{cases} \tag{4.3}$$

Ainsi, un observateur d'état bayésien (filtre de Kalman, particulaire, ensembliste...) peut reconstituer l'état étendu $\mathbf{x}_e$ et donc permettre de connaître la position du robot (localisation) et la position des amers (cartographie). Cependant, plusieurs problèmes peuvent se manifester :

- Les équations sont souvent fortement non-linéaires, ce qui est un problème pour les méthodes nécessitant une linéarisation (Kalman...).
- Le nombre d'équations est parfois très grand (particulièrement pour du SLAM offline), ce qui peut rendre le temps de calcul excessivement long (filtres particulaires...).

4.3 SLAM par intervalles

4.3.1 Principe

Dans le cas ensembliste avec l'arithmétique des intervalles, on peut écrire le problème du SLAM offline sans données aberrantes et avec l'association des données connue sous la forme d'un problème d'estimation d'état et de paramètres à erreur bornée, où $\mathbf{x}$ est le vecteur d'état à estimer et $\mathbf{m}_i$ des paramètres qui peuvent être connus ou à estimer (voir [Jaulin, 2009a], [Le Bars et al., 2010a]) :

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) & \text{(équation d'évolution)} \\ \mathbf{y} &= \mathbf{g}(\mathbf{x}, \mathbf{u}) & \text{(équation d'observation)} \\ \mathbf{z}_i &= \mathbf{h}_i(\mathbf{x}, \mathbf{u}, \mathbf{m}_i) & \text{(équation des détections d'amers).} \end{cases} \tag{4.4}$$

$\mathbf{x}$ est le vecteur d'état du robot, $\mathbf{u}$ est le vecteur des entrées, $\mathbf{y}$ est le vecteur des sorties, $\mathbf{m}_i$ est la position de l'amer i et $\mathbf{z}_i$ est le vecteur des mesures liées à l'amer i.

On considère que :

- A certains instants $t \in [t_0, t_f]$ où t_0 et t_f correspondent aux instants initial et final, on a un pavé borné $[\mathbf{x}](t)$ contenant le vecteur d'état :
$$\mathbf{x}(t) \in [\mathbf{x}](t). \tag{4.5}$$
- Pour tout $t \in [t_0, t_f]$, on a des pavés bornés englobant $\mathbf{u}(t)$ et $\mathbf{y}(t)$:
$$\forall t \in [t_0, t_f], \mathbf{u}(t) \in [\mathbf{u}](t) \text{ et } \mathbf{y}(t) \in [\mathbf{y}](t). \tag{4.6}$$
- On a un ensemble fini $\mathcal{M} \subset [t_0, t_f] \times \{1, \ldots, n\}$, n correspondant au nombre d'amers tel que si $(t, i) \in \mathcal{M}$, l'amer i a été détecté au temps t et on a des pavés bornés $[\mathbf{z}_i](t)$ tels que :
$$\mathbf{z}_i(t) \in [\mathbf{z}_i](t). \tag{4.7}$$
- Si certaines valeurs ne sont pas mesurées, l'intervalle correspondant est mis à $[-\infty, \infty]$.

Ceci a la forme d'un CSP, que nous pouvons résoudre par une propagation de contraintes (suivie éventuellement de bissections). Les étapes de résolution et les contracteurs utilisés sont les suivants :

Initialisation. Tous les domaines des variables sont initialisés avec les valeurs déjà connues. Par exemple, si on a une condition initiale sur le vecteur d'état, $[\mathbf{x}](t)$ sera déjà contracté à $t = t_0$.

Contracteur d'évolution. Une propagation *forward* suivie d'une propagation *backward* par rapport au temps (combinée avec une propagation *forward-backward* sur l'expression pour chaque pas de temps, en utilisant HC4Revise avec IBEX par exemple) est en général efficace (voir sous-section 3.4.1) pour les contraintes décrites par des équations différentielles comme $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ (voir [Jaulin, 2002a] pour le *forward-backward* sur une équation différentielle).

Contracteur d'observation et de détections. Les contractions de $[\mathbf{x}](t)$ et $[\mathbf{m}_i]$ par rapport aux contraintes $\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u})$ et $\mathbf{z}_i = \mathbf{h}_i(\mathbf{x}, \mathbf{u}, \mathbf{m}_i)$ peuvent être faites avec HC4Revise.

Comme pour tout algorithme de propagation de contraintes par intervalles, il faut répéter les contractions jusqu'à obtention d'un point fixe.

4.3.2 Application dans les cas des sous-marins *Daurade* et *Redermor*

Description des conditions de l'expérience et des données récupérées

Des expériences en lien avec la problématique de guerre des mines (détections de mines à la sortie d'un port) ont été réalisées avec les sous-marins *Daurade* [Le Bars et al., 2010a] et *Redermor* [Jaulin, 2009a] (voir figure 4.1). Dans ce contexte, les amers étaient soient des mines placées au fond de la mer pour l'exercice (cas de l'expérience avec le *Redermor*), soient des rochers, morceaux d'épaves ou autres points remarquables au fond de l'eau (cas de l'expérience avec la *Daurade*).

L'AUV *Daurade* a été lancé dans la baie de Douarnenez pour une expérience de quadrillage de zone d'environ 5 heures (de $t = 0$ à $t = 18445.75$ s). Pendant la mission, le robot a essayé de suivre une trajectoire prédéfinie par une liste de waypoints à atteindre en se localisant grâce à ses capteurs très précis et son filtre de Kalman embarqué. Ces points de passages ont été choisis par un opérateur humain avant la mission. L'erreur de l'algorithme de localisation embarqué était estimée à environ 50 m pour 10 km de trajet, ce qui est suffisant pour contrôler le robot pour une mission autonome dans l'océan sans risquer de le perdre. L'AUV a pris des données GPS à la surface 3 fois pendant la mission : pendant les premières secondes, au milieu de la mission (avant de changer la direction de ses allers-retours de Est-Ouest à Nord-Sud) et à la fin de la mission :

$$\begin{array}{ll} t = 0\,\text{s} & \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.332443 \\ -0.091429 \end{pmatrix} \text{m} \\ t = 9210\,\text{s} & \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 10.277683 \\ 663.195976 \end{pmatrix} \text{m} \\ t = 18419\,\text{s} & \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 875.402443 \\ 724.556278 \end{pmatrix} \text{m}. \end{array} \tag{4.8}$$

Un fichier contenant les données de navigation $x, y, z, \varphi, \theta, \psi, \dot{\varphi}, \dot{\theta}, \dot{\psi}, v_r^x, v_r^y, v_r^z, a$ et une estimation de leur erreur a été enregistré pendant la mission. De plus, 13 amers différents ont été vus dans 33 détections trouvées

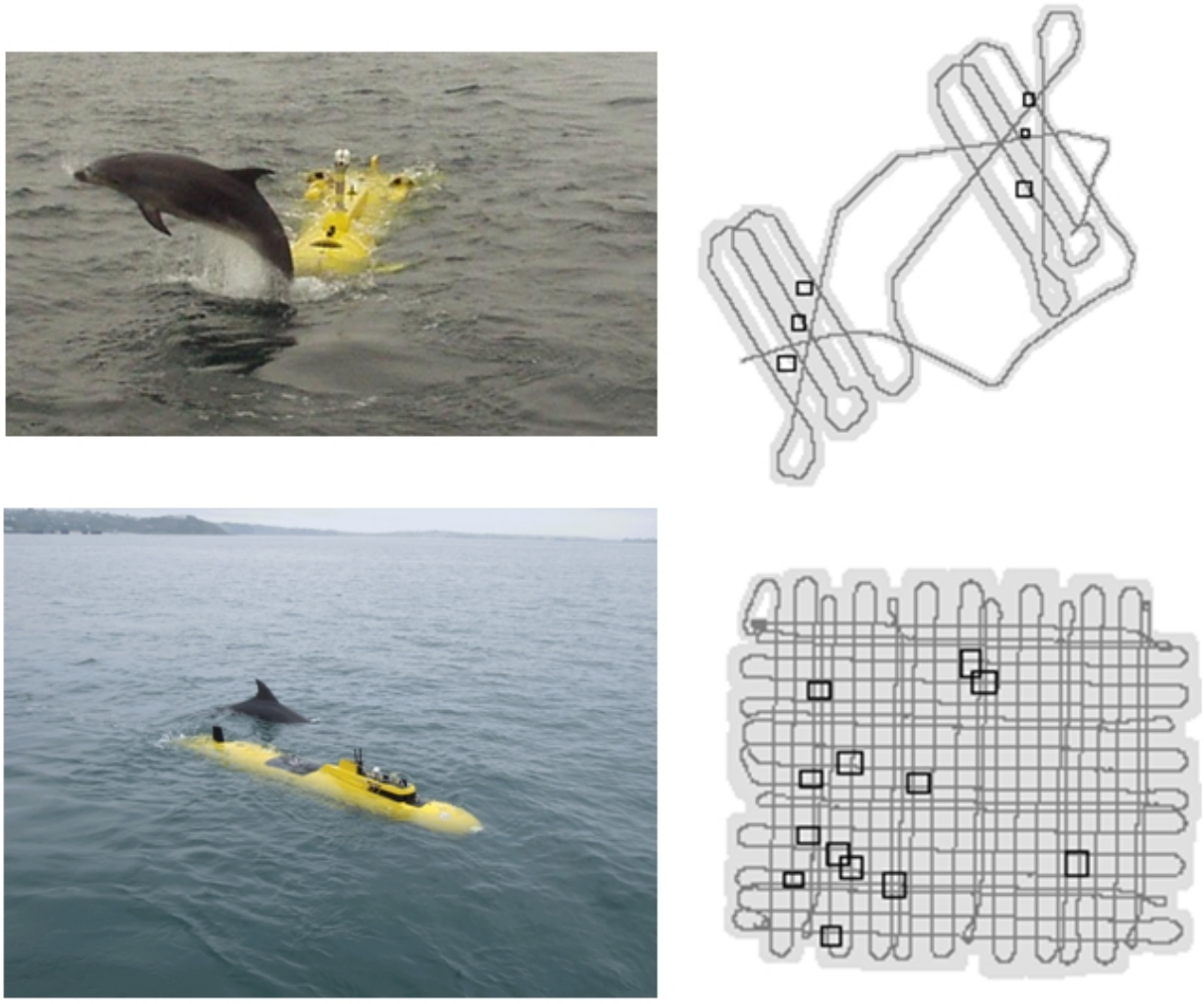


Figure 4.1 – En haut, le *Redermor* et sa trajectoire effectuée (vue du dessous), avec la position des amers (carrés noirs). En bas, la *Daurade* lors d'une expérience dans la baie de Douarnenez (photos DGA/GESMA).

sur l'image sonar. C'est un opérateur humain qui a examiné l'ensemble de l'image sonar avec SonarPro pour obtenir cette liste d'amers et détections (10 premières détections) :

t (en s)	i	d_m (en m)	α_m (en rad)
308	2	16.85	$\pi/2$
1019	2	55.22	$-\pi/2$
1237	4	53.04	$\pi/2$
1280	3	21.68	$\pi/2$
1459	5	73.31	$\pi/2$
1871	4	37.13	$-\pi/2$
2050	8	30.89	$-\pi/2$
2097	6	42.43	$\pi/2$
2155	3	32.76	$-\pi/2$
2335	5	29.79	$\pi/2$

. (4.9)

Mise en équation

L'équation d'évolution de la *Daurade* (ou le *Redermor*) peut être écrite de cette façon (voir 2.2.1 pour bien comprendre cette partie et [Jaulin, 2009a], [Le Bars et al., 2010a] pour les articles correspondants) :

$$\dot{\mathbf{p}}(t) = \mathbf{R}(\varphi(t), \theta(t), \psi(t)).\mathbf{v}_r(t) \tag{4.10}$$

où :

- $\mathbf{p}(t) = (x(t), y(t), z(t))$ sont les coordonnées du robot exprimées dans le repère de référence $(\mathbf{O}, \mathbf{i}, \mathbf{j}, \mathbf{k})$ où $\mathbf{O}$ est la position de mise à l'eau du robot, le vecteur $\mathbf{i}$ indique le Nord, $\mathbf{j}$ indique l'Est et $\mathbf{k}$ est orienté vers le centre de la Terre (voir figure 2.1). Le vecteur d'état du système est donc $\mathbf{x} = \mathbf{p}$.
- $\mathbf{R}(\varphi(t), \theta(t), \psi(t))$ représente la matrice de rotation définie à partir des angles de cap ψ, de tangage θ et roulis φ du sous-marin mesurés par son INS.
- $\mathbf{v}_r(t)$ représente la vitesse du robot mesurée par le DVL et exprimée dans le repère mobile lié au sous-marin. Le vecteur des entrées du système est donc $\mathbf{u} = (\varphi, \theta, \psi, v_r^x, v_r^y, v_r^z)$.

L'équation d'observation est :

$$\mathbf{y}(t) = \mathbf{p}(t) \tag{4.11}$$

car x et y peuvent être mesurés parfois par le GPS et z est obtenu à partir du capteur de pression.

L'équation des détections d'amers est plus compliquée. Si on note

$$\mathbf{e}_i(t) = \mathbf{p}(t) - \mathbf{m}_i \tag{4.12}$$

le vecteur distance robot-amer exprimé dans le repère de référence $(\mathbf{O}, \mathbf{i}, \mathbf{j}, \mathbf{k})$ (avec $\mathbf{m}_i = (x_{mi}, y_{mi}, z_{mi})$ la position de l'amer i dans ce repère) et

$$\mathbf{e}_{ri}(t) = \mathbf{R}^{\mathrm{T}}(\varphi(t), \theta(t), \psi(t)).\mathbf{e}_i(t) \tag{4.13}$$

son expression dans le repère du robot $(\mathbf{p}(t), \mathbf{i}_r(t), \mathbf{j}_r(t), \mathbf{k}_r(t))$, nous avons les relations suivantes (qui sont semblables aux équations trouvées en 2.2.1, mais avec plusieurs amers cette fois-ci) :

$$\left\{\begin{array}{ll} \text{(i)} & r_i(t) = ||\mathbf{e}_i(t)|| = d_{mi}(t) \text{ si } (t,i) \in \mathcal{M} \\ \text{(ii)} & \mathbf{e}_{ri}(t) = \left(0, -\sin(\alpha_{mi}(t)) \sqrt{d_{mi}^2(t) - a^2(t)}, -a(t)\right) \\ & \text{avec } \alpha_{mi}(t) = \frac{\pi}{2} \text{ (amer à tribord) ou } -\frac{\pi}{2} \text{ (amer à bâbord) si } (t,i) \in \mathcal{M} \\ \text{(iii)} & < \mathbf{e}_i(t), \mathbf{k} >= z(t) - z_{mi} \text{ et } < \mathbf{e}_i(t), \mathbf{k} >= -a(t) \text{ si } (t,i) \in \mathcal{M}. \end{array}\right. \tag{4.14}$$

L'égalité (i) signifie que lorsque l'amer $\mathbf{m}_i$ est détecté, la distance robot-amer $r_i(t)$ mesurée sur l'image sonar correspond à la distance robot-amer $d_{mi}(t)$ (définie pour tout t) qui est la norme du vecteur $\mathbf{e}_i(t)$. L'égalité (ii) (amer forcément sous le robot et dans un plan qui lui est perpendiculaire) est obtenue en utilisant le théorème de Pythagore (en supposant qu'il y a peu de roulis et tangage lorsque le sous-marin prend des données sonar) qui permet d'obtenir une expression de $\mathbf{e}_i(t)$ dans le repère du robot $(\mathbf{p}(t), \mathbf{i}_r(t), \mathbf{j}_r(t), \mathbf{k}_r(t))$, avec $a(t)$ la distance entre le robot et le fond en suivant la direction $\mathbf{k}$ (altitude mesurée par le DVL). Les 2 dernières équations sont obtenues en supposant que les amers sont au fond de l'océan et que celui-ci est plat (au moins localement).

Ces équations peuvent être mises sous la forme $\mathbf{z}_i = \mathbf{h}(\mathbf{p}, \varphi, \theta, \psi, \mathbf{m}_i)$ et le vecteur des mesures liées aux amers est donc $\mathbf{z}_i = (d_{mi}, \alpha_{mi}, z_{mi})$. Au final, les équations du SLAM pour la *Daurade* peuvent être écrites :

$$\left\{\begin{array}{lcll} \dot{\mathbf{p}} & = & \mathbf{R}(\varphi,\theta,\psi).\mathbf{v}_r & \text{(équation d'évolution)} \\ \mathbf{y} & = & \mathbf{p} & \text{(équation d'observation)} \\ \mathbf{z}_i & = & \mathbf{h}(\mathbf{p}, \varphi, \theta, \psi, \mathbf{m}_i) & \text{(équation des amers),} \end{array}\right. \tag{4.15}$$

ce qui correspond à la forme de l'équation 4.4.

Puisque $\dot{\mathbf{p}}(t) = \mathbf{R}(\varphi(t), \theta(t), \psi(t)).\mathbf{v}_r(t)$ et $\mathbf{e}_i(t) = \mathbf{p}(t) - \mathbf{m}_i$, on peut noter qu'on a aussi :

$$\dot{\mathbf{e}}_i(t) = \mathbf{R}(\varphi(t), \theta(t), \psi(t)).\mathbf{v}_r(t). \tag{4.16}$$

Cette équation pourrait aussi être utilisée pour contracter les domaines.

Résolution avec GESMI

Présentation GESMI (Guaranteed Estimation of Sea Mines with Intervals) est un programme permettant d'estimer la trajectoire d'un robot sous-marin et la position d'objets particuliers (préalablement détectés au moins 1 fois par un opérateur humain en parcourant les données sonar) en utilisant le calcul par intervalles (voir [Jaulin et al., 2006] et [Jaulin et al., 2007] pour la toute première version et [Le Bars et al., 2010a], figure 4.2 pour la nouvelle version que j'ai faite au cours de la thèse). Il permet en même temps de vérifier la cohérence entre les données qui lui sont fournies en entrée et d'aider à la détection des objets sur l'image sonar.

Les données dont il a besoin pour fonctionner sont les suivantes :

– Données de navigation du sous-marin (angles φ, θ, ψ du sous-marin, altitude a, profondeur z, vitesses dans le repère mobile (v_r^x, v_r^y, v_r^z), quelques positions GPS connues x, y, voir figure 4.3).

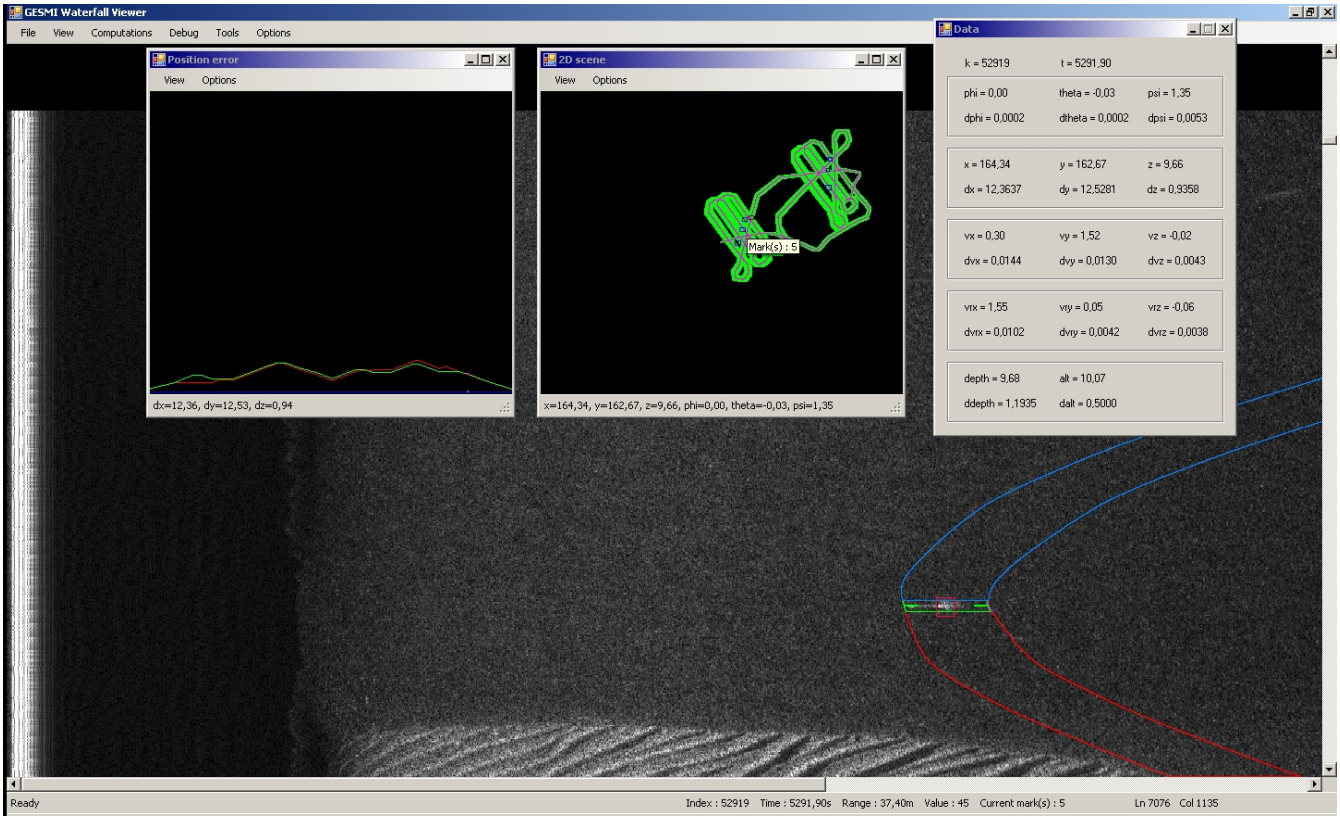


Figure 4.2 – Capture d'écran de GESMI, programme qui calcule et affiche la position des mines, l'enveloppe de la trajectoire, l'erreur en position d'un sous-marin lors d'une expérience de quadrillage de zone inconnue, et qui aide à identifier les mines parmi les différentes détections.

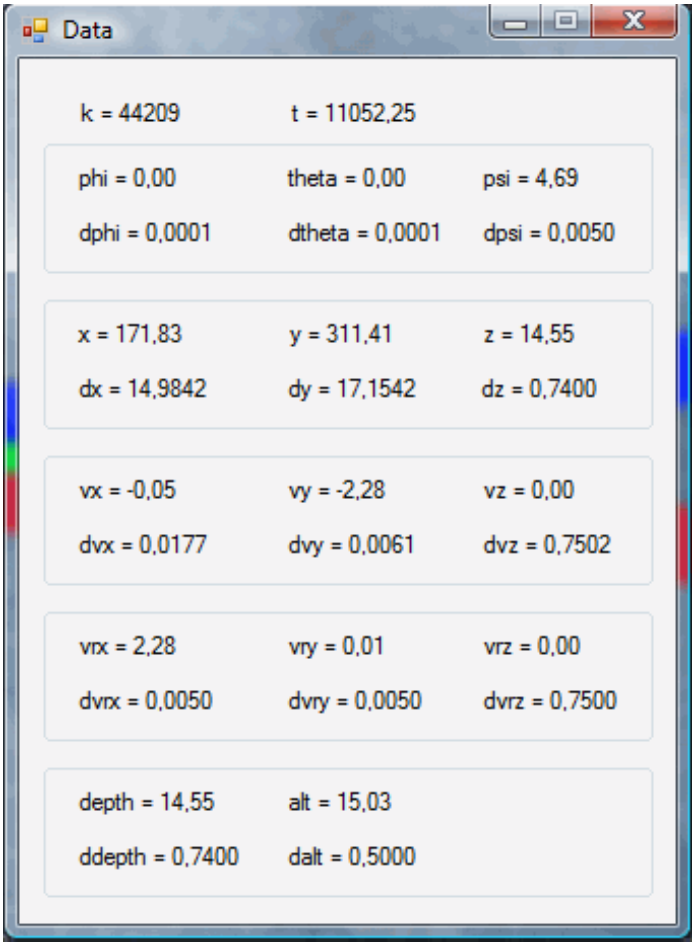


Figure 4.3 – Fenêtre de GESMI montrant les données de navigation à un instant t donné.

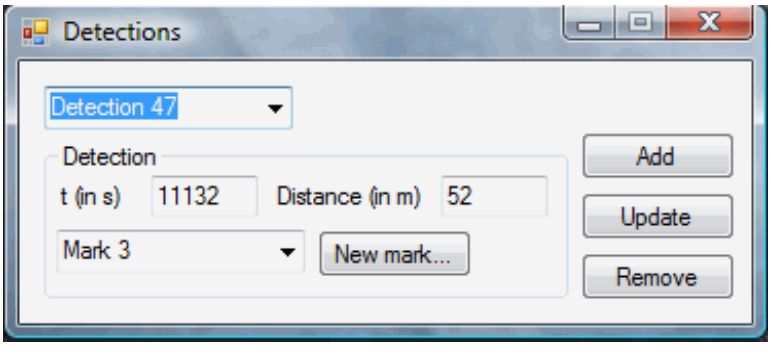


Figure 4.4 – Fenêtre de GESMI permettant de voir et modifier les données relatives aux détections d'amers.

– La distance d_m sur l'image sonar où un amer i a été détecté et s'il est sur tribord ou bâbord (α_m), voir figure 4.4.
– Les temps et estimation d'erreur maximale pour chacune des données.
– La portée et l'erreur en distance du sonar.
– Les images sonar (seulement utilisées pour l'affichage dans cette partie).

Les données qu'il produit ou affiche sont les suivantes :

– La trajectoire du sous-marin et la position des amers (sous forme d'enveloppe et centre de l'enveloppe ainsi que d'intervalles, voir figures 4.5 et 4.6).

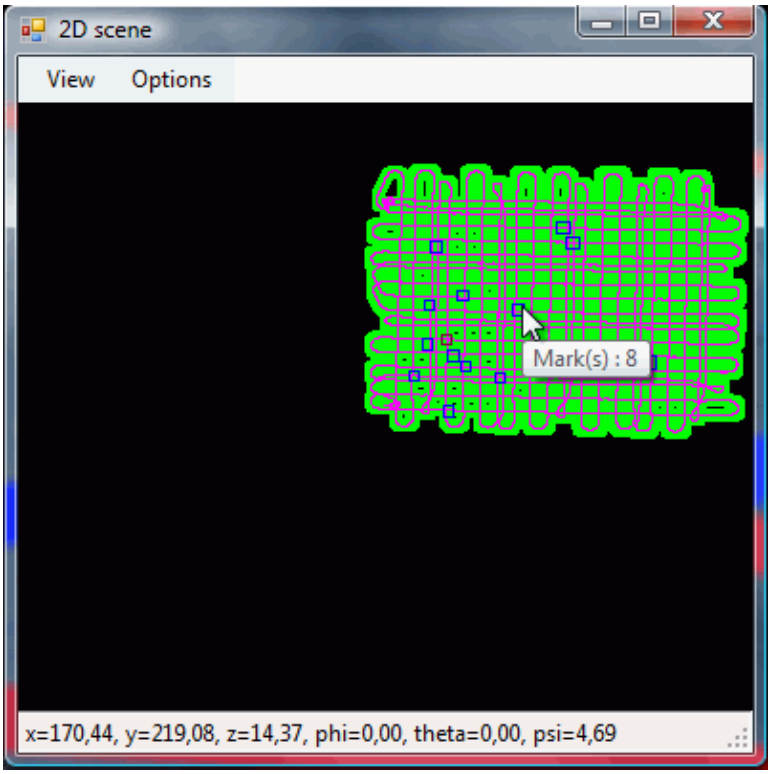


Figure 4.5 – Fenêtre de GESMI permettant de voir graphiquement la trajectoire du sous-marin (en 2D et vue du dessous) et les positions des amers. Les pavés enveloppant la position du sous-marin à chaque instant t sont dessinés en vert, et leur centre est indiqué en rose. Le pavé correspondant à la position courante du sous-marin (que l'on peut faire varier en bougeant le curseur de la souris sur les différentes fenêtres de GESMI) est dessiné en bordeau. Les pavés enveloppant les positions des amers sont dessinés en bleu.

– L'erreur en position au cours du temps (voir figure 4.7).
– La reconstitution de l'image sonar indiquant les distances sous-marin à amers, montrant les endroits où le sous-marin est passé à proximité des amers (voir figure 4.8).

J'ai repris et amélioré ce programme (initialement développé par Luc JAULIN et présenté dans [Jaulin et al., 2006] et [Jaulin et al., 2007]), notamment pour permettre son utilisation avec les données de la *Daurade* (rajout de la gestion de la partie bâbord des images sonar...), faciliter l'interprétation des résultats (affichage des erreurs en position...) et aider au maximum l'opérateur humain dans sa recherche de

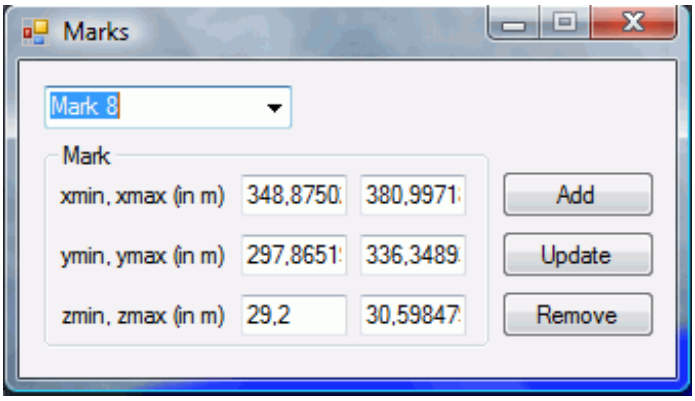


Figure 4.6 – Fenêtre de GESMI permettant de voir les positions des amers déterminées par le programme, sous forme d'intervalles.

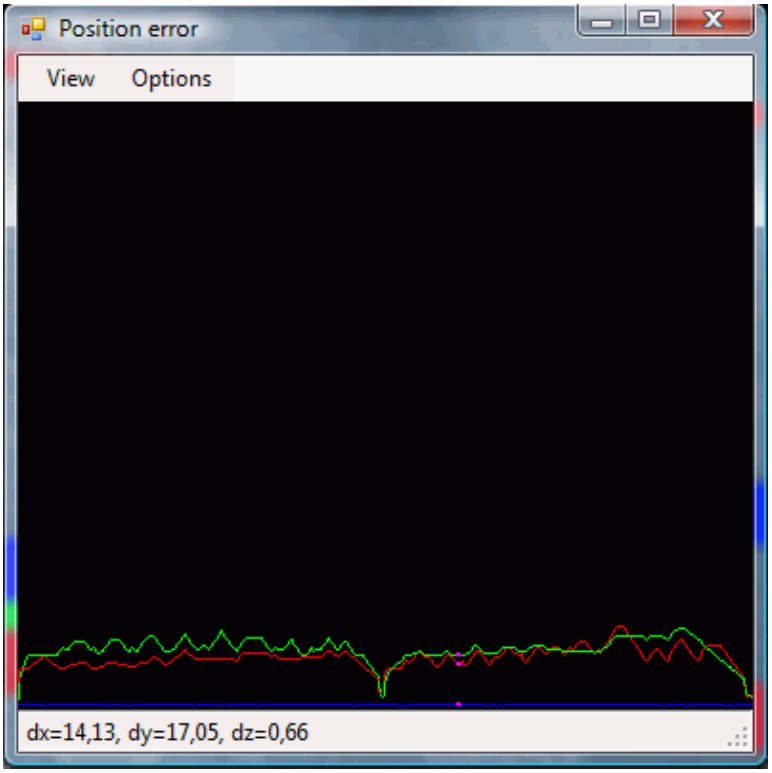


Figure 4.7 – Fenêtre de GESMI permettant de voir l'évolution de l'erreur (axe vertical en m) d'estimation des positions x, y, z du sous-marin (x en rouge, y, en vert et z en bleu) en fonction du temps (axe horizontal en s).

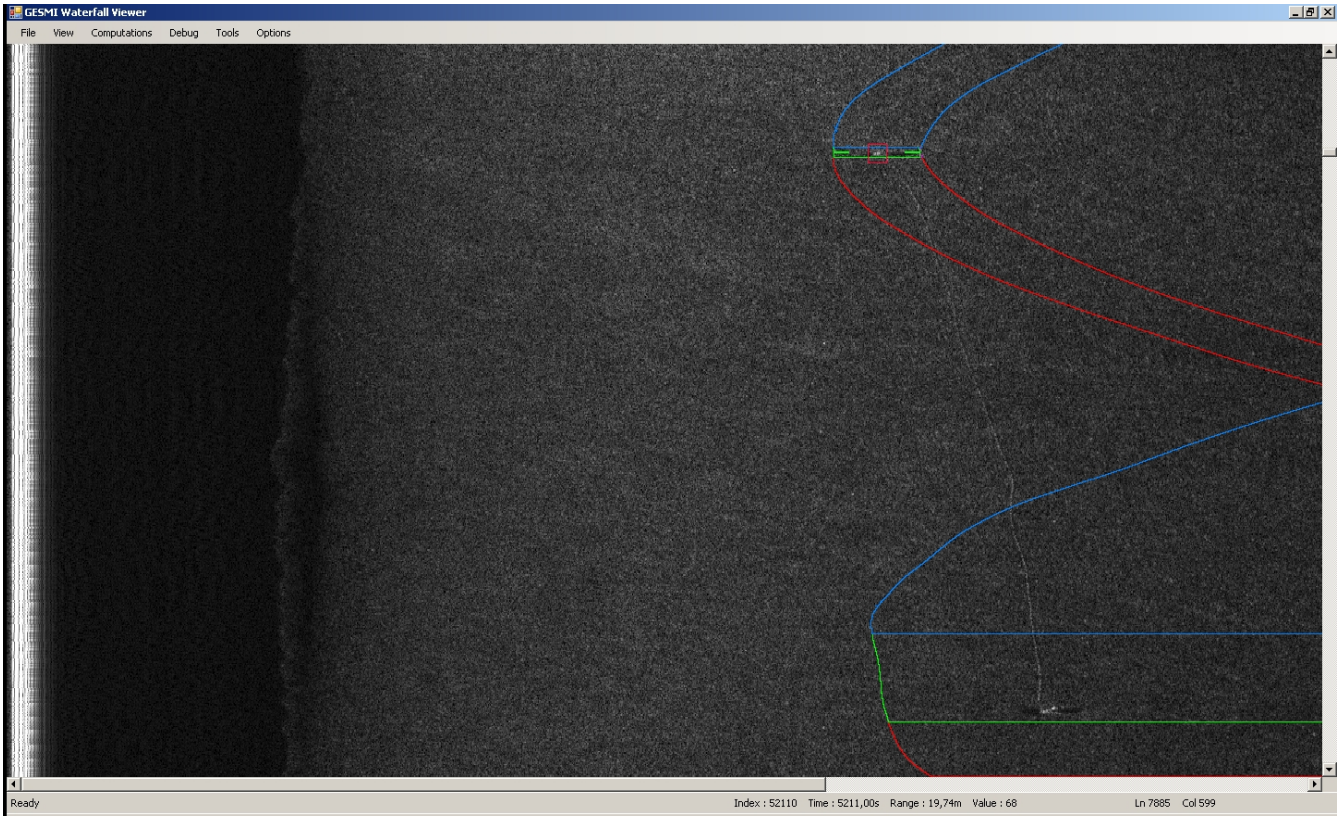


Figure 4.8 – Fenêtre de GESMI montrant l'image sonar tribord et les zones où il devrait y avoir un amer. Les enveloppes vertes, bleues et rouges représentent la distance robot-amer (axe horizontal) en fonction du temps (axe vertical) : en rouge l'amer est plutôt devant le sous-marin et en bleu plutôt derrière. C'est le fait qu'à un moment l'amer était devant et qu'à un autre moment il se trouve derrière qui nous permet (par le théorème des valeurs intermédiaires) de déterminer qu'à un instant t, le sonar aurait dû voir l'amer (zone entourée de vert). C'est aussi à cet instant que la distance robot-amer est localement minimale. Sur cette image, on distingue deux mines : une en haut à droite, qui a été détectée par l'opérateur humain (les mines avec un petit carré rouge sont celles que l'utilisateur a détecté lui-même) et une autre en-dessous (non détectée par l'opérateur, mais que GESMI met en évidence à l'intérieur de l'enveloppe verte). On voit aussi un câble les reliant.

mines (amélioration des indications de positions estimées de mines directement sur l'image sonar...).

Concepts de GESMI GESMI utilise à la fois les données de navigation (vitesses, angles...) et les détections d'amers vus par un opérateur humain sur les images sonar pour estimer le plus précisément possible une enveloppe de la trajectoire du sous-marin et de la position réelle de ces amers. La cohérence entre les données de navigation et les détections est vérifiée en même temps.

Dans le programme, toutes les données sont représentées par des intervalles, pavés ou tubes. Par exemple, si à $t = 308.0\,\text{s}$ la centrale inertielle du sous-marin indique un angle $\theta = -0.0258\,\text{rad}$ et que la documentation du constructeur de la centrale inertielle précise que son erreur maximale est de $0.0001\,\text{rad}$, l'angle θ à $t = 308.0\,\text{s}$ sera représenté dans le programme par l'intervalle $[-0.0258 - 0.0001, -0.0258 + 0.0001] = [-0.0259, -0.0257]$. Tous les intervalles obtenus à partir des fichiers de données chargés par le programme sont ensuite utilisés dans des calculs mettant en jeu les équations d'état du sous-marin ainsi que d'autres relations géométriques. Ces équations et relations sont discrétisées (par rapport au temps) et mettent en jeu des additions, soustractions, cos, sin... d'intervalles de la même façon qu'avec des nombres réels, ainsi que des intersections et unions d'intervalles. Si un intervalle vide est trouvé lors d'un calcul, cela signifie en général qu'il y a une incohérence dans les données de navigation ou détections. Le plus souvent, elle est due à des données capteur moins précises que l'on croyait. Il faut donc augmenter l'erreur avec laquelle on considère ces données. Des détections imprécises d'objets sur l'image sonar peuvent aussi provoquer des incohérences. Dans ce cas il faut vérifier les détections et éventuellement éliminer celles qui sont trop approximatives pour pouvoir les corriger par la suite à l'aide de la reconstitution de l'image sonar.

Les 3 principales étapes de calcul pour la reconstitution de la trajectoire du sous-marin sont les suivantes :

- Propagation (menu *Computations\Contract forward*) : les équations d'état du sous-marin donnent une relation entre les positions et données de navigation à t et $t + dt$. Des calculs sont donc faits en partant de $t = 0\,\text{s}$ vers la fin. Par exemple, la position GPS du sous-marin à $t = 0\,\text{s}$ va permettre d'estimer ses positions suivantes avec une erreur qui augmente au cours du temps et qui dépend aussi de l'erreur avec laquelle on connait ses différentes données de navigation (vitesses, angles...).
- Retro-propagation (menu *Computations\Contract backward*) : les équations d'état du sous-marin peuvent donner aussi une relation entre les positions et données de navigation à $t + dt$ et t. Des calculs sont donc faits en partant de la fin vers $t = 0\,\text{s}$. Par exemple, la position GPS finale du sous-marin va permettre d'améliorer l'estimation de ses positions précédentes.
- Mise en cohérence avec les objets particuliers détectés sur l'image sonar (menu *Computations\ Contract marks*) : des relations géométriques entre les positions du sous-marin et celles des objets à leurs temps de détection sur l'image sonar permettent d'une part d'estimer la position des objets dans le repère de navigation et d'améliorer l'estimation de la trajectoire du sous-marin d'autre part (principe du SLAM).

Ces 3 étapes doivent en général être réexécutées plusieurs fois (en utilisant le menu *Computations\ Contract f/b/m/b/f* par exemple, qui exécute la séquence propagation→retro-propagation→mise en cohérence avec détections→retro-propagation→propagation) pour atteindre la meilleure précision (on s'arrête lorsqu'on voit que l'estimation de l'erreur et de la position ne change plus).

Une fois la trajectoire reconstituée avec la meilleure précision qu'on pouvait, GESMI peut indiquer à l'utilisateur les éventuelles détections d'objets sur l'image sonar qu'il aurait oubliées (menu *Computations*

Contract waterfall). En effet, connaissant la trajectoire du sous-marin et au moins une détection sur l'image sonar d'un objet, le logiciel peut indiquer les autres moments où le sous-marin a pu repasser devant l'objet et le voir sur le sonar. Ceci est affiché sous forme de reconstitution de l'image sonar qui affiche les enveloppes de la distance objet-sous-marin, superposées aux données sonar réelles.

Utilisation de GESMI

- **Préparation des données.**
  - **Obtention des données de navigation**
    Lors de l'expérience les données de navigation du sous-marin indiquées par ses capteurs (GPS, centrale inertielle, capteur de pression, DVL...) doivent être enregistrées, avec leurs erreurs associées. La documentation des capteurs peut aider à trouver une estimation de leurs erreurs. Dans le cas de la *Daurade*, le filtre de Kalman embarqué donne déjà toutes ces informations. Des changements de repère ou conversion d'unités sont seulement nécessaires pour les adapter à ce qu'attend GESMI.
  - **Utilisation d'un logiciel dédié pour effectuer les détections sur l'image sonar**
    Un premier parcours des images sonar doit être fait par un opérateur humain pour détecter des objets particuliers (rochers, épaves...) devant lesquels le sous-marin serait repassé plusieurs fois. Ceci peut être fait à l'aide d'un logiciel tel que SonarPro (voir figure 4.9 et [L-3 Klein Associates, Inc, 2011]).
  - **Conversion des données sonar brutes en données utilisables pour** GESMI
    GESMI permet aussi d'afficher en partie les données sonar réelles dans sa fenêtre principale. Ces données doivent juste être converties pour qu'elles soient synchronisées avec les données de navigation. Il faut éventuellement envisager de réduire le volume de données (en prendre 1 sur 2 par exemple) si leur volume en mémoire devient ingérable par l'ordinateur.
- **Vérification de la cohérence entre les données de navigation.**
  Des incohérences entre les données de navigation au cours du temps peuvent être détectées avec une propagation (menu *Computations\Contract forward*) ou une retro-propagation (menu *Computations\Contract backward*). Par exemple, une erreur de changement de repère pour la vitesse de la *Daurade* a pu être détectée de cette façon. De même, l'erreur sur certaines données avait été sous-évaluée. Les données de navigation peuvent être considérées comme correctes si aucune incohérence n'est signalée par le programme au bout de plusieurs propagations et retro-propagations. On peut s'arrêter de faire des propagations et retro-propagations lorsqu'on voit que l'estimation de l'erreur sur la trajectoire (fenêtre *Position error*) ne varie plus.
- **Vérification de la cohérence entre les données de navigation et les détections faites par l'utilisateur pour chaque objet.**
  La prise en compte de détections d'objets devant lesquels le sous-marin est passé plusieurs fois peut améliorer l'estimation de la trajectoire du sous-marin. Pour chaque objet, il faut prendre une première détection et générer la reconstitution de l'image sonar (menu *Computations\Contract waterfall*) pour voir les endroits sur l'image sonar où l'objet aurait pu être revu. Il faut ensuite les comparer avec les éventuelles autres détections faites par l'opérateur humain et vérifier si celles-ci paraissent cohérentes. Des détections un peu trop approximatives peuvent souvent causer des incohérences. On peut alors recommencer les calculs avec toutes les détections qui paraissent convenables ou qu'on a corrigées. Si aucune incohérence n'est signalée par le programme après plusieurs propagations, retro-propagations et mises en cohérence avec les détections (menu *Computations\Contract f/b/m/b/f*), on peut passer à l'objet suivant et faire de même, puis prendre en compte toutes les détections pour tous les objets à la fin. On aura alors la meilleure

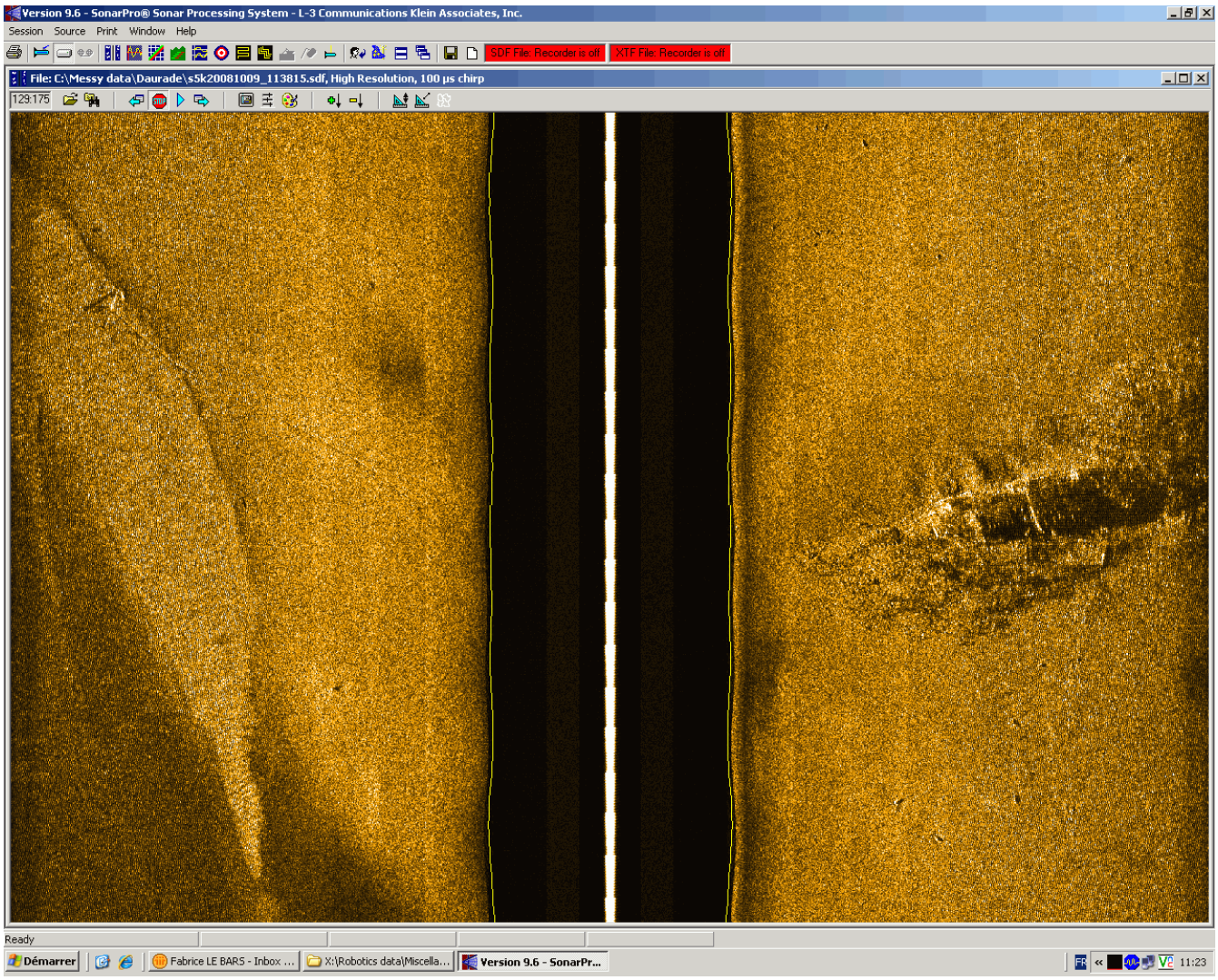


Figure 4.9 – Capture d'écran d'une image sonar (bâbord et tribord) affichée par SonarPro. L'opérateur humain doit faire défiler l'image sonar de bas en haut pour parcourir toutes les données dans le temps. On voit à droite (côté tribord) une forme correspondant à une épave.

estimation possible des positions du sous-marin et des objets dans le repère de référence/navigation.

Résultats pour l'expérience avec la *Daurade* A partir des données de navigation et des 3 positions GPS mesurées au début, au milieu et à la fin de la mission, on peut obtenir une enveloppe de la trajectoire avec une propagation *forward* sur l'équation d'évolution. La figure 4.10 a) représente l'enveloppe estimée avec le centre des pavés correspondants. A cause de la nature de la mission du robot (quadrillage), les pavés se superposent et il est difficile de voir la précision de l'enveloppe sans la figure 4.10 b), qui représente les erreurs (moitié de la largeur des pavés contenant les positions estimées) sur les positions x, y et z par rapport à t.

Une propagation *backward* améliore l'estimation de position et produit les résultats des figures 4.10 c) et 4.10 d). On voit que les erreurs en position en x et y ont une sorte de seuil limite. Ceci est dû à la différence entre les positions estimées et les positions mesurées au niveau des points GPS. Si la position estimée avait été égale à la position mesurée, il n'y aurait pas eu de contraction de la trajectoire et on aurait eu une forme de courbe triangulaire, sans seuil.

Plus de mesures et plus d'informations sur l'expérience impliquent une meilleure précision des résultats. Ainsi, si on prend en compte les détections faites par l'opérateur humain sur l'image sonar, l'estimation de la trajectoire du robot peut être améliorée avec l'estimation des postions des amers (voir figures 4.10 e) et 4.10 f)). Après une propagation globale en prenant en compte les détections des amers faites sur l'image sonar et toutes les données des autres capteurs, on obtient une localisation de l'AUV et des amers avec une erreur de moins de 40 m en moins de 10 s de temps de calcul. L'amélioration de la précision des positions trouvées par le SLAM par intervalles comparée à celles obtenues par le filtre de Kalman embarqué dans la *Daurade* (simple localisation sans SLAM) est faible (l'amélioration est de moins de 10 m). Cette faible amélioration est due au fait que (i) les données fournies par l'INS et le DVL sont très précises (ii) un point GPS a été pris au milieu de la mission, ce qui rend le SLAM moins utile et (iii) le filtre de Kalman suppose que les bruits d'état et de mesure sont blancs, gaussiens et centrés ce qui permet d'avoir une erreur augmentant avec $\sqrt{t}$ (ce qui est peut-être optimiste en pratique). A l'inverse, l'erreur calculée par la méthode intervalle considère le pire cas et suppose seulement qu'on a des bornes sur les erreurs de mesure. Comme aucune linéarisation n'a été faite et qu'aucune hypothèse sur la nature du bruit n'est faite, les erreurs fournies par l'approche par intervalles peuvent être considérées comme sûres. Dans le contexte de cette expérience aucune incohérence n'a été détectée (cependant GESMI a permis de détecter des erreurs faites au début dans les conversions des données et certaines fausses détections, qui ont été rapidement corrigées par la suite), *i.e.* toutes les données et erreurs correspondantes fournies par le système de navigation du robot sont cohérentes avec les détections d'amers de l'opérateur humain et les enveloppes calculées par la propagation intervalles. En conséquence, ce type de propagation peut être utilisé pour valider un système de navigation (et même être intégré au système de navigation vu que les temps de calculs sont faibles). Même si une validation rapide est déjà possible en comparant les positions réelles obtenues par GPS où le sous-marin a fait surface et les positions prévues (positions où il aurait dû faire surface), avec la propagation de contraintes intervalles, plus de choses sont vérifiées : les détections d'amers faites pas l'opérateur humain, les données du système de navigation, les points GPS, les pavés représentant l'ensemble de la trajectoire, les pavés représentant les positions des amers dans l'eau... sont tous cohérents entre eux.

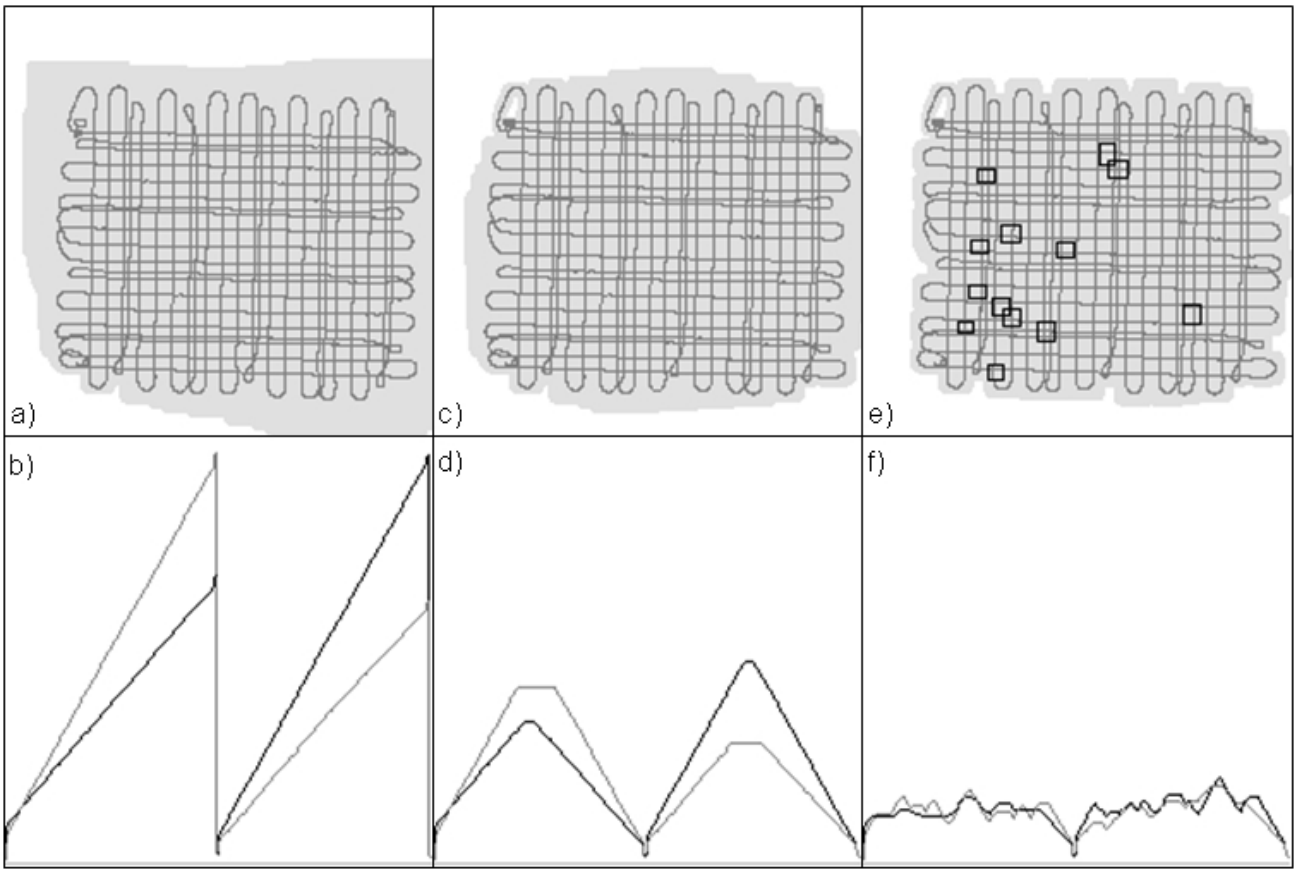


Figure 4.10 – a) Enveloppe de la trajectoire après une propagation *forward* (temps de calcul inférieur à 1 s sur un ordinateur à base de processeur Intel Core 2 Duo), dessinée en gris. Le centre des pavés correspondant à la position du sous-marin à chaque instant est dessiné en gris foncé. Le repère est $[-200, 1000]$ m $\times$ $[-200, 1000]$ m (dans le plan (x, y)). b) Erreurs en position en x (noir), y (gris foncé), z (gris clair) en fonction du temps après une propagation *forward*. Le repère est $[0, 18445.75]$ s $\times$ $[0, 200]$ m. c) Enveloppe de la trajectoire après une propagation *forward-backward* (temps de calcul inférieur à 2 s). d) Erreurs en position après une propagation *forward-backward*. e) Enveloppe de la trajectoire après 10 contractions, en prenant en compte les amers détectés sur l'image sonar (temps de calcul inférieur à 10 s). Les pavés contenant les positions des amers sont dessinés en noir. f) Erreurs en position après 10 contractions, en prenant en compte les amers détectés sur l'image sonar.

Aide à l'identification des amers Bien que les détections d'amers sur l'image sonar et la correspondance entre les amers détectés plusieurs fois aient été faites manuellement par un opérateur humain à la fin de la mission du robot pour les expériences avec la *Daurade* et le *Redermor*, GESMI peut aider l'opérateur humain à corriger ses erreurs de correspondance (voir figure 4.11). GESMI permet de voir les données sonar, ajouter une détection, voir les enveloppes contenant la trajectoire du robot et la position de l'amer ainsi que de dessiner schématiquement les endroits sur l'ensemble de l'image sonar où cet amer aurait dû avoir été revu et de confirmer ces prédictions en ajoutant des nouvelles détections de l'amer. En conséquence, l'opérateur humain ne devrait avoir qu'à détecter une seule fois manuellement chaque amer, les instants et les distances robot-amer des autres éventuelles détections d'un amer donné devraient être prédites grâce aux données de navigation et il n'aurait alors qu'à confirmer ces détections supplémentaires indiquées par GESMI.

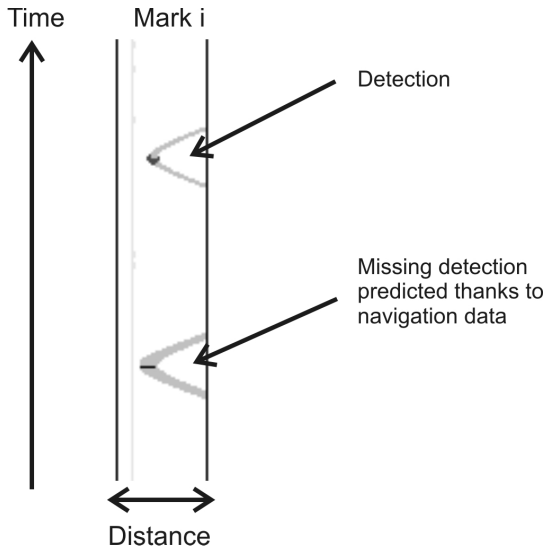


Figure 4.11 – Image sonar reconstituée schématiquement montrant un amer détecté et la prédiction de la distance robot-amer à un autre instant obtenue grâce à la prise en compte des données de navigation du sous-marin.

Prenons par exemple les données du *Redermor* et supposons que nous avons chargé avec GESMI les données de navigation et les données sonar, mais qu'on n'ait pas encore cherché d'amers. Pour cette expérience, le *Redermor* n'a pris que 2 points GPS, un au début ($\mathbf{p}(0) = (x(0), y(0), z(0))$ avec $x(0) = 0.06, y(0) = 0, z(0) = 0.76$) et l'autre à la fin ($\mathbf{p}(5999.40) = (x(5999.40), y(5999.40), z(5999.40))$ avec $x(5999.40) = 689.96, y(5999.40) = 264.10, z(5999.40) = 0.752$). En lançant une propagation *forward* puis une propagation *backward* (seule l'équation d'évolution du sous-marin est donc prise en compte), on obtient les estimations de trajectoire et d'erreur présentées dans les figures 4.12 et 4.13.

En parcourant l'image sonar tribord avec GESMI, on distingue un objet qui correspond en fait à une mine. On peut donc la prendre comme amer ($i = 0$) et indiquer à GESMI qu'on l'a détecté 1 fois sur l'image sonar tribord ($z_0 = (d_{m0}, \alpha_{m0}, z_{m0})$ avec $d_{m0} = 37.13, \alpha_{m0} = \pi/2, z_{m0} = 9.81$,voir figure 4.14).

Si maintenant on prend en compte l'équation des amers, on obtient la position de la mine dans le repère de référence (figure 4.15).

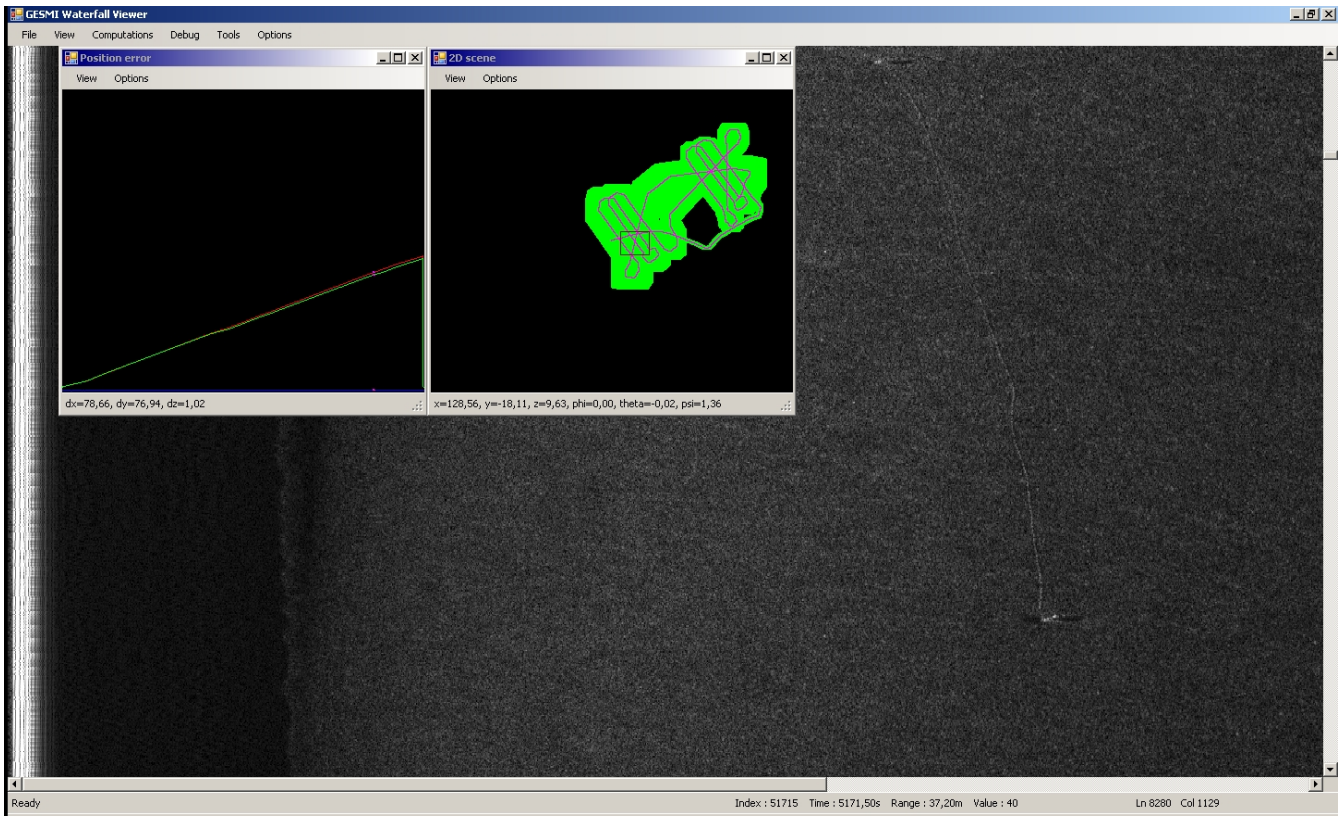


Figure 4.12 – Résultats d'une propagation *forward* sur les données de navigation du *Redermor* avec GESMI. Dans la fenêtre *Position error* (voir figure 4.7 pour plus de détails sur ce que représente cette fenêtre), on constate que l'erreur d'estimation de position augmente régulièrement en fonction du temps. En effet, seul le point GPS initial aide à l'estimation de trajectoire dans le cas d'une propagation *forward* sur l'équation d'évolution, les erreurs d'estimation ne peuvent que s'accumuler au cours du temps. L'enveloppe de la trajectoire correspondante est dessinée dans la fenêtre *2D scene* (voir la figure 4.5 pour plus de détails sur ce que représente cette fenêtre). L'enveloppe devrait être la plus fine près du point de départ, qui se situe au centre de la fenêtre, mais il est difficile de le distinguer à cause de la forme de la trajectoire qui provoque des superpositions, d'où l'intérêt de la fenêtre *Position error*. Une partie de l'image sonar tribord est dessinée dans la fenêtre principale (voir la figure 4.8 pour plus de détails sur ce que représente cette fenêtre), on y voit notamment deux mines reliées par un câble, qui vont être utilisées dans la suite.

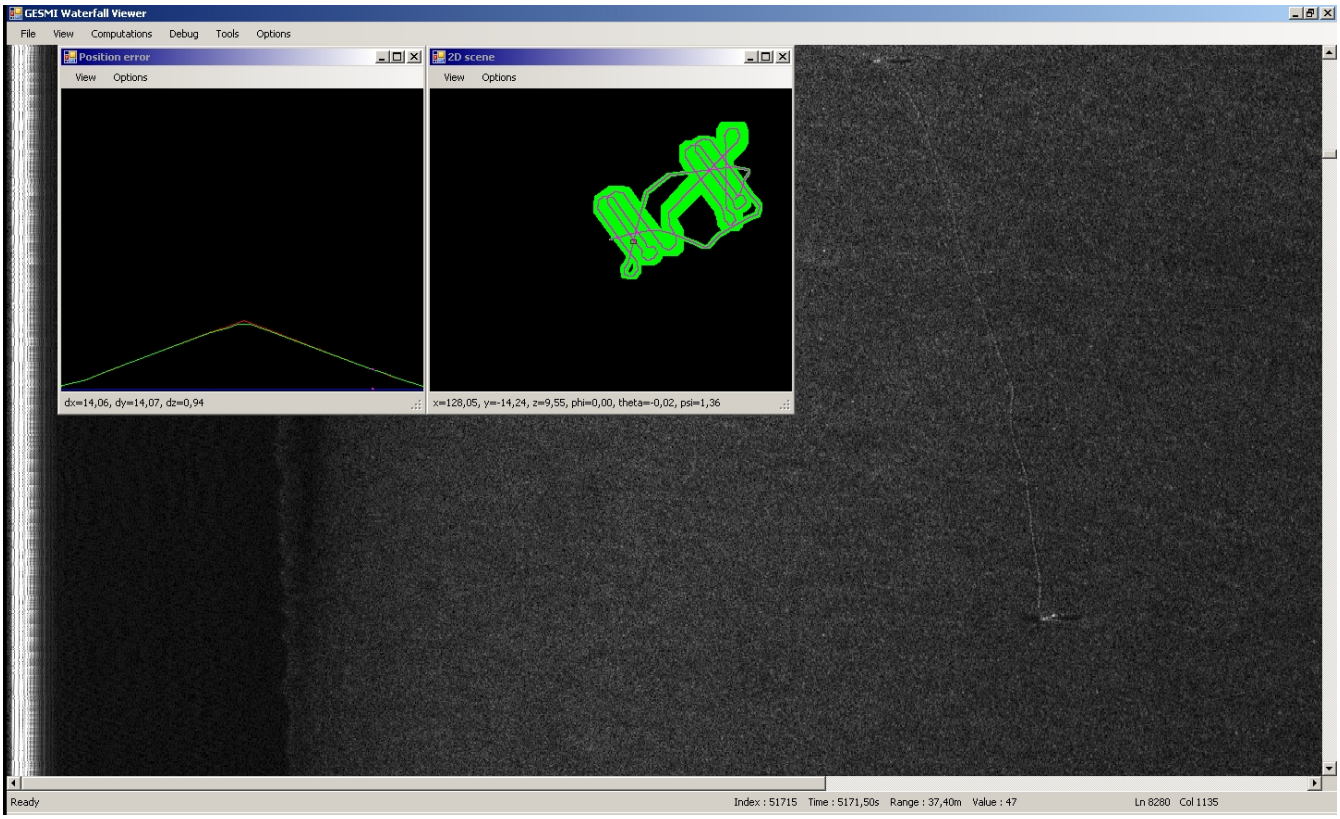


Figure 4.13 – Résultats d'une propagation *forward-backward* sur les données de navigation du *Redermor* avec GESMI. Dans la fenêtre *Position error* (voir figure 4.7 pour plus de détails sur ce que représente cette fenêtre), on constate que l'erreur d'estimation de position est maintenant minimale au début et à la fin de la mission. C'est grâce au point GPS final, qui a maintenant pu être pris en compte grâce à la propagation *backward* sur l'équation d'évolution. L'enveloppe de la trajectoire correspondante dessinée dans la fenêtre *2D scene* (voir la figure 4.5 pour plus de détails sur ce que représente cette fenêtre) est maintenant globalement plus fine qu'avec la propagation *forward* seule.

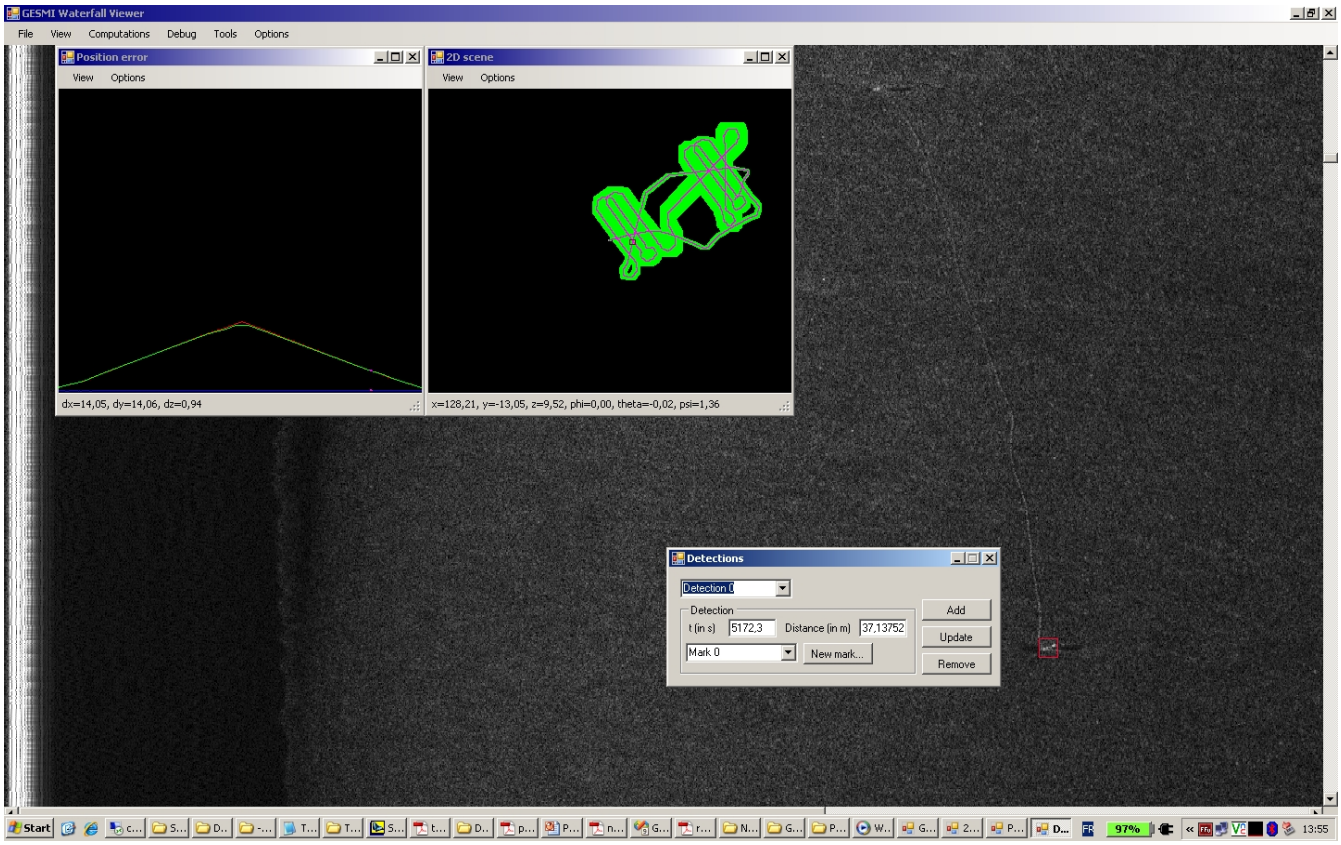


Figure 4.14 – Ajout d'une détection de mine sur une image sonar du *Redermor* avec GESMI.

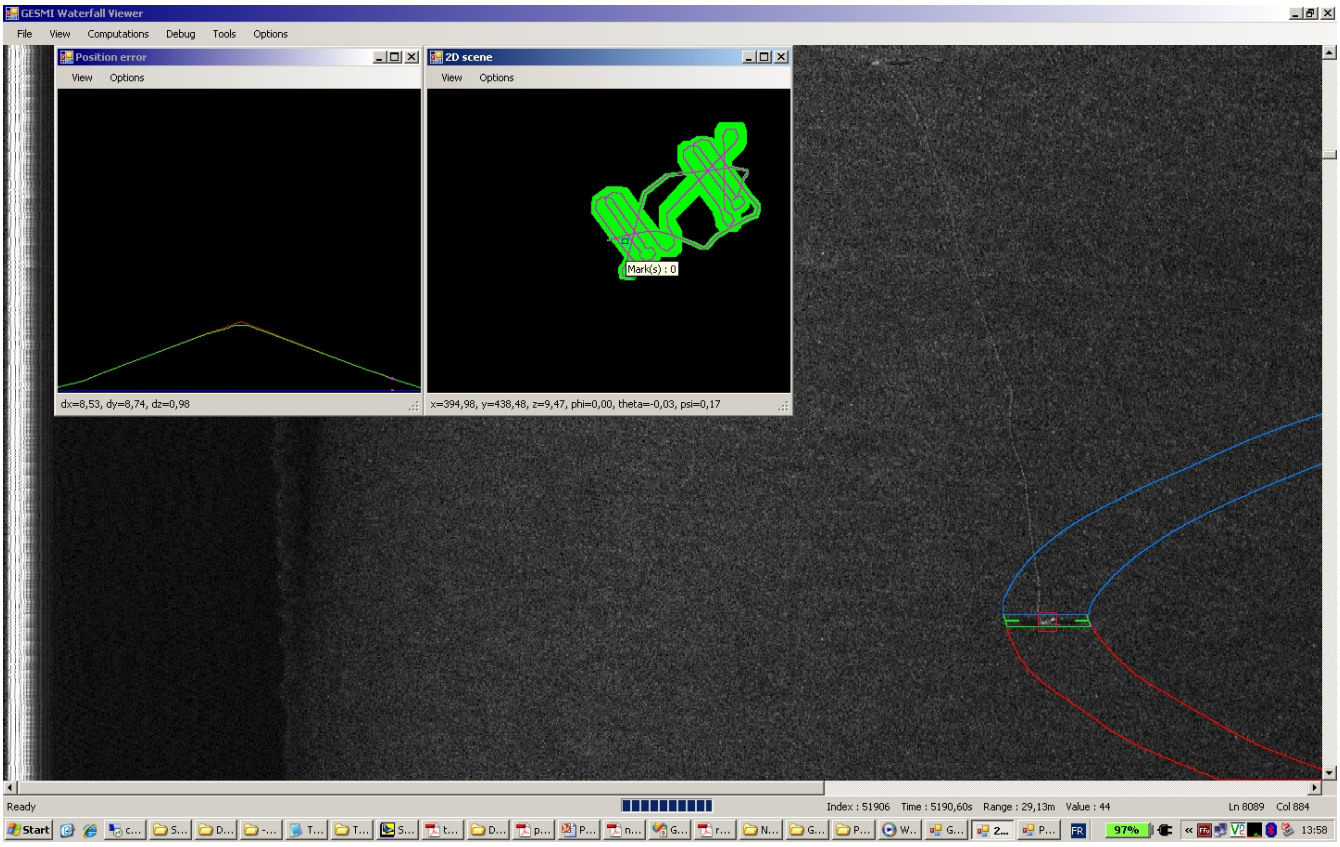


Figure 4.15 – Résultats d'une propagation forward/backward/mark sur les données de navigation du *Redermor* avec GESMI. Vu qu'on n'a qu'une seule détection d'un amer pour l'instant, l'estimation de la trajectoire du sous-marin reste inchangée après une nouvelle propagation *forward-backward* sur l'équation d'évolution cependant, la prise en compte de l'équation des amers permet d'obtenir la position de la mine dans l'eau et de la dessiner (carré bleu) dans la fenêtre *2D scene* (voir la figure 4.5 pour plus de détails sur ce que représente cette fenêtre).

Il est alors possible de calculer la distance robot-amer $\|\mathbf{e}_0(t)\|$ (avec $\mathbf{e}_0(t) = \mathbf{p}(t) - \mathbf{m}_0$) pour tout t et de la dessiner sur l'image sonar. En calculant en plus $\arg\left((\mathbf{e}_{ri}(t).\mathbf{i}_r(t), \mathbf{e}_{ri}(t).\mathbf{j}_r(t))\right)$ (avec $\mathbf{e}_{ri}(t) = \mathbf{R}^{\mathrm{T}}(\varphi(t), \theta(t), \psi(t)).\mathbf{e}_i(t)$ et $(\mathbf{p}(t), \mathbf{i}_r(t), \mathbf{j}_r(t), \mathbf{k}_r(t))$ le repère associé au robot), on peut déterminer les instants où le sonar du sous-marin devrait voir l'amer. Ainsi, en continuant le parcours de l'image sonar avec GESMI, on constate que le sous-marin est repassé en face de l'amer 0 à un autre moment (figure 4.16).

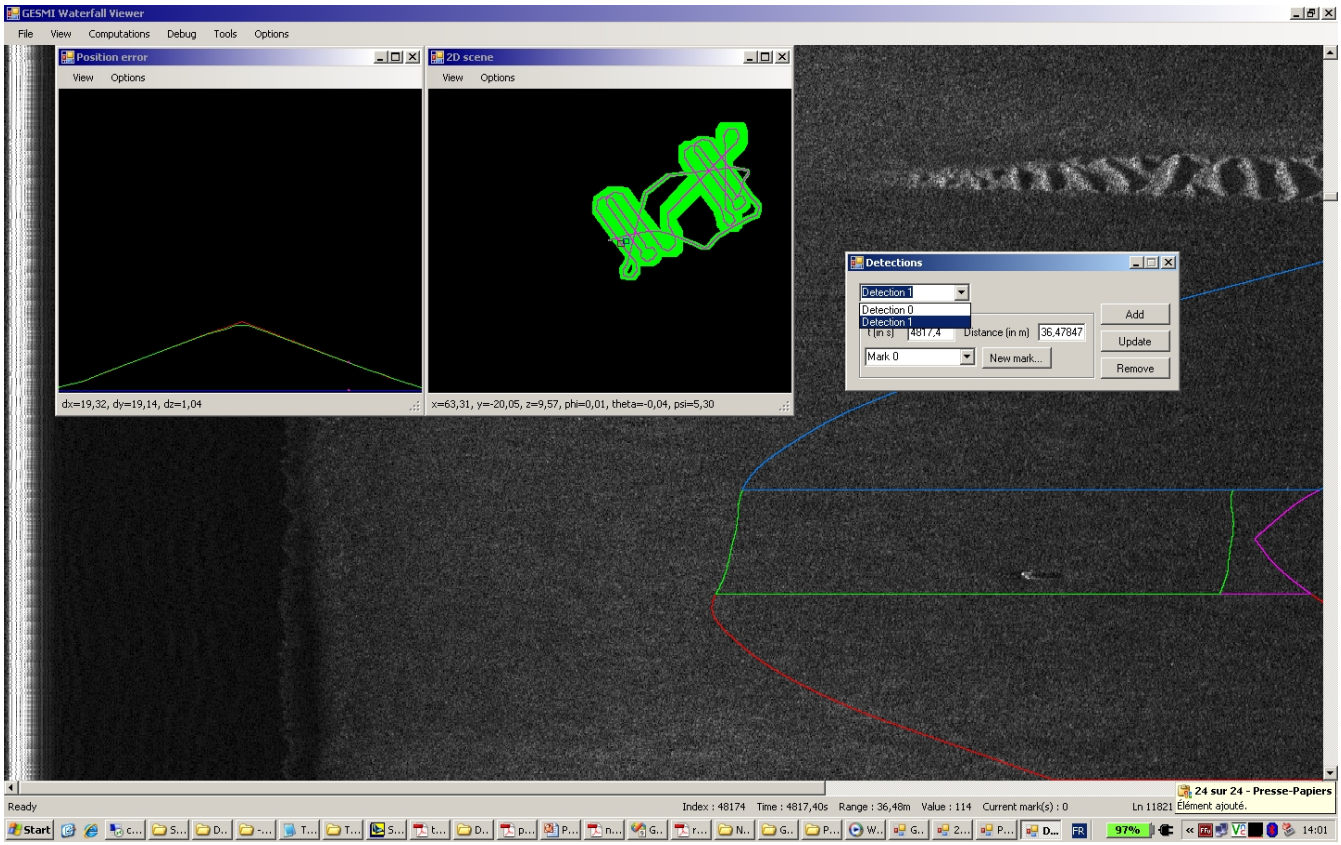


Figure 4.16 – Ajout d'une deuxième détection de la mine détectée manuellement auparavant, grâce à l'indication modélisée par une enveloppe verte dans la fenêtre principale de GESMI.

On peut alors rajouter une deuxième détection de cet amer et repropager les résultats. On voit alors clairement en regardant l'estimation d'erreur en position que celle-ci est plus précise aux endroits où le sous-marin est passé à proximité de l'amer (voir figure 4.17).

GESMI a donc permis de faire un SLAM offline et a en plus aidé (en partie) à résoudre le problème d'association des données, précédemment entièrement géré par l'opérateur humain lors de son analyse des données sonar.

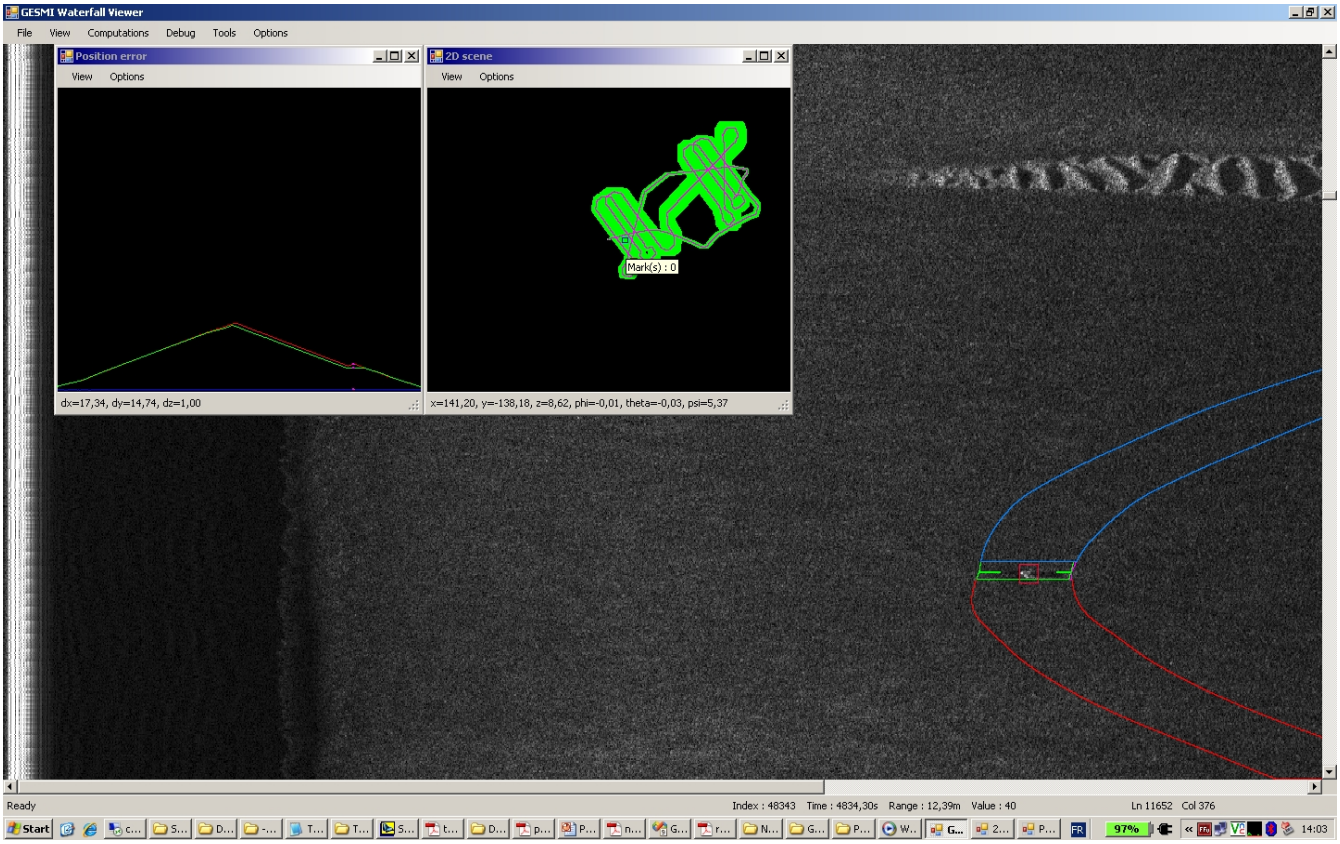


Figure 4.17 – Amélioration de l'estimation de position du sous-marin grâce à 2 détections du même amer. Dans la fenêtre *Position error* (voir figure 4.7 pour plus de détails sur ce que représente cette fenêtre), on constate que l'erreur d'estimation de position a une sorte de seuil à un moment donné, qui doit correspondre au passage en face de l'amer détecté. L'enveloppe de la distance robot-amer est maintenant plus étroite au niveau de la deuxième détection d'amer dans la fenêtre principale (voir la figure 4.8 pour plus de détails sur ce que représente cette fenêtre) car cette détection a été rajoutée dans la liste des détections connues précisément à l'étape précédente (le fait qu'elle soit maintenant connue est indiqué par le petit carré rouge).

4.4 Validation de différentes méthodes de résolution du problème de SLAM

Dans cette section, nous allons considérer un problème de SLAM offline 2D/3D bearing-only (appliqué à une simulation d'expérience avec un robot-char muni d'une caméra omnidirectionnelle) sans données aberrantes, en supposant que l'on connait l'association des données de détections, avec des amers ponctuels et toujours visibles. Les données utilisées sont celles présentées dans [Joly, 2010] (collaboration avec Cyril JOLY), où une comparaison entre méthodes probabilistes et intervalles a déjà été faite (voir aussi [Joly and Rives, 2008]). Le but de cette section est de compléter cette comparaison pour essayer d'avoir une idée de l'optimalité, la précision de la méthode de SLAM intervalles présentée précédemment. Pour cela, il est possible de rajouter des équations (interdétections) ou astuces (orientation des axes) propres au problème traité pour voir si les enveloppes de la trajectoire et des positions des amers se réduisent ou tester d'autres algorithmes (bissections, qui marchent pour tout type de problème mais au prix d'un temps de calcul parfois rédhibitoire). Pour obtenir un maximum de possibilités de comparaison, les données traitées dans [Joly, 2010] ont été directement reprises. Les conditions de l'expérience simulée étaient les suivantes : un robot char muni d'odomètres (pour obtenir les vitesses tangentielle et de rotation) et d'une caméra omnidirectionnelle se déplace dans un environnement plan où se trouvent $n = 200$ amers ponctuels (placés dans un espace 3D, avec une coordonnée d'altitude), dont un traitement d'images fait sur les données de la caméra permet de mesurer l'angle de gisement α et d'élévation β.

Notons $\mathbf{x} = (x, y, \theta)$ l'état du robot avec (x, y) ses coordonnées de position dans le repère de référence $(\mathbf{O}, \mathbf{i}, \mathbf{j})$, $\mathbf{u} = (V_x, V_y, \omega)$ l'entrée du système avec V_x sa vitesse tangentielle, V_y sa vitesse latérale et ω sa vitesse de rotation, $\mathbf{m}_i = (x_{mi}, y_{mi}, z_{mi})$ la position de l'amer i et $\mathbf{z}_i = (\alpha_{mi}, \beta_{mi})$ les mesures d'angle des amers par rapport au robot (voir les figures 2.3 pour les notations et 4.18 pour la trajectoire réelle du robot et les positions réelles des amers en 2D).

L'équation d'évolution du robot peut être écrite sous cette forme (équation d'un char susceptible de glisser latéralement) :

$$\begin{cases} \dot{x}(t) &= V_x(t)\cos\theta(t) - V_y(t)\sin\theta(t) + b_x(t) \\ \dot{y}(t) &= V_x(t)\sin\theta(t) + V_y(t)\cos\theta(t) + b_y(t) \\ \dot{\theta}(t) &= \omega(t) + b_\theta(t). \end{cases} \tag{4.17}$$

L'équation d'observation est :

$$\mathbf{y}(t) = \mathbf{x}(t) \tag{4.18}$$

car on mesure l'état initial du robot (*i.e.* on connaît l'état seulement à $t = 0$). L'équation des amers peut être écrite de cette façon :

$$\begin{cases} \alpha_{mi}(t) = \arg\left(x_{mi} - x(t), y_{mi} - y(t)\right) - \theta(t) \\ \beta_{mi}(t) = \arg\left(\sqrt{(x_{mi} - x(t))^2 + (y_{mi} - y(t))^2}, z_{mi}\right). \end{cases} \tag{4.19}$$

Ces équations sont de la même forme que les équations 4.4. De cette manière, les mêmes méthodes de résolution pourront être utilisées.

De plus, nous utiliserons 2 types d'équations d'interdétections spécifiques au problème traité :

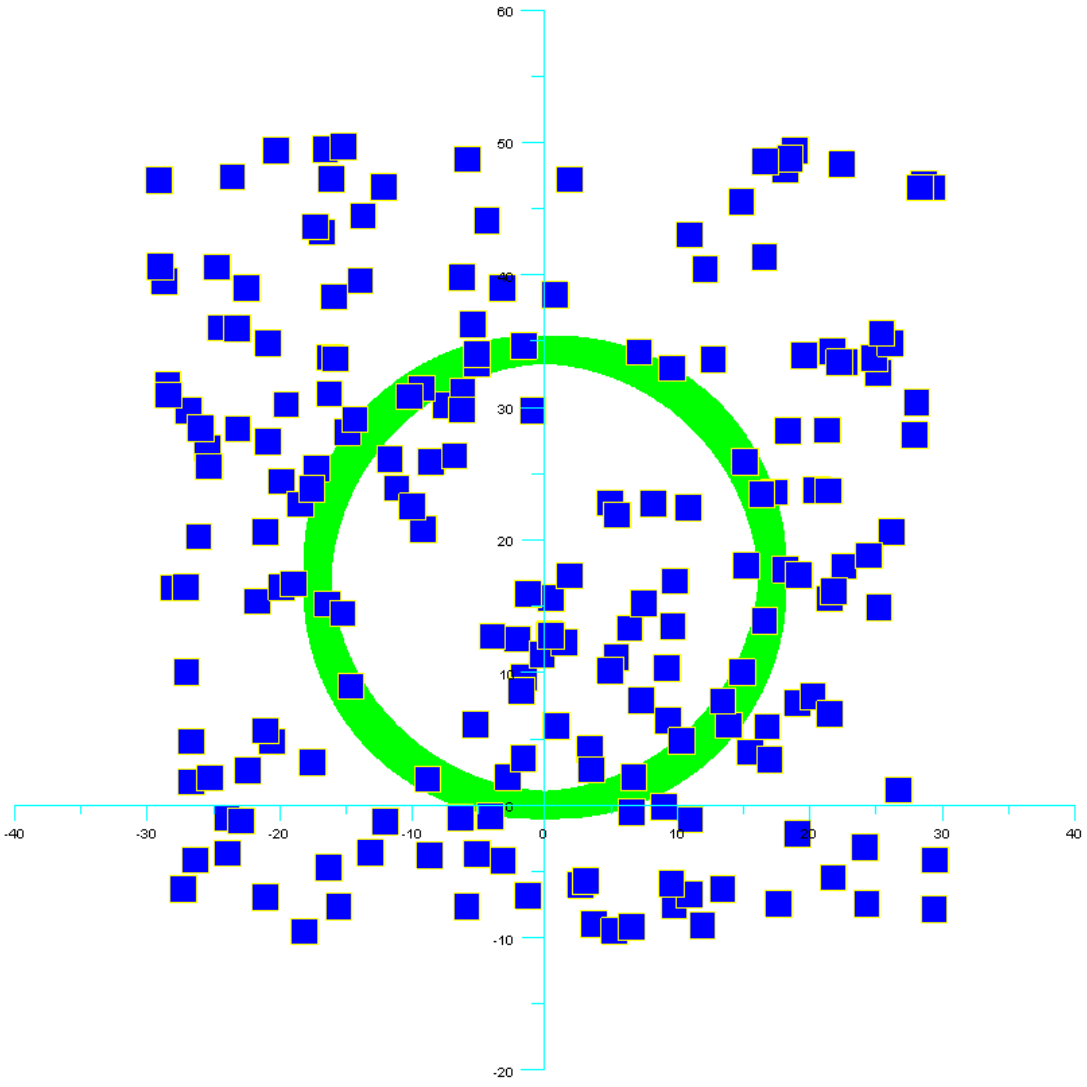


Figure 4.18 – Trajectoire réelle du robot en vert et positions réelles des amers en bleu, en 2D.

– Equations inter-amers : lorsque 2 amers i_1 et i_2 sont détectés à un instant t (voir figure 4.19) :

$$\begin{cases} x\left(t\right) = \dfrac{y_{mi_1} - y_{mi_2} + x_{mi_2}\tan\left(\alpha_{mi_2}(t)+\theta(t)\right) - x_{mi_1}\tan\left(\alpha_{mi_1}(t)+\theta(t)\right)}{\tan\left(\alpha_{mi_2}(t)+\theta(t)\right) - \tan\left(\alpha_{mi_1}(t)+\theta(t)\right)} \\ y\left(t\right) = \dfrac{x_{mi_1} - x_{mi_2} + \dfrac{y_{mi_2}}{\tan\left(\alpha_{mi_2}(t)+\theta(t)\right)} - \dfrac{y_{mi_1}}{\tan\left(\alpha_{mi_1}(t)+\theta(t)\right)}}{\dfrac{1}{\tan\left(\alpha_{mi_2}(t)+\theta(t)\right)} - \dfrac{1}{\tan\left(\alpha_{mi_1}(t)+\theta(t)\right)}}. \end{cases} \tag{4.20}$$

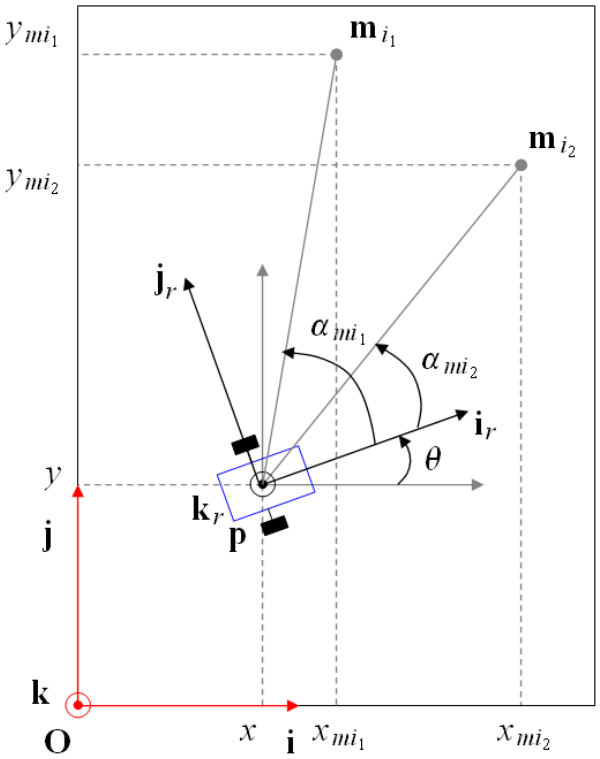


Figure 4.19 – Robot à un instant t voyant 2 amers i_1 et i_2. Cette situation permet d'écrire des équations inter-amers.

– Equations intertemporelles : lorsqu'un amer i est détecté à 2 instants t_1 et t_2 (voir figure 4.20) :

$$\begin{cases} x_{mi} = \dfrac{y(t_1) - y(t_2) + x(t_2)\tan(\alpha_{mi}(t_2)+\theta(t_2)) - x(t_1)\tan(\alpha_{mi}(t_1)+\theta(t_1))}{\tan(\alpha_{mi}(t_2)+\theta(t_2)) - \tan(\alpha_{mi}(t_1)+\theta(t_1))} \\ y_{mi} = \dfrac{x(t_1) - x(t_2) + \dfrac{y(t_2)}{\tan(\alpha_{mi}(t_2)+\theta(t_2))} - \dfrac{y(t_1)}{\tan(\alpha_{mi}(t_1)+\theta(t_1))}}{\dfrac{1}{\tan(\alpha_{mi}(t_2)+\theta(t_2))} - \dfrac{1}{\tan(\alpha_{mi}(t_1)+\theta(t_1))}}. \end{cases} \tag{4.21}$$

Au final, les algorithmes de résolution qui pourront être comparés seront les suivants (il est à noter que les algorithmes 5 et 6 ne seront pas étudiés ici, cependant leurs résultats sur le même jeu de données sont disponibles dans [Joly, 2010]) :

1. La propagation de contraintes par intervalles décrite en 4.1 et appliquée sur les données des sous-marins *Daurade* et *Redermor* (le but est justement d'avoir une idée de son optimalité).
2. L'algorithme précédent avec les équations d'interdétections propres au problème (traitées avec le contracteur HC4Revise, de la même façon que les équations d'observation et de détections d'amers).

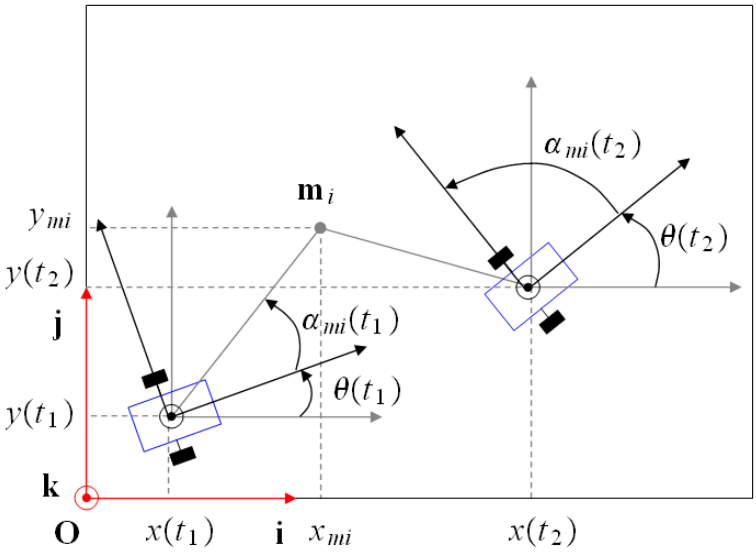


Figure 4.20 – Robot voyant le même amer i à 2 instants t_1 et t_2. Cette situation permet d'écrire des équations intertemporelles.

3. L'algorithme précédent avec utilisation de l'algorithme 3BCID (présent dans IBEX), qui découpe les pavés retournés par chaque contracteur en petites tranches et cherche à éliminer celles qui sont inconsistantes (voir [Trombettoni and Chabert, 2007b] pour plus de détails sur 3BCID).
4. L'algorithme 1 (propagation de contraintes intervalles sans équations d'interdétections) avec utilisation de l'algorithme STRANGLE, qui fait des bissections complètes sur les pavés représentant les positions des amers (voir [Jaulin, 2002a] pour plus détails sur STRANGLE). C'est cet algorithme qui devrait permettre d'obtenir les sous-pavages les plus petits contenant les amers, au prix d'un temps de calcul important.
5. Le SAM (Smoothing And Mapping, une méthode probabiliste) décrit dans [Joly, 2010].
6. L'algorithme intervalles avec optimisation de l'orientation des axes (propre au problème) décrit dans [Joly, 2010]. En effet, la taille des pavés calculés peut changer selon l'orientation du repère utilisé. Une façon rapide pour faire cela peut être de changer l'orientation initiale du robot si aucune supposition n'est faite sur les positions des amers (cependant la méthode utilisée dans [Joly, 2010] est plus élaborée).

Les scénarios de simulations utilisés sont les mêmes que dans [Joly, 2010]. Pour toutes les simulations, on a $V_x = 1.5\,\mathrm{m.s^{-1}}$, $V_y = 0\,\mathrm{m.s^{-1}}$ (pas de glissement latéral du robot), $\omega = \frac{5\pi}{180}\,\mathrm{rad.s^{-1}}$, $b_x = 0$, $b_y = 0$, $b_\theta = 0$ (pas de bruit additif). Le robot fait 2.1 tours d'un cercle de rayon 17.2 m, dans une mission de 150 s. Pour les calculs, on connait parfaitement l'état initial du robot $\mathbf{x}(0) = (x(0), y(0), \theta(0)) = (0, 0, 0)$ ainsi que les entrées $\mathbf{u} = (V_x, V_y, \omega)$ et les mesures d'angles $\mathbf{z}_i = (\alpha_{mi}, \beta_{mi})$ par rapport aux amers i pour tout t et on suppose que $\forall t\ [b_x(t)] = [b_y(t)] = [-0.001, 0.001]$ et $[V_y(t)] = 0.01\,([V_x(t)] - \mathrm{mid}([V_x(t)]))$. Différentes formes (gaussienne centrée, uniforme centrée et uniforme biaisée) et bornes pour les erreurs des mesures $\tilde{V}_x$, $\tilde{\omega}$, $\tilde{\alpha}_{mi}$, $\tilde{\beta}_{mi}$ de V_x, ω, α_{mi}, β_{mi} sont considérées :

– Erreurs des mesures de V_x, ω, α_{mi}, β_{mi} gaussiennes centrées. Comme par définition ces erreurs ne peuvent pas être bornées, les algorithmes de résolution 1, 2, 3, 4 sont lancés en considérant des erreurs contenues dans des intervalles de longueur $2k\sigma$ avec $k \in \{3, 3.1, 3.2, 3.3, ..., 4\}$, les valeurs de k les plus faibles donnant

souvent des résultats inconsistants (une incohérence due à une erreur en dehors des bornes supposées génère un ensemble vide pour une valeur au cours des calculs). 4 scénarios de simulation ont été générés :
  - $\sigma_{V_x} = 0.1\,\mathrm{m.\,s^{-1}}$, $\sigma_\omega = 0.1\,\mathrm{rad.\,s^{-1}}$, $\sigma_{\alpha_{mi}} = \frac{\pi}{180}\,\mathrm{rad}$, $\sigma_{\beta_{mi}} = \frac{\pi}{180}\,\mathrm{rad}$ (scénario 1).
  - $\sigma_{V_x} = 0.1\,\mathrm{m.\,s^{-1}}$, $\sigma_\omega = 0.1\,\mathrm{rad.\,s^{-1}}$, $\sigma_{\alpha_{mi}} = \frac{0.1\pi}{180}\,\mathrm{rad}$, $\sigma_{\beta_{mi}} = \frac{0.1\pi}{180}\,\mathrm{rad}$ (scénario 2).
  - $\sigma_{V_x} = 0.025\,\mathrm{m.\,s^{-1}}$, $\sigma_\omega = 0.005\,\mathrm{rad.\,s^{-1}}$, $\sigma_{\alpha_{mi}} = \frac{3\pi}{180}\,\mathrm{rad}$, $\sigma_{\beta_{mi}} = \frac{3\pi}{180}\,\mathrm{rad}$ (scénario 3).
  - $\sigma_{V_x} = 0.05\,\mathrm{m.\,s^{-1}}$, $\sigma_\omega = 0.01\,\mathrm{rad.\,s^{-1}}$, $\sigma_{\alpha_{mi}} = \frac{\pi}{180}\,\mathrm{rad}$, $\sigma_{\beta_{mi}} = \frac{\pi}{180}\,\mathrm{rad}$ (scénario 4).
- Erreurs des mesures de V_x, ω, α_{mi}, β_{mi} uniformes centrées dans un intervalle :
  - $V_x \in \left[\tilde{V}_x - 0.2, \tilde{V}_x + 0.2\right]\,\mathrm{m.\,s^{-1}}$, $\omega \in [\tilde{\omega} - 0.2, \tilde{\omega} + 0.2]\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} \in \left[\tilde{\alpha}_{mi} - \frac{\pi}{180}, \tilde{\alpha}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$, $\beta_{mi} \in \left[\tilde{\beta}_{mi} - \frac{\pi}{180}, \tilde{\beta}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$ (scénario 5).
  - $V_x \in \left[\tilde{V}_x - 0.2, \tilde{V}_x + 0.2\right]\,\mathrm{m.\,s^{-1}}$, $\omega \in [\tilde{\omega} - 0.2, \tilde{\omega} + 0.2]\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} \in \left[\tilde{\alpha}_{mi} - 0.1\frac{\pi}{180}, \tilde{\alpha}_{mi} + 0.1\frac{\pi}{180}\right]\,\mathrm{rad}$, $\beta_{mi} \in \left[\tilde{\beta}_{mi} - 0.1\frac{\pi}{180}, \tilde{\beta}_{mi} + 0.1\frac{\pi}{180}\right]\,\mathrm{rad}$ (scénario 6).
  - $V_x \in \left[\tilde{V}_x - 0.05, \tilde{V}_x + 0.05\right]\,\mathrm{m.\,s^{-1}}$, $\omega \in [\tilde{\omega} - 0.01, \tilde{\omega} + 0.01]\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} \in \left[\tilde{\alpha}_{mi} - 9\frac{\pi}{180}, \tilde{\alpha}_{mi} + 9\frac{\pi}{180}\right]\,\mathrm{rad}$, $\beta_{mi} \in \left[\tilde{\beta}_{mi} - 9\frac{\pi}{180}, \tilde{\beta}_{mi} + 9\frac{\pi}{180}\right]\,\mathrm{rad}$ (scénario 7).
  - $V_x \in \left[\tilde{V}_x - 0.05, \tilde{V}_x + 0.05\right]\,\mathrm{m.\,s^{-1}}$, $\omega \in [\tilde{\omega} - 0.05, \tilde{\omega} + 0.05]\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} \in \left[\tilde{\alpha}_{mi} - \frac{\pi}{180}, \tilde{\alpha}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$, $\beta_{mi} \in \left[\tilde{\beta}_{mi} - \frac{\pi}{180}, \tilde{\beta}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$ (scénario 8).
- Erreurs des mesures de V_x, ω, α_{mi}, β_{mi} uniformes et biaisées : les erreurs générées lors des simulations ne se trouvent que dans une partie de l'intervalle considéré par les algorithmes de résolution, d'où un biais. Ces algorithmes considèrent que $V_x \in \left[\tilde{V}_x - 0.1, \tilde{V}_x + 0.1\right]\,\mathrm{m.\,s^{-1}}$, $\omega \in [\tilde{\omega} - 0.02, \tilde{\omega} + 0.02]\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} \in \left[\tilde{\alpha}_{mi} - \frac{\pi}{180}, \tilde{\alpha}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$, $\beta_{mi} \in \left[\tilde{\beta}_{mi} - \frac{\pi}{180}, \tilde{\beta}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$ alors que les 4 scénarios de simulations générés ont des erreurs uniformes et centrées dans les intervalles suivants :
  - $V_x \in \left[\tilde{V}_x - 0.1, \tilde{V}_x\right]\,\mathrm{m.\,s^{-1}}$, $\omega \in [\tilde{\omega}, \tilde{\omega} + 0.05]\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} \in \left[\tilde{\alpha}_{mi} - \frac{\pi}{180}, \tilde{\alpha}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$, $\beta_{mi} \in \left[\tilde{\beta}_{mi} - \frac{\pi}{180}, \tilde{\beta}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$ (scénario 9).
  - $V_x \in \left[\tilde{V}_x - 0.1, \tilde{V}_x + 0.1\right]\,\mathrm{m.\,s^{-1}}$, $\omega \in [\tilde{\omega} - 0.05, \tilde{\omega} + 0.05]\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} \in \left[\tilde{\alpha}_{mi}, \tilde{\alpha}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$, $\beta_{mi} \in \left[\tilde{\beta}_{mi} - \frac{\pi}{180}, \tilde{\beta}_{mi}\right]\,\mathrm{rad}$ (scénario 10).
  - $V_x \in \left[\tilde{V}_x - 0.1, \tilde{V}_x\right]\,\mathrm{m.\,s^{-1}}$ et $\omega \in [\tilde{\omega}, \tilde{\omega} + 0.05]\,\mathrm{rad.\,s^{-1}}$ pendant la première moitié de la simulation puis $V_x \in \left[\tilde{V}_x, \tilde{V}_x + 0.1\right]\,\mathrm{m.\,s^{-1}}$ et $\omega \in [\tilde{\omega} - 0.05, \tilde{\omega}]\,\mathrm{rad.\,s^{-1}}$ ensuite, avec $\alpha_{mi} \in \left[\tilde{\alpha}_{mi}, \tilde{\alpha}_{mi} + \frac{\pi}{180}\right]\,\mathrm{rad}$ et $\beta_{mi} \in \left[\tilde{\beta}_{mi} - \frac{\pi}{180}, \tilde{\beta}_{mi}\right]\,\mathrm{rad}$ tout au long de la simulation (scénario 11).
  - $V_x = \tilde{V}_x - 0.1\,\mathrm{m.\,s^{-1}}$, $\omega = \tilde{\omega} + 0.05\,\mathrm{rad.\,s^{-1}}$, $\alpha_{mi} = \tilde{\alpha}_{mi} + \frac{\pi}{180}\,\mathrm{rad}$, $\beta_{mi} = \tilde{\beta}_{mi} + \frac{\pi}{180}\,\mathrm{rad}$ (scénario 12).

Pour ce problème, un programme C++ utilisant la bibliothèque IBEX a été réalisé (qTraj et le langage QUIMPER ont aussi été utilisés pour les premiers tests). Celui-ci permet de lancer les algorithmes 1, 2, 3, 4 sur les différents scénarios de simulation. Les temps de calculs pour 20 contractions (20 appels aux contracteurs d'évolution *forward-backward*, détection...) sur des ordinateurs récents (avec processeurs Intel Core 2 Duo, Intel Quad Core ou Intel Xeon) sont de cet ordre :

- Algorithme 1 (propagation de contraintes intervalles) : 1-2 min
- Algorithme 2 (propagation de contraintes intervalles+interdétections) : 5 min
- Algorithme 3 (propagation de contraintes intervalles+interdétections+3BCID) : 6-7 h
- Algorithme 4 (propagation de contraintes intervalles+Strangle) : 3 jours

Bien que les calculs aient été faits en 3D pour les amers, les résultats seront présentés dans le plan (seules les coordonnées (x, y) du robot et des amers seront considérées). Voici les résultats par scénario :

– Erreurs gaussiennes centrées.

Scénarios	$1_{k=3}$	$1_{k=3.1}$	$1_{k=3.2}$	$1_{k=3.3}$	$1_{k=3.4}$	$1_{k=3.5}$	$1_{k=3.6}$	$1_{k=3.7}$	$1_{k=3.8}$	$1_{k=3.9}$	$1_{k=4}$
Algorithmes testés	1	1	1	1	1	1	1	1	1	1	1,4
Algorithmes ayant donné des résultats	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	1	1	1	1	1,4

(4.22)

Scénarios	$2_{k=3}$	$2_{k=3.1}$	$2_{k=3.2}$	$2_{k=3.3}$	$2_{k=3.4}$	$2_{k=3.5}$	$2_{k=3.6}$	$2_{k=3.7}$	$2_{k=3.8}$	$2_{k=3.9}$	$2_{k=4}$
Algorithmes testés	1	1	1	1	1	1	1	1	1	1	1,4
Algorithmes ayant donné des résultats	1	1	1	1	1	1	1	1	1	1	1,4

(4.23)

Scénarios	$3_{k=3}$	$3_{k=3.1}$	$3_{k=3.2}$	$3_{k=3.3}$	$3_{k=3.4}$	$3_{k=3.5}$	$3_{k=3.6}$	$3_{k=3.7}$	$3_{k=3.8}$	$3_{k=3.9}$	$3_{k=4}$
Algorithmes testés	1	1	1	1	1	1	1	1	1	1	1,4
Algorithmes ayant donné des résultats	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	1

(4.24)

Scénarios	$4_{k=3}$	$4_{k=3.1}$	$4_{k=3.2}$	$4_{k=3.3}$	$4_{k=3.4}$	$4_{k=3.5}$	$4_{k=3.6}$	$4_{k=3.7}$	$4_{k=3.8}$	$4_{k=3.9}$	$4_{k=4}$
Algorithmes testés	1	1	1	1	1	1	1	1	1	1	1,4
Algorithmes ayant donné des résultats	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	1	1	1	1,4

(4.25)

Comme prévu, ces résultats ont abouti à des incohérences et donc des ensembles solutions vides dans certains cas. L'algorithme 1 a parfois donné des résultats (figures 4.21, 4.22 pour les résultats du scénario 4 par exemple). Les algorithmes 2 et 3 n'ont pas été testés sur ces scénarios. L'algorithme 4 n'a été lancé que pour des intervalles d'erreurs de la forme $[-4\sigma, 4\sigma]$ mais le scénario 3 a abouti a des ensembles vides (voir les figures 4.23 et 4.24 où les résultats des algorithmes 1 et 4 sur le scénario 4 pour 4σ sont superposés). On peut supposer qu'il y avait des données en-dehors des intervalles d'erreurs supposés pour le scénario 3, même pour 4σ. Il pourrait être intéressant de tester une méthode robuste par rapport aux données

aberrantes (en s'aidant par exemple des méthodes présentées dans [Jaulin, 2009b], [Sliwka et al., 2009] ou [Sliwka et al., 2011b]) pour traiter ces scénarios. Les méthodes probabilistes telles que l'algorithme de SAM sont aussi bien adaptées à ce type d'erreur (voir [Joly, 2010]).

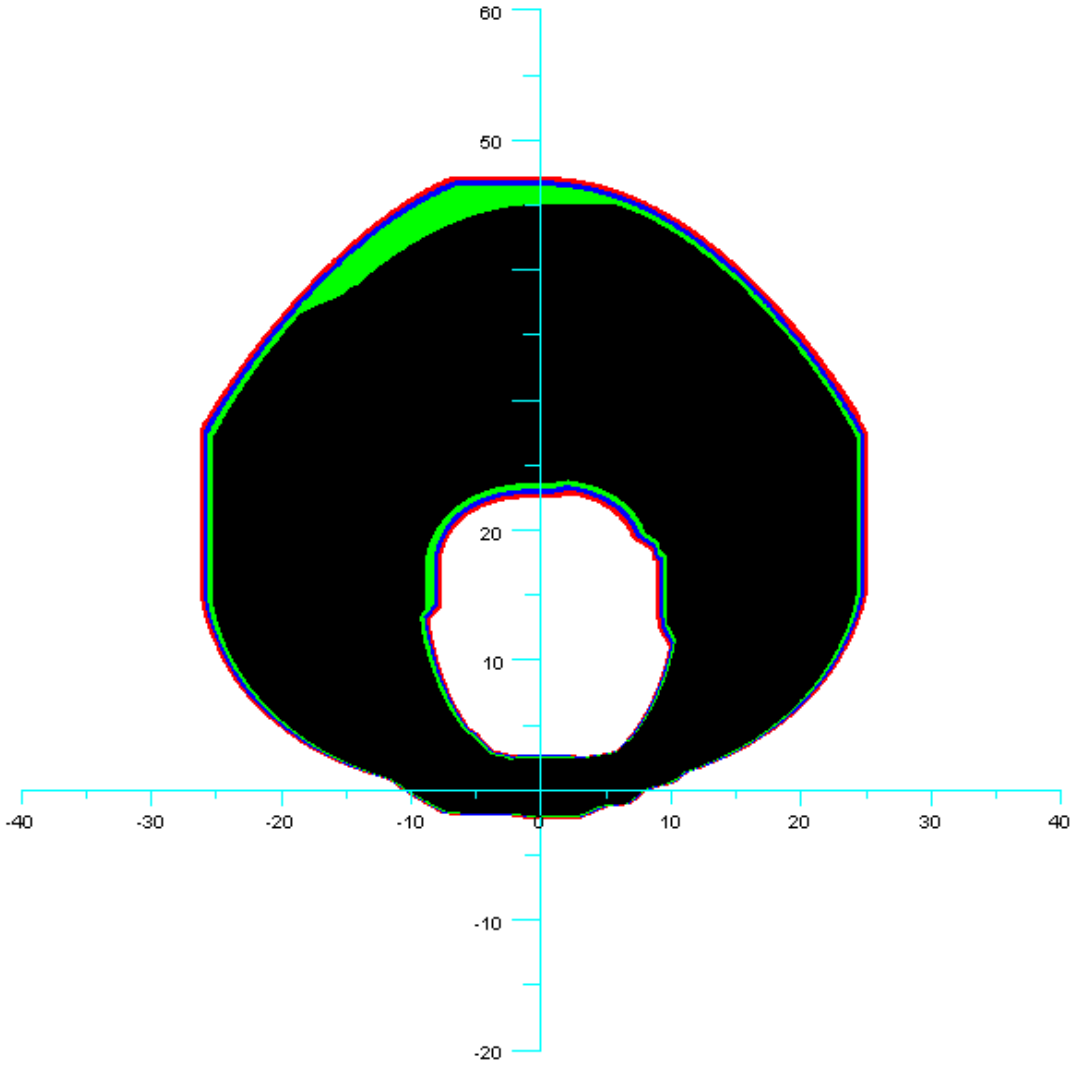


Figure 4.21 – Enveloppes de la trajectoire du robot trouvées par l'algorithme 1 pour le scénario 4 (rouge : $k = 4$, bleu : $k = 3.9$, vert : $k = 3.8$, noir : $k = 3.7$).

– Erreurs uniformes centrées.

Scénarios	5	6	7	8
Algorithmes testés	1,4	1,4	1,4	1,2,3,4
Algorithmes ayant donné des résultats	1,4	1,4	1,4	1,2,3,4

(4.26)

Les algorithmes 1 et 4 ont été lancés sur tous les scénarios et les 2 et 3 l'ont en plus été sur le scénario 8 (voir les figures 4.25 et 4.26 où les résultats des algorithmes 1, 2, 3 et 4 sur le scénario 8 sont superposés). On constate que ces algorithmes donnent des résultats très proches : on voit notamment que les pavés pour la trajectoire et la position des amers ne sont presque pas améliorés après le rajout d'équations



Figure 4.22 – Enveloppes des positions des amers trouvées par l'algorithme 1 pour le scénario 4 (rouge : $k = 4$, bleu : $k = 3.9$, vert : $k = 3.8$, noir : $k = 3.7$).

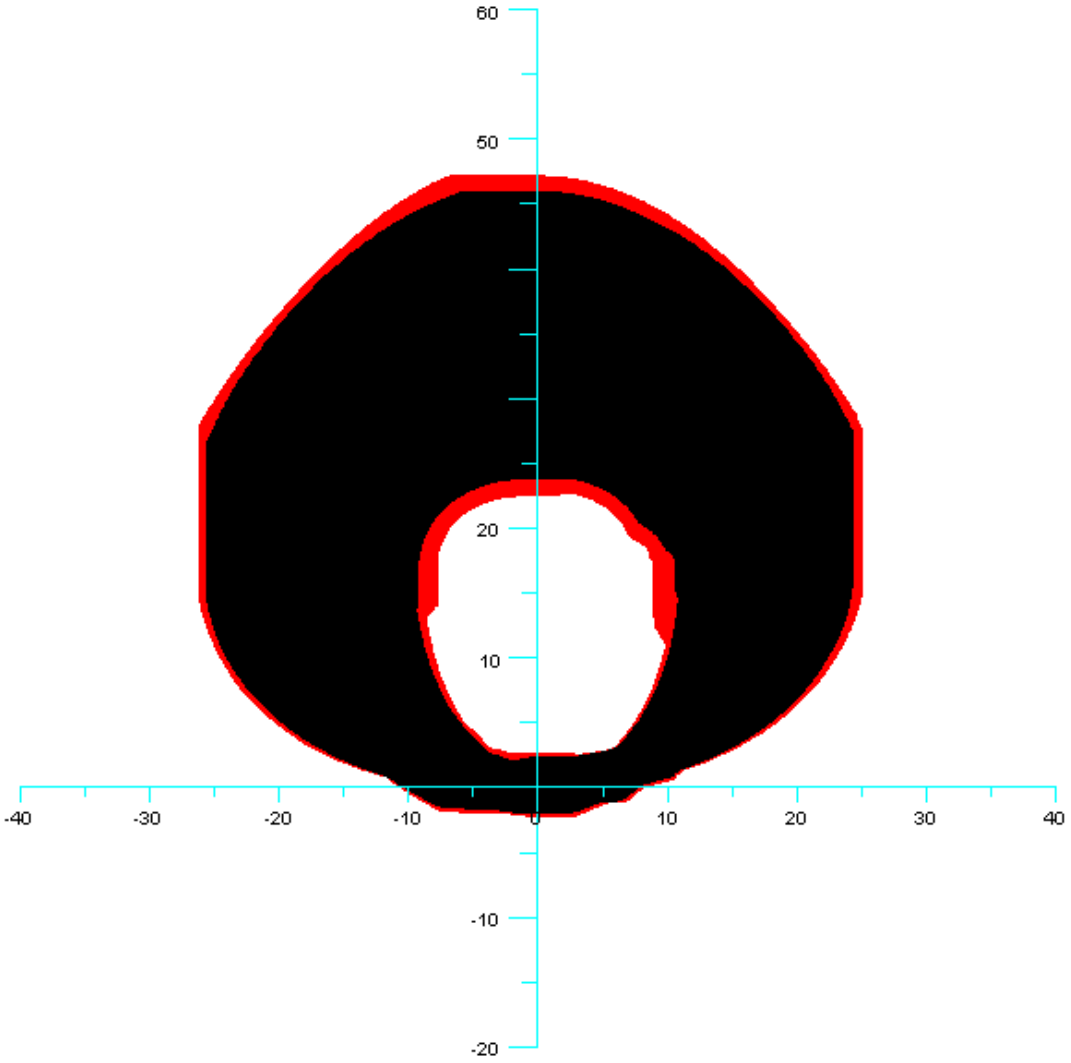


Figure 4.23 – Enveloppes de la trajectoire du robot trouvées pour $k = 4$ pour le scénario 4 (rouge : algorithme 1, noir : algorithme 4). Les résultats des 2 algorithmes sont presque confondus.

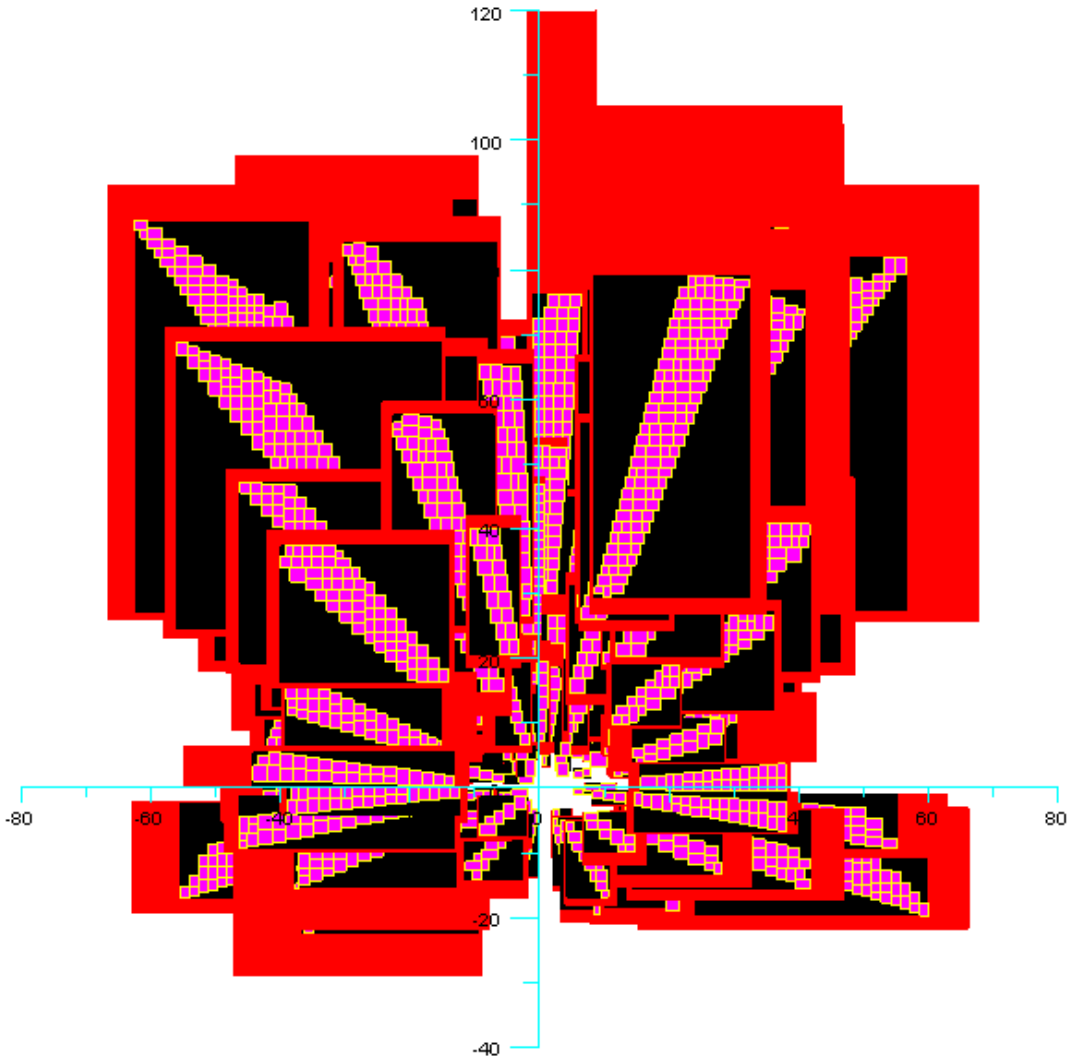


Figure 4.24 – Enveloppes des positions des amers trouvées pour $k = 4$ pour le scénario 4 (rouge : algorithme 1, noir : pavés produits à la fin de l'algorithme 4, magenta : sous-pavage produit par les bissections de l'algorithme 4). Il faut noter que les pavés noirs sont (en général) les plus petits pavés englobant les sous-pavages magenta trouvés en faisant des bissections (en fait il est possible que les pavés noirs soient encore plus petits que le plus petit pavé englobant car pour un amer donné, le pavé noir trouvé peut avoir été amélioré après la génération de son sous-pavage magenta). On remarque que l'algorithme 4 réussit à améliorer principalement les grands pavés. On peut donc conclure que l'algorithme 1 n'est pas optimal, mais il n'en est pas trop loin pour les petits pavés.

d'interdétections et pas beaucoup plus après des bissections, ce qui montre que l'algorithme 1 est déjà presque optimal. On remarque aussi que le dessin des sous-pavages représentant les positions des amers coïncide avec les pavés obtenus après changement de repère dans [Joly, 2010], ce qui montre que les résultats de cet algorithme (6) sont aussi proches de l'optimalité dans ce repère.

– Erreurs biaisées.

Scénarios	9	10	11	12
Algorithmes testés	1,4	1,4	1,4	1,4
Algorithmes ayant donné des résultats	1,4	1,4	1,4	1,4

(4.27)

Seuls les algorithmes 1 et 4 ont été lancés sur ces scénarios (voir par exemple les figures 4.27 et 4.28 pour le scénario 12). Les résultats confirment ce qui a été constaté précédemment.

On peut donc tirer plusieurs conclusions de ces études :

– La méthode de SLAM par intervalles présentée en 4.3.1 (qui vient de [Jaulin, 2009a]) n'est pas optimale mais n'en est pas loin : que l'on rajoute des équations supplémentaires propres au problème ou qu'on fasse des bissections, on n'améliore qu'assez peu la précision des pavés contenant la trajectoire du robot et les positions des amers (en considérant la longueur des intervalles $[x]$ et $[y]$ représentant les coordonnées du robot et des amers dans le plan comme critère de précision).
– Cette méthode est peu dépendante du type de problème : il n'y a pas de linéarisation ou d'étape spécifique d'initialisation des amers comme c'est le cas pour le SAM présenté dans [Joly, 2010]. qTraj et le langage QUIMPER permettent par exemple de lancer la résolution d'un problème de SLAM par intervalles en ne modifiant seulement que les équations d'évolution, observation et détection par rapport à ce problème (et les données). Les expériences du *Redermor* et de la *Daurade* ont par exemple aussi été traitées avec ces outils. Il faut noter que le principal apport de GESMI (utilisé et fait à la base pour les sous-marins) par rapport à qTraj, QUIMPER et IBEX est l'aspect graphique et opérationnel, ainsi qu'une meilleure gestion de la mémoire adaptée au problème sous-marin traité, qTraj et QUIMPER sont plutôt des outils généraux pour des tests de faisabilité et prototypages rapides, mais les concepts théoriques utilisés sont les mêmes. De ce fait, on peut dire que les méthodes par intervalles sont plutôt des méthodes globales, qui produisent rapidement des résultats exploitables sur des problèmes généraux (et aussi dans le sens où elles ne perdent aucune solution), alors que les méthodes probabilistes telles que le SAM sont plutôt des méthodes locales, performantes pour des cas particuliers.

4.5 Conclusion du chapitre

Le problème du SLAM, le principe des méthodes existantes et une méthode de résolution par propagation de contraintes par intervalles proposée pour la première fois dans [Jaulin, 2009a] ont été décrits. Cette dernière a été appliquée sur les données de la *Daurade*, grâce à une nouvelle version du programme GESMI (utilisé à l'origine pour obtenir les résultats présentés dans [Jaulin, 2009a]). De plus, l'algorithme a été comparé à d'autres pour essayer de caractériser son optimalité.

Dans ce chapitre, nous avons d'abord rappelé ce qu'était le SLAM, à travers l'exemple d'un robot sous-marin. Les différentes variantes et conditions possibles ont ensuite été énumérées et les hypothèses considérées dans le cadre de cette thèse ont été indiquées. Nous avons ensuite décrit synthétiquement le principe de résolution du SLAM, tel qu'il peut être utilisé pour différentes méthodes. L'algorithme de SLAM par propagation par

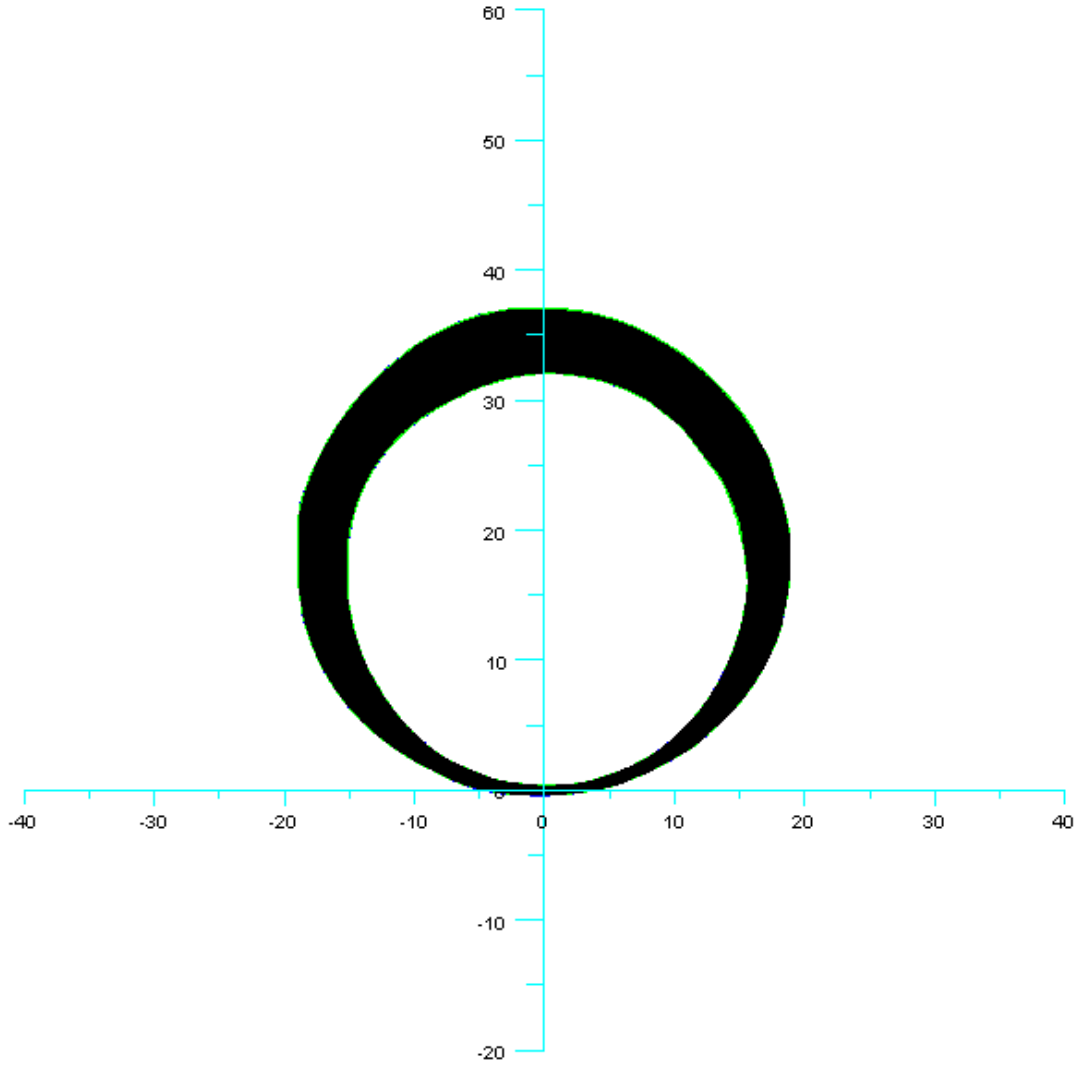


Figure 4.25 – Enveloppes de la trajectoire du robot trouvées pour le scénario 8 (rouge : algorithme 1, bleu : algorithme 2, vert : algorithme 3, noir : algorithme 4). Les résultats des différents algorithmes sont presque confondus : on voit que ni la prise en compte d'équations d'interdétections propres au problème (algorithme 2), ni l'ajout de bissections qu'elles soient sur des tranches de pavés (algorithme 3) ou complètes sur les positions des amers (algorithme 4) n'améliorent de manière significative l'estimation des positions x et y du robot.

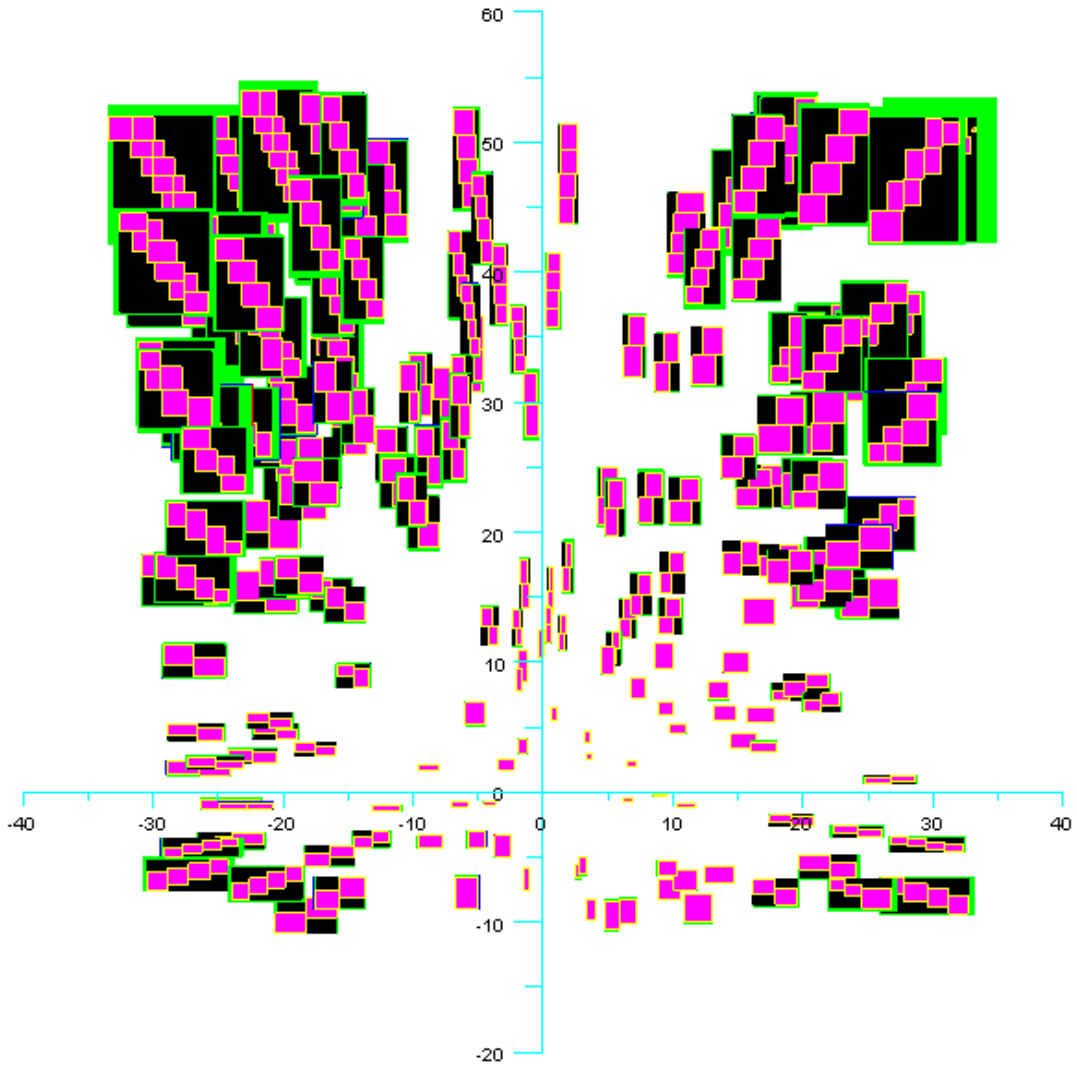


Figure 4.26 – Enveloppes des positions des amers trouvées pour le scénario 8 (rouge : algorithme 1, bleu : algorithme 2, vert : algorithme 3, noir : pavés produits à la fin de l'algorithme 4, magenta : sous-pavage produit par les bissections de l'algorithme 4). Il faut noter que les pavés noirs sont (en général) les plus petits pavés englobant les sous-pavages magenta trouvés en faisant des bissections (en fait il est possible que les pavés noirs soient encore plus petits que le plus petit pavé englobant car pour un amer donné, le pavé noir trouvé peut avoir été amélioré après la génération de son sous-pavage magenta). Cette figure est la plus importante : on voit que ni la prise en compte d'équations d'interdétections propres au problème (algorithme 2), ni l'ajout de bissections qu'elles soient sur des tranches de pavés (algorithme 3) ou complètes sur les positions des amers (algorithme 4) n'améliorent de manière significative l'estimation des positions x et y des amers. On peut alors en déduire que l'algorithme 1, qui est rapide et utilisable directement pour tout type de problème de SLAM est déjà presque optimal.

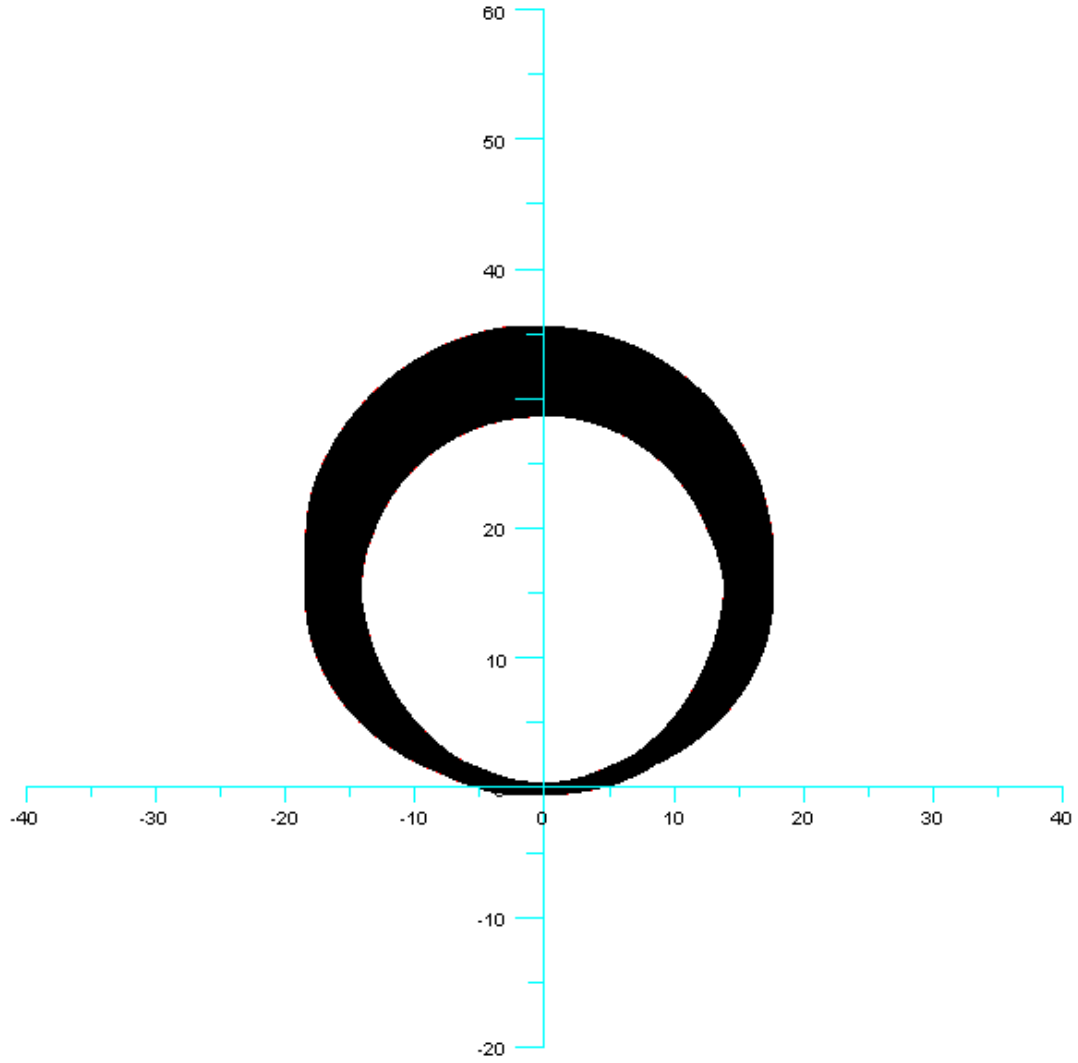


Figure 4.27 – Enveloppes de la trajectoire du robot trouvées pour le scénario 12 (rouge : algorithme 1, noir : algorithme 4). Les résultats des 2 algorithmes sont aussi presque confondus.

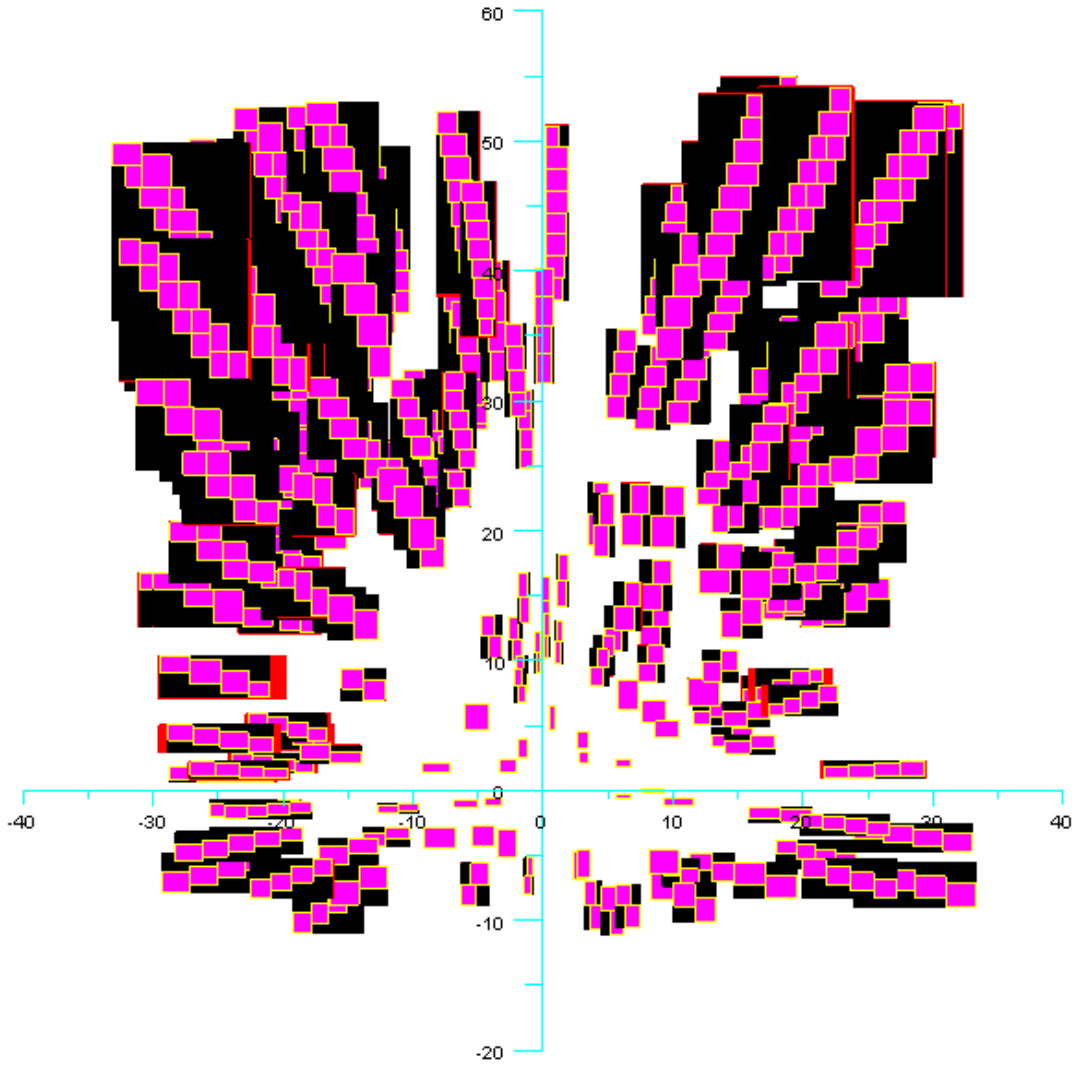


Figure 4.28 – Enveloppes des positions des amers trouvées pour le scénario 12 (rouge : algorithme 1, noir : pavés produits à la fin de l'algorithme 4, magenta : sous-pavage produit par les bissections de l'algorithme 4). Il faut noter que les pavés noirs sont (en général) les plus petits pavés englobant les sous-pavages magenta trouvés en faisant des bissections (en fait il est possible que les pavés noirs soient encore plus petits que le plus petit pavé englobant car pour un amer donné, le pavé noir trouvé peut avoir été amélioré après la génération de son sous-pavage magenta). Ces résultats confirment ce qui a été constaté avec le scénario 8 par exemple.

intervalles présenté pour la première fois dans [Jaulin, 2009a] a ensuite été décrit et appliqué sur les données de la *Daurade*. Ceci a conduit à la modification du programme GESMI existant : gestion d'images sonar bâbord et tribord, ajout du tracé de l'erreur, amélioration du processus d'aide à la détection de mines par un opérateur humain... Enfin, nous avons comparé les résultats de ce type de propagation avec d'autres algorithmes intervalles, sur des données simulées (issues d'une comparaison entre algorithmes intervalles et probabilistes faite dans [Joly, 2010]) pour mesurer l'optimalité de l'algorithme utilisé précédemment pour traiter les données des sous-marins *Daurade* et *Redermor*. Il en résulte que l'algorithme de SLAM par propagation par intervalles n'est pas optimal, mais n'en est pas trop loin et semble être un bon rapport entre précision des résultats, temps de calcul et facilité d'adaptation au problème particulier considéré.

Le SLAM sous-marin décrit (notamment à travers la mise au point et l'utilisation de GESMI) a mis en évidence une difficulté particulière : les mines ne sont visibles dans les images sonars qu'à des instants bien précis, qu'un opérateur humain doit au préalable rechercher en examinant attentivement toute les données sonar (ce qui est long et difficile car les données correspondent à des missions de plusieurs heures et des parcours de plusieurs kilomètres, et il faut une certaine expérience pour être capable de distinguer une mine d'un rocher dans de telles images). Bien que des algorithmes de traitement d'images pour la détection de mines dans les images sonars existent ou soient en cours d'élaboration, c'est encore bien souvent l'opérateur humain qui est le plus utilisé et le plus précis. Le chapitre suivant montrera comment il est possible de faciliter le travail de l'opérateur humain ou des algorithmes de détection en formalisant le problème des données fugaces (données significatives à des instants inconnus et courts, comme c'est le cas pour les mines sous-marines dans les images sonar) et en proposant une méthode de traitement.

Chapitre 5

Estimation d'état et SLAM avec données fugaces

Une nouvelle façon de formaliser les problèmes de localisation ou SLAM avec des robots munis de sonars, télémètres, caméras... sera présentée dans ce chapitre. Un algorithme de résolution testé sur des données simulées (représentant une localisation dynamique d'un robot de type char muni d'un télémètre rotatif laser, grâce à un amer connu et dans un environnement inconnu encombré par des obstacles mobiles) sera ensuite décrit en détail. Enfin, son extension au SLAM sous-marin sera évoquée. Ce chapitre présente la contribution académique principale de la thèse et a fait l'objet d'un article accepté dans la revue Automatica ([Le Bars et al., 2011a]).

La première section introduira le problème des données fugaces, avec l'exemple des sous-marins.

La deuxième section présentera de manière plus formelle ce problème, et expliquera notamment ce qu'est un point fugace sur une waterfall. Nous verrons que la présence d'un point fugace sur une waterfall se décrit par une condition de visibilité, que l'on cherchera à utiliser pour améliorer la localisation d'un robot. Un algorithme utilisant la propagation de contraintes sur des tubes sera mis au point et testé sur une simulation de robot muni d'un télémètre laser rotatif.

Dans la troisième section, nous expliquerons comment il est possible de modifier l'algorithme utilisé pour l'appliquer à un contexte de SLAM sous-marin.

La dernière section conclura ce chapitre.

5.1 Principe

Lors de la résolution du problème de SLAM sous-marin présenté dans la section précédente, les données sonar n'avaient pas été directement utilisées, c'était un opérateur humain qui les avaient analysées pour indiquer où il avait vu des amers (mines...). La nouveauté va être ici de tenter de remplacer le contracteur réalisé indirectement par l'opérateur humain qui recherche les mines dans les données sonar par un "vrai" contracteur qui prend en entrée les données sonar pour automatiser ou améliorer les calculs de position du

sous-marin et de mines. Ce nouveau contracteur effectuera un moyennage et seuillage de l'image sonar (le traitement d'image utilisé est donc basique, on se concentre juste sur le principe) pour éliminer les zones où il est sûr qu'il n'y a pas de mines, et ces zones seront alors utilisées pour contracter l'intervalle où se trouve la mine, qui était calculé à partir des données de navigation et (éventuellement) des détections de l'opérateur (voir figure 5.1). Le but est de faciliter/minimiser au maximum l'intervention d'un opérateur

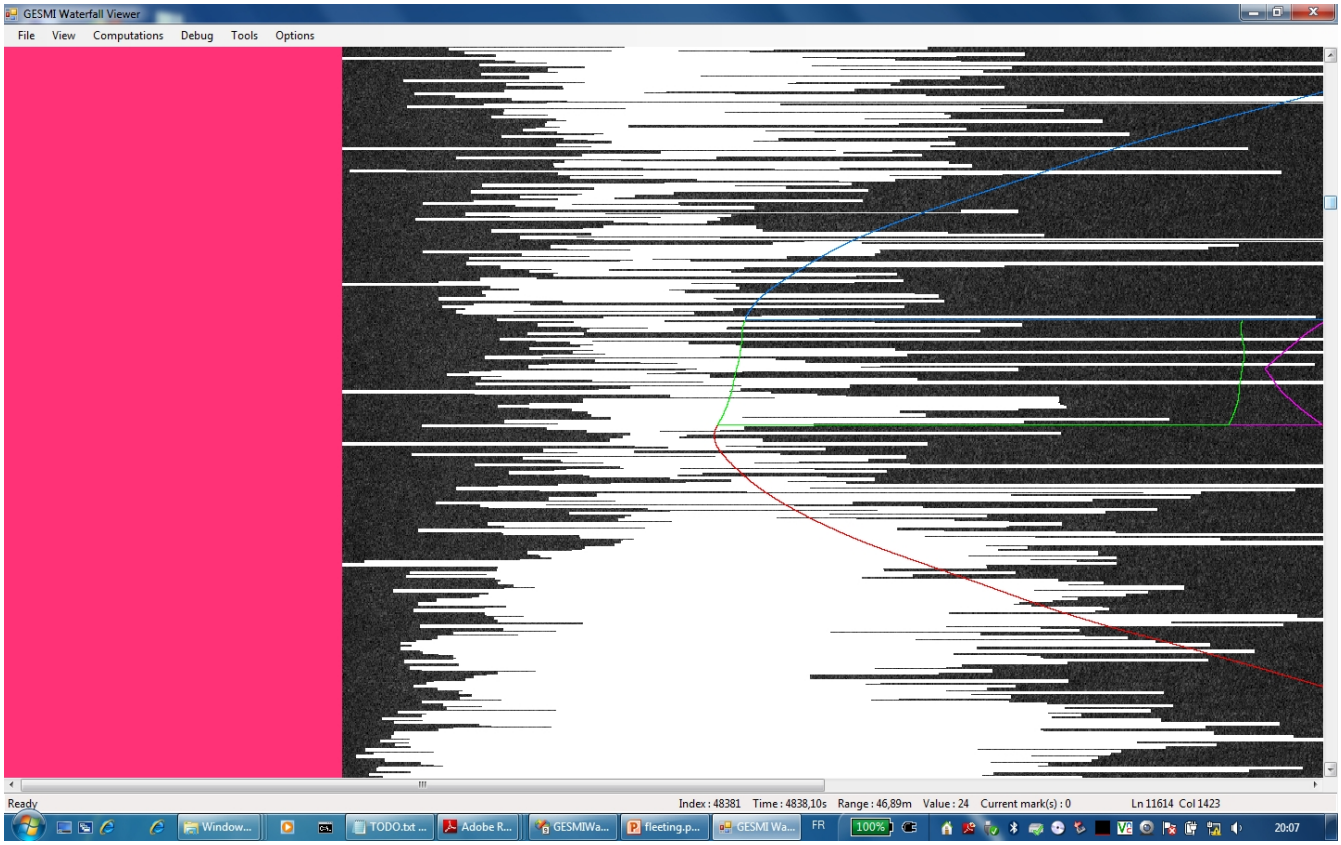


Figure 5.1 – Partie d'image sonar du *Redermor* affichée dans GESMI. La zone rose à gauche est considérée comme étant des données sonar non exploitables pour détecter un amer. En effet, elle correspond à la water column (voir 2.2.1). Les rectangles blancs sont les pavés récupérés suite à un seuillage de l'image sonar : il est possible qu'un amer s'y cache. La zone entourée de vert est l'indication qu'un amer devrait avoir été revu sur l'image sonar d'après les calculs de GESMI. En regardant l'intersection entre cette zone et les pavés résultants du seuillage, on voit qu'il serait possible de réduire la taille de la zone verte estimée par GESMI de manière automatique.

humain (notamment en réduisant la taille des zones de recherche sur l'image sonar) et ainsi de s'approcher d'une automatisation complète du SLAM. Nous allons voir dans les sections suivantes que la détection des mines dans l'image sonar est un problème de données fugaces : en effet, les mines ne sont vues par le sonar qu'à des instants très précis (fugaces) et inconnus. Ceci sera valable pour des robots munis de sonar, radar, télémètre, caméra...

5.2 Formalisation et résolution du problème des données fugaces avec les tubes

5.2.1 Problème des données fugaces

Dans cette section, nous allons étudier le cas d'une estimation d'état offline (avec des équations potentiellement non linéaires) où des mesures sont seulement significatives lorsque certaines conditions d'égalité sont satisfaites, et ceci à des instants précis (fugaces) inconnus (voir [Le Bars et al., 2011a]). C'est pour cela que les données sont qualifiées de fugaces (fleeting en anglais). Un problème d'estimation d'état avec données fugaces peut être représenté par une équation d'état classique, avec une condition de visibilité :

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{b}(t) \\ h(\mathbf{x}(t)) = 0 \Rightarrow g(\mathbf{x}(t)) \in \mathbb{W}(t) \end{cases} \tag{5.1}$$

où $t \in \mathbb{R}$ est le temps, $\mathbf{x}(t)$ est le vecteur d'état, $\mathbf{b}(t)$ est le vecteur de bruit, $\mathbf{f} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ est la fonction d'*évolution*, $h : \mathbb{R}^n \to \mathbb{R}$ est la fonction de *visibilité* et $g : \mathbb{R}^n \to \mathbb{R}$ est la fonction d'*observation*. La fonction $\mathbb{W}$ est appelée *waterfall* et associe à chaque temps t un sous-ensemble de $\mathbb{R}$. Ce nom a été choisi car $\mathbb{W}$ correspond à une image sonar dans le cas d'un robot sous-marin (voir chapitre 2). Les fonctions $\mathbf{f}, h, g$ sont supposées continues et différentiables.

Un *point fugace* est une paire $(t, g(\mathbf{x}(t)))$ telle que $h(\mathbf{x}(t)) = 0$. Dans la figure 5.2, on a 4 points fugaces

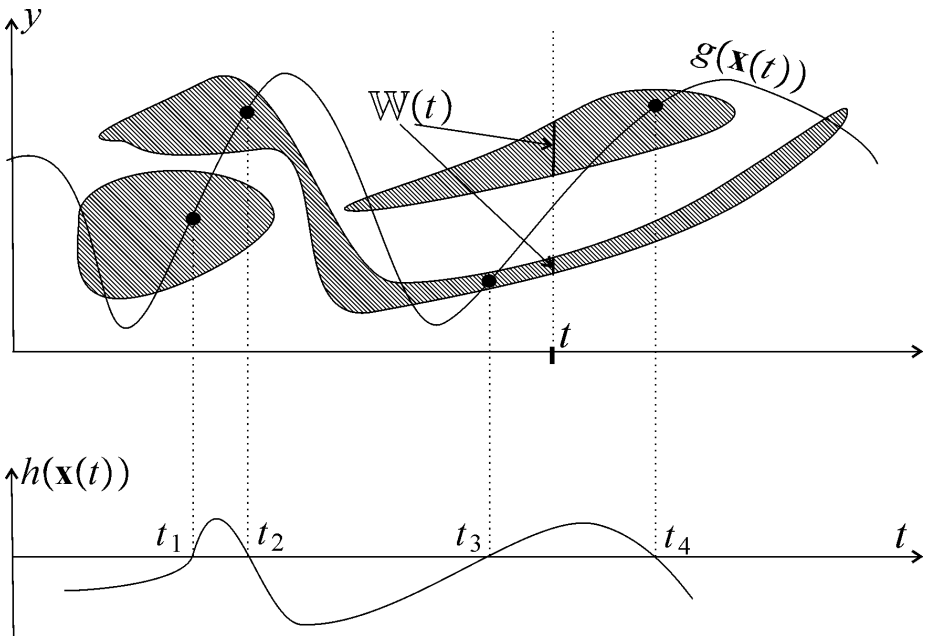


Figure 5.2 – Les points fugaces (points noirs) sont supposés appartenir à la waterfall $\mathbb{W}(t)$.

(points noirs pouvant représenter des objets cachés dans leur environnement). Les 4 instants fugaces correspondants sont t_1, t_2, t_3, t_4. L'existence et la position de ces points fugaces sont inconnues mais peuvent être estimées en utilisant la waterfall $\mathbb{W}(t)$ et les équations d'état. La plupart des éléments de $\mathbb{W}(t)$ ne sont en général pas des données significatives.

En pratique, la waterfall est obtenue à partir de capteurs tels que des sonars, radars, télémètres ou encore caméras et est souvent représentée par une image. Par exemple, dans la waterfall représentée dans la figure

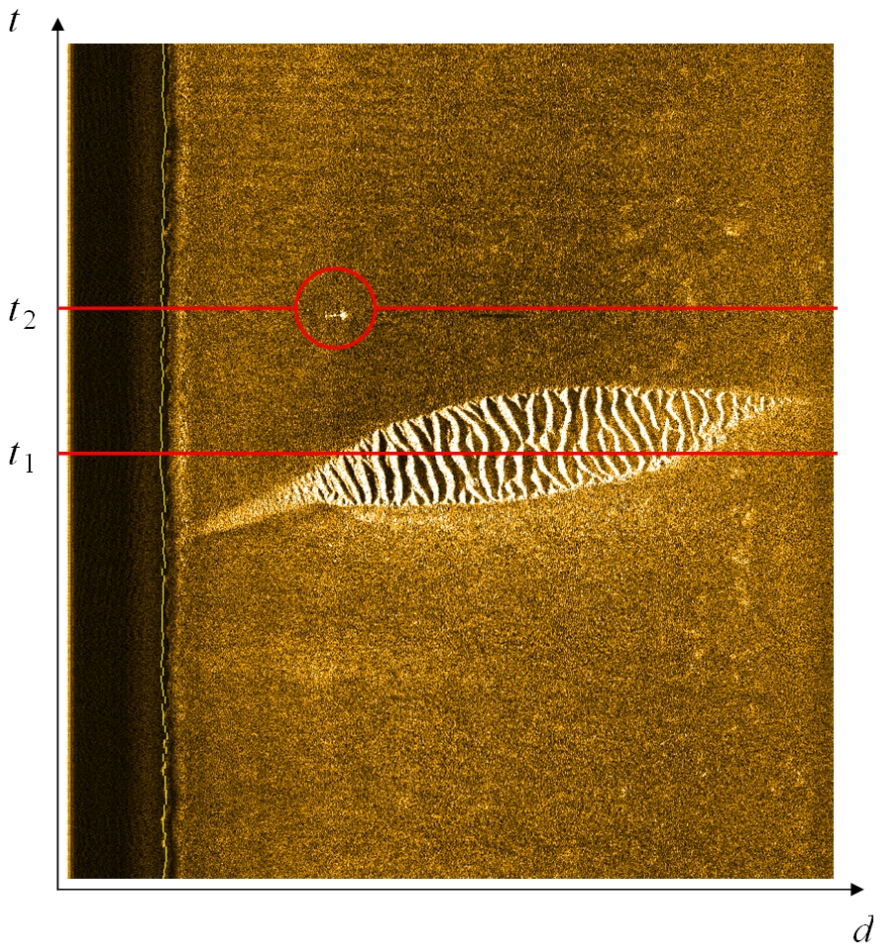


Figure 5.3 – Partie d'une image sonar venant du *Redermor*, où on voit des rides de sable et une mine.

5.3 (voir la sous-section 2.2.1 pour plus d'informations sur les données produites par des sonars latéraux), les 2 segments noirs verticaux correspondent à la partie du signal enregistrée par un sonar latéral aux instants t_1 et t_2. A t_2, une mine est vue par le sonar (point brillant entouré dans le cercle) alors qu'à t_1 les données ne correspondent qu'à des rides de sable, mais il est difficile d'affirmer qu'il n'y a aucune mine cachée dedans. Par contre, en dehors des zones où il y a des points brillants (zones sombres), on peut affirmer sans trop de risques qu'il n'y a aucune mine. Ainsi, avec une waterfall, on obtient des zones où il est sûr qu'il n'y a pas d'amer plutôt que des zones où on détecte un amer. De même, il est possible d'obtenir une waterfall à partir de télémètres laser rotatifs comme par exemple ceux des robots BOURRICOTs (voir 2.3.2). La figure 5.4 montre comment il est possible de dessiner aussi les données d'un télémètre laser rotatif sous la même forme que pour un sonar latéral. Il faut noter que les informations données par un tel télémètre laser sont moins riches qu'avec un sonar latéral (un objet caché derrière un autre est en général invisible car le faisceau laser est plus directif qu'une onde ultrasonore). Le fait que le télémètre soit rotatif rend la waterfall plus difficilement interprétable pour un humain (il en serait de même si on produisait une

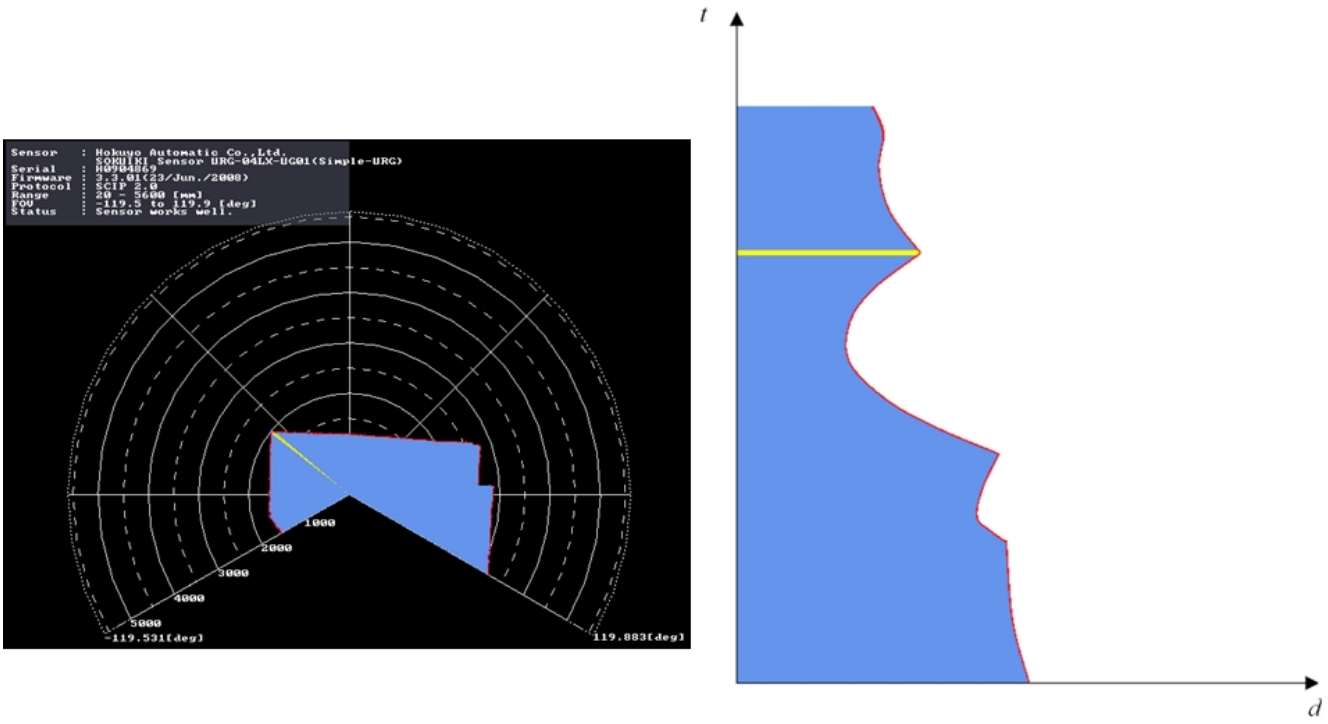


Figure 5.4 – A gauche, image produite par un télémètre laser rotatif dans une petite pièce. Le rouge correspond aux murs de la pièce (comme si c'était une vue du dessus), les zones bleues à des zones vides d'obstacles et le jaune à ce qui a changé depuis le dernier tour de laser (le laser était presque immobile). A droite, waterfall produite schématiquement à partir de l'image de gauche. Le rouge, bleu et jaune correspondent aux mêmes éléments qu'à gauche. On voit que les discontinuités correspondent aux coins de la pièce. Cette waterfall est obtenue en mettant les distances mesurées par le télémètre à mesure qu'il tourne les unes au-dessus des autres.

waterfall à partir d'un sonar rotatif comme celui de *SAUC'ISSE*, voir figure 2.16). Cependant, l'algorithme de résolution de problèmes avec données fugaces proposé dans la sous-section suivante sera valable pour les robots sous-marins avec sonars latéraux comme pour les robots terrestres avec télémètres lasers rotatifs.

5.2.2 Résolution du problème d'estimation d'état avec données fugaces

Principe

Bien que de nombreux travaux aient déjà été réalisés sur des problèmes d'estimation d'état offline non linéaire (voir par exemple [Abdallah et al., 2008], [Jaulin et al., 2001a], [Raissi et al., 2004], [Gning and Bonnifait, 2006], [Videau et al., 2009] dans les cas à erreur bornée par exemple), le problème d'estimation d'état avec données fugaces tel qu'on l'a défini ne semble pas avoir été abordé jusqu'à maintenant, même s'il correspond à des situations courantes notamment en robotique. Une nouvelle approche basée sur une propagation de contraintes sur des tubes va donc être présentée.

On peut décomposer le problème initial défini par 5.1 sous la forme d'un CSP ayant pour contraintes les équations et conditions

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{b}(t) \\ v(t) = h(\mathbf{x}(t)) \\ \dot{y}(t) = \frac{\partial g}{\partial \mathbf{x}}(\mathbf{x}(t)).\dot{\mathbf{x}}(t) \\ y(t) = g(\mathbf{x}(t)) \\ v(t) = 0 \Rightarrow y(t) \in \mathbb{W}(t), \end{cases} \tag{5.2}$$

pour variables les trajectoires $\mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{b}(t), y(t), \dot{y}(t), v(t)$ et pour domaines les tubes (intervalles de trajectoires) $[\mathbf{x}](t), [\dot{\mathbf{x}}](t), [\mathbf{b}](t), [y](t), [\dot{y}](t), [v](t)$ en introduisant les trajectoires $y(t), \dot{y}(t), v(t)$. Ceci permettra de gérer les contraintes indépendamment et simplifiera la construction des contracteurs associés. Le but sera de trouver une enveloppe contenant toutes les trajectoires $\mathbf{x}(t)$ compatibles avec les équations 5.1 (ou 5.2), la waterfall $\mathbb{W}(t)$ et un pavé $[\mathbf{x}(0)]$ pour condition initiale.

Il faut noter que les équations 5.2 vont mettre en jeu des dérivées. Même si en général la dérivée d'un tube n'est pas définissable (voir sous-section 3.3.3), nous avons des expressions analytiques pour des tubes contenant des dérivées dans notre cas. En effet, vu que

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), t) + \mathbf{b}(t) \\ \dot{y}(t) &= \frac{\partial g}{\partial \mathbf{x}}(\mathbf{x}(t)).\dot{\mathbf{x}}(t) \\ \dot{v}(t) &= \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}(t)).\dot{\mathbf{x}}(t), \end{aligned}$$

les tubes contenant les fonctions $\dot{\mathbf{x}}(t)$, $\dot{y}(t)$ et $\dot{v}(t)$ peuvent être obtenus à partir du tube $[\mathbf{x}](t)$:

$$\begin{aligned} [\dot{\mathbf{x}}](t) &= [\mathbf{f}]([\mathbf{x}](t), t) + [\mathbf{b}](t) \\ [\dot{\mathbf{y}}](t) &= \left[\frac{\partial g}{\partial \mathbf{x}}\right]([\mathbf{x}](t)).[\dot{\mathbf{x}}](t) \\ [\dot{\mathbf{v}}](t) &= \left[\frac{\partial h}{\partial \mathbf{x}}\right]([\mathbf{x}](t)).[\dot{\mathbf{x}}](t) \end{aligned}$$

où $[\mathbf{f}]$, $\left[\frac{\partial g}{\partial \mathbf{x}}\right]$ et $\left[\frac{\partial h}{\partial \mathbf{x}}\right]$ sont des fonctions d'inclusion (par exemple des fonctions d'inclusion naturelles) pour $\mathbf{f}$, $\frac{\partial g}{\partial \mathbf{x}}$ et $\frac{\partial h}{\partial \mathbf{x}}$ (dont on suppose qu'on a une expression analytique).

Contraction de la relation de visibilité

La difficulté principale du CSP défini par les contraintes 5.2 est la suivante : étant donné 3 tubes $[v](t)$, $[y](t)$, $[\dot{y}](t)$ associés aux trajectoires $v(t)$, $y(t)$, $\dot{y}(t)$, contracter les tubes $[v](t)$, $[y](t)$ par rapport à la relation

$$v(t) = 0 \Rightarrow y(t) \in \mathbb{W}(t). \tag{5.3}$$

Pour cela, voici 2 nouveaux théorèmes démontrés au cours de la thèse, le premier pour la contraction de $[y](t)$ et le second pour la contraction de $[v](t)$.

Théorème 1 *Si* $0 \in v([t])$ *alors pour tout* $t \in \mathbb{R}$,

$$y(t) \in \bigcup_{\tau \in [t]} \left((\mathbb{W}(\tau) \cap [y](\tau)) + \int_{\tau}^{t} [\dot{y}](\alpha)\, d\alpha \right). \tag{5.4}$$

Démonstration. Supposons que $0 \in v([t])$. Comme $v(t)$ est une fonction continue, d'après le théorème des valeurs intermédiaires (ou théorème de Bolzano), on a

$$\exists \tau \in [t], v(\tau) = 0.$$

Ensuite, d'après (5.3) et le fait que $y(\tau) \in [y](\tau)$, on a

$$y(\tau) \in \mathbb{W}(\tau) \cap [y](\tau). \tag{5.5}$$

En prenant en compte le fait que

$$y(t) = y(\tau) + \int_{\tau}^{t} \dot{y}(\alpha)\, d\alpha,$$

et d'après (3.17),

$$\dot{y}(\alpha) \in [\dot{y}](\alpha) \Rightarrow \int_{\tau}^{t} \dot{y}(\alpha)\, d\alpha \in \int_{\tau}^{t} [\dot{y}](\alpha)\, d\alpha$$

on obtient

$$\begin{array}{rcl} y(t) = y(\tau) + \int_{\tau}^{t} \dot{y}(\alpha)\, d\alpha & \Rightarrow & y(t) \in \left\{ y(\tau) + \int_{\tau}^{t} \dot{y}(\alpha)\, d\alpha \right\} \\ & \Rightarrow & y(t) \in \{y(\tau)\} + \left\{ \int_{\tau}^{t} \dot{y}(\alpha)\, d\alpha \right\} \\ & \Rightarrow & y(t) \in (\mathbb{W}(\tau) \cap [y](\tau)) + \int_{\tau}^{t} [\dot{y}](\alpha)\, d\alpha. \end{array}$$

Et comme $\tau \in [t]$, on trouve (5.4). ■

On peut conclure à partir de ce théorème que si $0 \in v([t])$, le tube $[y](t)$ pour $y(t)$ peut être contracté de cette manière :

$$\begin{array}{rcl} [y](t) & := & [y](t) \cap \\ & & \bigcup_{\tau \in [t]} \left((\mathbb{W}(\tau) \cap [y](\tau)) + \int_{\tau}^{t} [\dot{y}](\alpha)\, d\alpha \right). \end{array} \tag{5.6}$$

Une extension de ce théorème peut être faite si v est une fonction vectorielle en utilisant les outils mathématiques présentés dans [Goldsztejn and Jaulin, 2006].

Corollaire 1 *Si* $0 \in v([t])$ *et* $0 \notin \dot{v}([t])$*, alors il existe un unique instant fugace dans* $[t]$*.*

Ce corollaire peut être utilisé pour vérifier le nombre de détections mais ne nous sera pas utile pour la contraction des tubes. Il est à noter que ceci nécessite d'avoir un tube pour $\dot{v}(t)$, qui peut être obtenu grâce à la relation $\dot{v}(t) = \frac{\partial v}{\partial \mathbf{x}}(\mathbf{x}(t)).\dot{\mathbf{x}}(t)$.

Corollaire 2 *Si* $0 \in v([t])$ *alors pour tout* $t \in \mathbb{R}$*,*

$$y(t) \in \bigcup_{\tau \in [t]} ((\mathbb{W}(\tau) \cap [y](\tau)) + [y_0](t) - [y_0](\tau)) \tag{5.7}$$

avec

$$[y_0](t) = \int_0^t [\dot{y}](\alpha)\, d\alpha. \tag{5.8}$$

Démonstration. y_0 est la primitive de $\dot{y}$ telle que $y_0(0) = 0$. ■

Ainsi, pour contracter $[y](t)$, on peut calculer d'abord le tube $[y_0](t) = \int_0^t [\dot{y}](\alpha)\, d\alpha$ pour tout $t \in [t]$ puis utiliser les résultats mémorisés pour faire la contraction

$$\begin{array}{rcl} [y](t) & := & [y](t) \cap \\ & & \bigcup_{\tau \in [t]} ((\mathbb{W}(\tau) \cap [y](\tau)) + [y_0](t) - [y_0](\tau)). \end{array}$$

Théorème 2 *On a l'implication suivante :*

$$\forall t \in [t], [y](t) \cap \mathbb{W}(t) = \emptyset \Rightarrow 0 \notin v([t]). \tag{5.9}$$

Démonstration. La démonstration peut être faite par contradiction. Supposons que $0 \in v([t])$, alors d'après le théorème des valeurs intermédiaires, on a $\exists \tau \in [t], v(\tau) = 0$ et donc $y(\tau) \in \mathbb{W}(\tau)$ d'après (5.3). Comme $y(\tau) \in [y](\tau)$, on a $y(\tau) \in [y](\tau) \cap \mathbb{W}(\tau)$ ce qui contredit l'hypothèse $\forall t \in [t], [y](t) \cap \mathbb{W}(t) = \emptyset$. ■

Une conséquence directe de ce théorème est que si on a trouvé un intervalle $[t]$ de $\mathbb{R}$ tel que

$$\forall t \in [t], [y](t) \cap \mathbb{W}(t) = \emptyset,$$

et qu'ensuite on trouve un $t_1 \in [t]$ tel que $0 \notin [v](t_1)$, alors le tube $[v](t)$ pour $v(t)$ peut être contracté par l'une de ces opérations (mutuellement exclusives) :

$$\begin{array}{rcl} \text{Si } [v](t_1) & > & 0, \text{ alors } \forall t \in [t], [v](t) := [v](t) \cap \mathbb{R}^+ \\ \text{Si } [v](t_1) & < & 0, \text{ alors } \forall t \in [t], [v](t) := [v](t) \cap \mathbb{R}^-. \end{array}$$

Exemple. Considérons le CSP suivant :

$$\left\{ \begin{array}{c} v(t) = 0 \Rightarrow y(t) \in \mathbb{W}(t) \\ y(t) = y_\tau + \int_\tau^t \dot{y}(\alpha) d\alpha \\ v(t) \in [v](t), y(t) \in [y](t) \end{array} \right. \tag{5.10}$$

où $\mathbb{W}(t)$ et les tubes initiaux $[v](t)$, $[y](t)$ sont donnés par la figure 5.5 (a). Le point fugace inconnu est représenté par un point noir. Cherchons à contracter $[v](t)$, $[y](t)$. Avec le théorème 2, on peut contracter le tube $[v](t)$ en supprimant les zones noires de la figure 5.5 (b).

On en déduit aussi qu'il existe un $\tau \in [\tau]$ correspondant à un point fugace et comme ce point doit appartenir à la waterfall $\mathbb{W}(t)$ et au tube $[y](t)$, il ne peut être que dans la zone noire représentée 5.5 (c), contenue dans le pavé $[\tau] \times [y_\tau]$ figure 5.5 (d). On a

$$y_\tau \in \bigcup_{\tau \in [\tau]} (\mathbb{W}(\tau) \cap [y](\tau)).$$

Le théorème 1 et ses conséquences peuvent être utilisés pour contracter le tube $[y](t)$ (l'application du théorème 1 sera illustrée en détail sur un exemple d'application plus complet dans la partie 5.2.2).

Algorithme de contraction

Voici les différentes étapes :

Initialisation. Tous les tubes sont initialisés avec les valeurs déjà connues. Par exemple, si on a une condition initiale sur le vecteur d'état, le tube correspondant $[\mathbf{x}](t)$ sera contracté à $t = 0$. La waterfall $\mathbb{W}(t)$ est supposée connue pour tout t (on suppose qu'on est dans le cas d'un problème d'estimation d'état offline).

Contracteur d'évolution. Une propagation *forward-backward* par rapport au temps combinée avec une propagation *forward-backward* sur l'expression pour chaque pas de temps (voir sous-section 3.4.1) sera efficace pour contracter les tubes $[\mathbf{x}](t), [\dot{\mathbf{x}}](t)$ par rapport à la contrainte $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{b}(t)$.

Contracteurs d'observation. Les contractions des tubes $[\mathbf{x}](t), [\dot{\mathbf{x}}](t), [\dot{y}](t), [y](t), [v](t)$ par rapport aux contraintes $v(t) = h(\mathbf{x}(t)), \dot{y}(t) = \frac{\partial g}{\partial \mathbf{x}}(\mathbf{x}(t)).\dot{\mathbf{x}}(t)$ et $y(t) = g(\mathbf{x}(t))$ peuvent être faites avec HC4Revise.

Contracteur de visibilité. La relation $v(t) = 0 \Rightarrow y(t) \in \mathbb{W}(t)$ peut contracter les tubes $[v](t)$ et $[y](t)$. La contraction du tube $[y](t)$ est basée sur le théorème 1 et est illustrée par la figure 5.6.

Voici la méthode :

1. On prend 2 tranches $[v](k_1)$ et $[v](k_2)$ telles que $0 \in [v](k)$ pour tout $k \in [k_1 + 1, k_2 - 1]$ et telles que $[v](k_1)$ et $[v](k_2)$ soient de signes opposés (voir figure 5.6 (a)).
2. Pour chaque $j \in [k_1 + 1, k_2 - 1]$, on calcule le sous-tube composé des tranches $(\mathbb{W}(j) \cap [y](j)) + [y_0](k) - [y_0](j)$, pour tout $k \in [k_1 + 1, k_2 - 1]$ (voir figure 5.6 (b) et (c)).
3. On calcule l'union des sous-tubes résultants et on intersecte cette union avec le tube initial $[y](t)$ (voir figure 5.6 (d)).

La contraction du tube $[v](t)$ est basée sur le théorème 2. Le principe est de trouver un sous-tube $\{[y](k_1), \ldots, [y](k_2)\}$ du tube $[y](t)$ qui n'intersecte pas $\mathbb{W}(t)$. Comme toutes les trajectoires correspondantes $v(t)$ devraient avoir le même signe, il suffit qu'on trouve une tranche du sous-tube $[v](k_1), \ldots, [v](k_2)$ qui soit positive (ou négative) pour pouvoir le contracter.

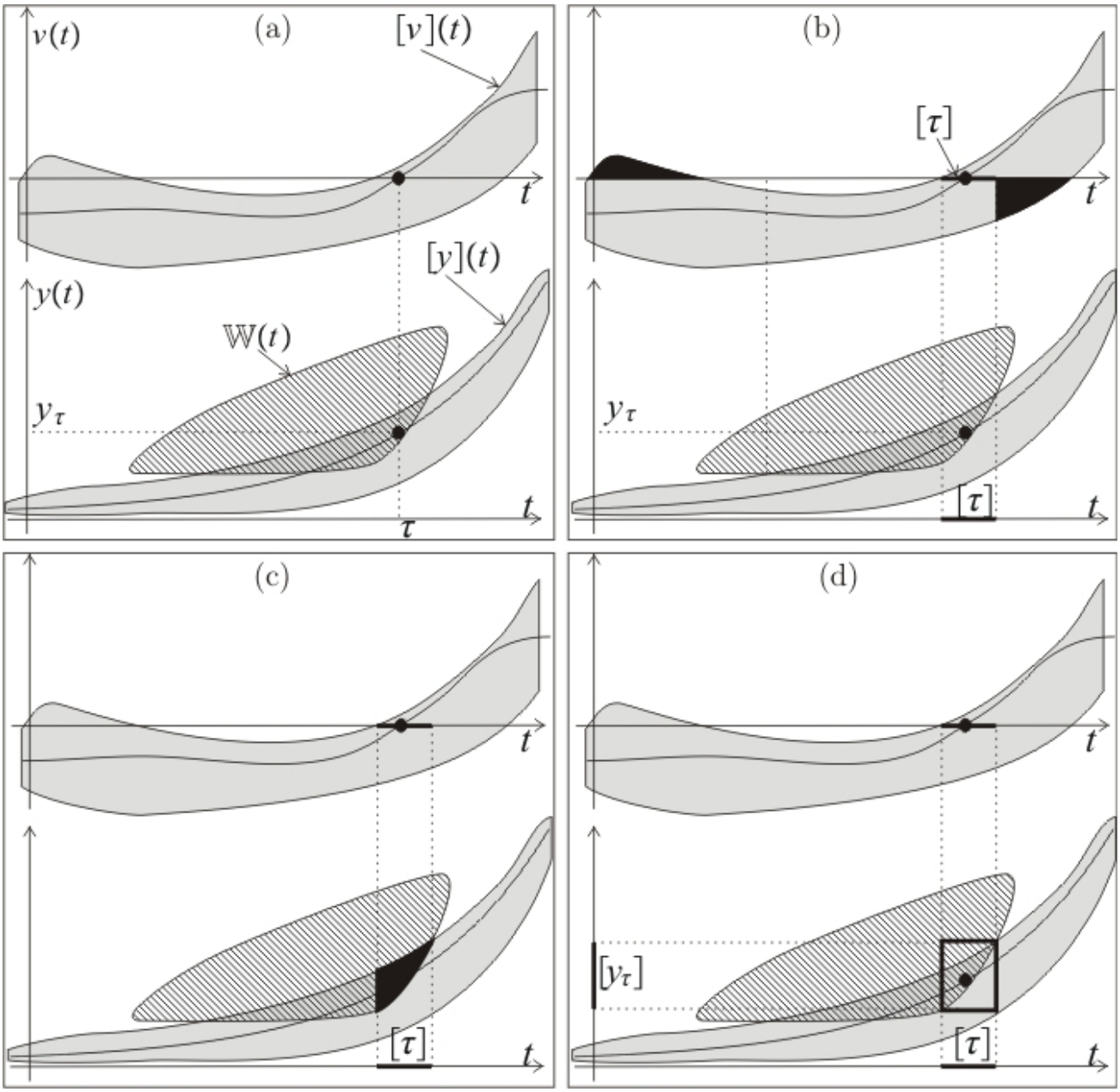


Figure 5.5 – Exemple de contraction avec la relation de visibilité.

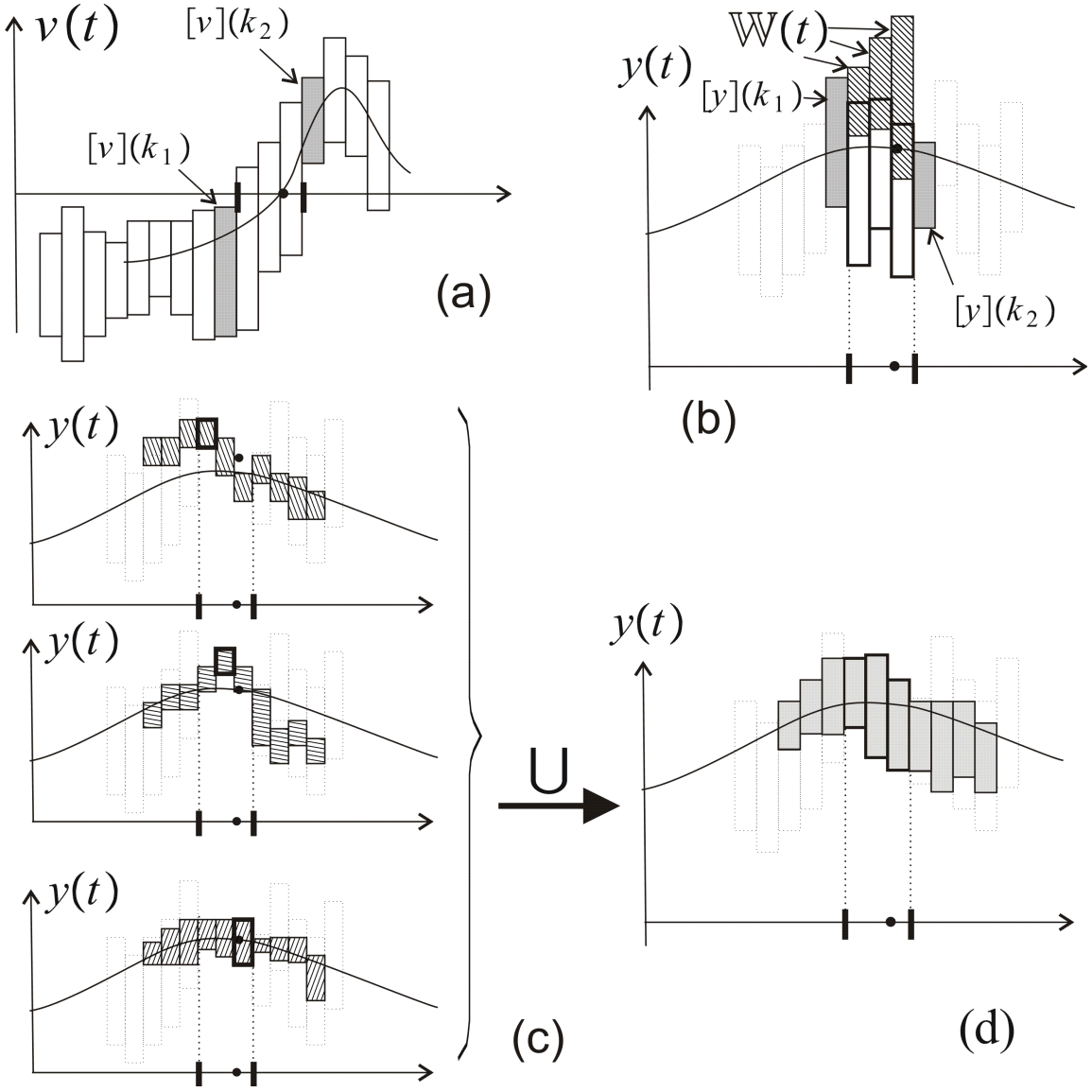


Figure 5.6 – Contraction du tube $[y](t)$ quand on sait (par le signe de v) qu'une donnée visible a été détectée. a) Sélection de tranches telles que $0 \in [v](k)$ pour tout $k \in [k_1+1, k_2-1]$ (étape 1). b) Illustration de l'intersection entre $[y](k)$ et $\mathbb{W}(k)$. c) Calculs des sous-tubes temporaires pour chaque $k \in [k_1+1, k_2-1]$ (étape 2). d) Tube $[y](t)$ résultant de l'union des sous-tubes calculés en c) suivie de l'intersection de cette union avec le tube initial $[y](t)$ (étape 3).

Comme pour tout algorithme de propagation de contraintes par intervalles (ici les variables sont des trajectoires et les domaines des intervalles de trajectoires ou tubes), les contractions doivent être répétées jusqu'à obtention d'un point fixe.

Exemple d'application

Considérons le problème de localisation dynamique d'un robot roulant de type char se déplaçant dans un environnement plan encombré par des objets fixes et mobiles (inconnus) masquant par moment un amer ponctuel et immobile dont la position est connue (voir figure 5.7). Les seuls capteurs utilisés par le robot sont un télémètre laser, qui permet de détecter la présence de l'amer et une boussole (ce robot pourrait être un de ceux présentés dans la sous-section 2.3.2). Les équations d'état du robot peuvent être décrites de cette manière :

$$\begin{cases} \dot{x}_1 &= \cos x_3 + b_1 \\ \dot{x}_2 &= \sin x_3 + b_2 \\ \dot{x}_3 &= u + b_3 \\ \dot{x}_4 &= \omega + b_4. \end{cases} \tag{5.11}$$

où $u(t)$ correspond à la commande, (x_1, x_2) aux coordonnées du robot, x_3 son orientation et x_4 l'angle courant du télémètre laser (qui tourne à une vitesse de $\omega = 2\,\text{rad.s}^{-1}$). On suppose que x_3 et x_4 sont mesurés avec une précision de ± 0.01 rad tout au long de l'expérience, que l'état initial du robot est dans le pavé $[-2,2]\times[-7,-3]\times[-2,2]\times[-2,2]$ et que les bruits (liés aux incertitudes de modèle, aux perturbations...) $b_i(t)$ sont uniformément distribués dans l'intervalle $[-0.01, 0.01]$.

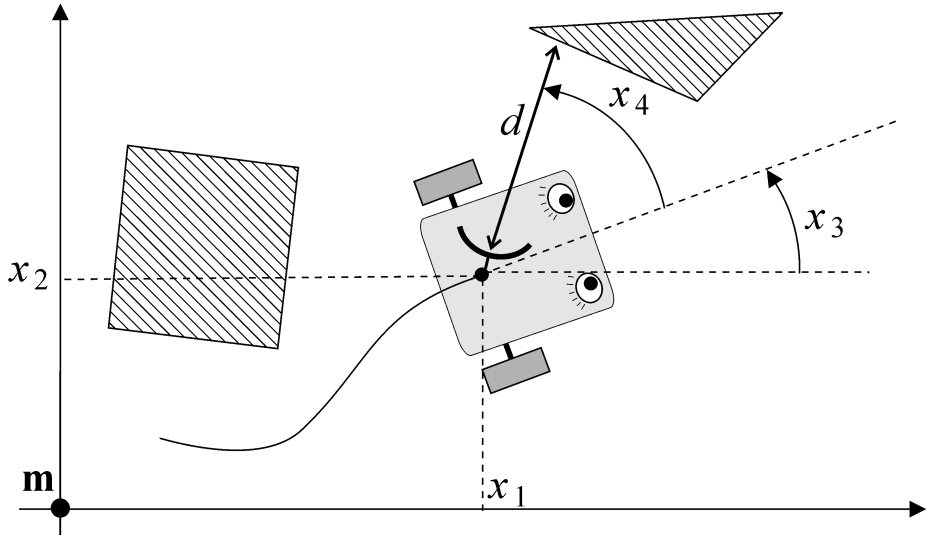


Figure 5.7 – Le robot est seulement équipé d'un télémètre laser pour se localiser et connait seulement la position de l'amer **m**. Le carré et le triangle peuvent être détectés par le télémètre mais leur présence est inconnue du robot à l'avance.

Le robot sait que l'amer **m** est aux coordonnées $(0, 0)$. La distance entre le robot et l'amer est mesurée avec une précision de ± 0.01 m, mais seulement lorsque le faisceau laser de son télémètre pointe exactement dans la direction de l'amer et qu'aucun obstacle ne les sépare. La portée du télémètre laser correspond à

l'intervalle $[s] = [s^-, s^+] = [1, 10]$ m. On peut donc définir les fonctions de visibilité h et d'observation g comme suit :

$$\left\{\begin{array}{rcl} h(\mathbf{x}) & = & \det\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} \cos(x_3+x_4) \\ \sin(x_3+x_4) \end{pmatrix}\right) \\ & = & x_1 \sin(x_3+x_4) - x_2 \cos(x_3+x_4) \\ g(\mathbf{x}) & = & \begin{pmatrix} \cos(x_3+x_4) \\ \sin(x_3+x_4) \end{pmatrix} . \begin{pmatrix} -x_1 \\ -x_2 \end{pmatrix} \\ & = & -x_1 \cos(x_3+x_4) - x_2 \sin(x_3+x_4) . \end{array}\right. \tag{5.12}$$

La fonction h est obtenue à partir du déterminant entre un vecteur qui a pour direction l'angle sous lequel le robot voit l'amer ($(x_1 - 0, x_2 - 0)^{\mathrm{T}}$) et un vecteur qui a pour direction le faisceau laser (par exemple le vecteur unitaire $(\cos(x_3+x_4), \sin(x_3+x_4))^{\mathrm{T}}$) pour qu'elle s'annule lorsque le faisceau laser pointe vers l'amer. La fonction g est obtenue à partir du produit scalaire entre ces mêmes vecteurs, pour obtenir la distance entre le robot et l'amer lorsque le faisceau laser pointe vers l'amer. Notons $d(t)$ la distance du robot au premier obstacle en suivant la direction pointée par le laser. Si le premier obstacle est dans la portée du télémètre, il retourne un intervalle de distance $[d] = [d^-, d^+]$ qui contient d. L'amer $\mathbf{m}$ est vu par le télémètre si *(i)* la ligne correspondant au faisceau laser du télémètre contient $\mathbf{m}$ (*i.e.* $h(\mathbf{x}) = 0$), *(ii)* la distance entre le robot et l'amer est dans l'intervalle correspondant à la portée du télémètre ($g(\mathbf{x}) \in [s] = [s^-, s^+]$) et *(iii)* il n'y a aucun objet entre le robot et l'amer ($\neg(d < g(\mathbf{x}))$). On a donc la relation

$$(h(\mathbf{x}) = 0 \text{ et } g(\mathbf{x}) \in [s] \ \cap [-\infty, d]) \Rightarrow d = g(\mathbf{x}) . \tag{5.13}$$

En utilisant les équivalences logiques

$$\begin{array}{rcl} (A \wedge B \Rightarrow C) & \Leftrightarrow & (\neg(A \wedge B) \vee C) \\ & \Leftrightarrow & (\neg A \vee \neg B \vee C) \\ & \Leftrightarrow & (A \Rightarrow \neg B \vee C) , \end{array} \tag{5.14}$$

l'implication 5.13 devient :

$$(h(\mathbf{x}) = 0) \Rightarrow (g(\mathbf{x}) \notin ([s] \cap [-\infty, d]) \text{ ou } d = g(\mathbf{x})) . \tag{5.15}$$

De plus,

$$\begin{array}{rl} & g(\mathbf{x}) \notin ([s] \cap [-\infty, d]) \\ \Leftrightarrow & g(\mathbf{x}) \in (\mathrm{Comp}([s] \cap [-\infty, d])) \\ \Leftrightarrow & g(\mathbf{x}) \in (\mathrm{Comp}([s]) \cup \mathrm{Comp}([-\infty, d])) \\ \Leftrightarrow & g(\mathbf{x}) \in [-\infty, s^-] \cup [s^+, \infty] \cup [d, \infty] \end{array} \tag{5.16}$$

(où Comp($\mathbb{A}$) est l'ensemble complémentaire de $\mathbb{A}$ dans $\mathbb{R}$), donc

$$\begin{array}{rcl} (h(\mathbf{x}) = 0) & \Rightarrow & (g(\mathbf{x}) \in [-\infty, s^-] \cup [s^+, \infty] \cup [d, \infty] \text{ ou } d = g(\mathbf{x})) \\ & \Rightarrow & (g(\mathbf{x}) \in [-\infty, s^-] \cup [s^+, \infty] \cup [d, \infty] \text{ ou } g(\mathbf{x}) \in \{d\}) \\ & \Rightarrow & (g(\mathbf{x}) \in [-\infty, s^-] \cup [s^+, \infty] \cup [d, \infty] \cup \{d\}) \\ & \Rightarrow & (g(\mathbf{x}) \in [-\infty, s^-] \cup [s^+, \infty] \cup [d, \infty]) \end{array} \tag{5.17}$$

Comme $d^- < d$, on peut écrire

$$h(\mathbf{x}(t)) = 0 \Rightarrow g(\mathbf{x}(t)) \in \mathbb{W}(t) \tag{5.18}$$

avec

$$\mathbb{W}(t) = \left[-\infty, s^-\right] \cup \left[s^+, \infty\right] \cup \left[d^-(t), \infty\right]. \tag{5.19}$$

Notre problème de localisation dynamique est donc est problème d'estimation d'état avec données fugaces (voir équations 5.1).

Génération des données. La commande du robot est constante avec $u(t) = 0.2$ (de façon à ce que le robot fasse un simple cercle), l'état initial est $\mathbf{x}_0 = (0, -5, 0, 0)^{\mathrm{T}}$, le pas de temps est $\delta = 0.02\,\mathrm{s}$ et $t \in [0, 40]$ s (dans la simulation, les équations discrétisées avec Euler sont utilisées, voir les équations 1.2, 3.11 et 3.12). L'environnement du robot est une pièce presque carrée où se trouvent des obstacles fixes et mobiles. L'amer $\mathbf{m}$ est représenté par un petit carré noir, qui se trouve au bout d'un obstacle triangulaire, au centre de la pièce. La figure 5.8 montre des captures d'écran de la mission effectuée par le robot aux instants $t \in \{0, 5, \ldots, 35\}$ s. A $t = 35\,\mathrm{s}$ par exemple, le premier obstacle est en-dehors de la portée du télémètre.

La figure 5.9 représente $\mathbb{W}(t)$ avec $t \in [0, 40]$, $d \in [1, 10]$ (voir la figure 5.4 pour l'explication du principe d'obtention d'une waterfall à partir de télémètres lasers rotatifs). Les cercles correspondent aux points fugaces $(t, g(\mathbf{x}(t)))$. Pour tous ces points, on a $(t, g(\mathbf{x}(t))) \in \mathbb{W}(t)$. Lorsqu'un cercle correspond à un point de l'extrémité gauche de $\mathbb{W}$, l'amer a été vu par le télémètre. Vu que le robot ne connait rien de son environnement à part la position de l'amer $\mathbf{m}$, lorsque le télémètre renvoie un intervalle de distance dans l'intervalle $[d] = [4.99, 5.01]$ m, le robot traduit cette information par "l'amer $\mathbf{m}$ n'est pas dans la partie du faisceau laser située dans l'intervalle de distance au robot $[1, 4.99]$ m". En fait, le robot mesure l'absence d'amer dans une partie du faisceau laser plutôt que sa distance à l'amer.

Enveloppe de la trajectoire. La figure 5.10 (gauche) montre l'enveloppe de la trajectoire (pavés gris) après une simple propagation intervalle sans prendre en compte les données du télémètre. La taille du pavé initial est de 4 m et elle est de 6 m pour le pavé final. Après une propagation intervalle prenant en compte les données du télémètre, on obtient la trajectoire dessinée sur la figure 5.10 (droite) et la précision de la localisation est alors de moins d'1 m pour tout t. Les repères sont de $[-14, 14]^2$ et la trajectoire réelle est le cercle noir.

Détail de l'effet du contracteur de visibilité sur $y(t) = g(\mathbf{x}(t))$. La figure.5.11 montre une partie (redimensionnée) de la waterfall produite au cours des traitements, au niveau d'une zone fugace. Les zones bleues claires sont les endroits où le télémètre n'a rien vu et où on est donc sûr qu'il n'y a pas l'amer. A l'inverse, on ne peut rien dire des zones noires. Les zones entourées de bleu, vert ou rouge sont le dessin du tube $[y](t)$ (le temps est sur les ordonnées, la distance au robot est en abscisses) : lorsque l'amer est avant le faisceau du télémètre laser, $[v](t) = [h](\mathbf{x}(t)) < 0$ (zone entourée de bleu), lorsqu'il est après, $[v](t) > 0$ (zone entourée de rouge) et lorsqu'il est possible que l'amer ait été vu, $0 \in [v](t)$ (zone entourée de vert). Cette figure correspond à l'étape 1 de la méthode de contraction de $[y](t)$ par rapport à la contrainte de visibilité : on prend 2 tranches $[v](k_1)$ et $[v](k_2)$ (les instants k_1 et k_2 sont symbolisés par les 2 traits blancs horizontaux) telles que $0 \in [v](k)$ pour tout $k \in [k_1 + 1, k_2 - 1]$ et telles que $[v](k_1)$ et $[v](k_2)$ soient de signes opposés.

Les figures.5.12, 5.13 et 5.14 présentent différents cas montrant ce qui se passe à l'étape 2 : pour chaque $k \in [k_1 + 1, k_2 - 1]$ (représenté par le trait horizontal orange), on suppose temporairement que ce k est un point fugace (car on ne sait pas où se trouve le point fugace dans $[k_1 + 1, k_2 - 1]$) en calculant $(\mathbb{W}(k) \cap [y](k))$. S'il est vide, ce ne peut pas être un point fugace, comme dans la figure 5.13 : on voit clairement que pour le

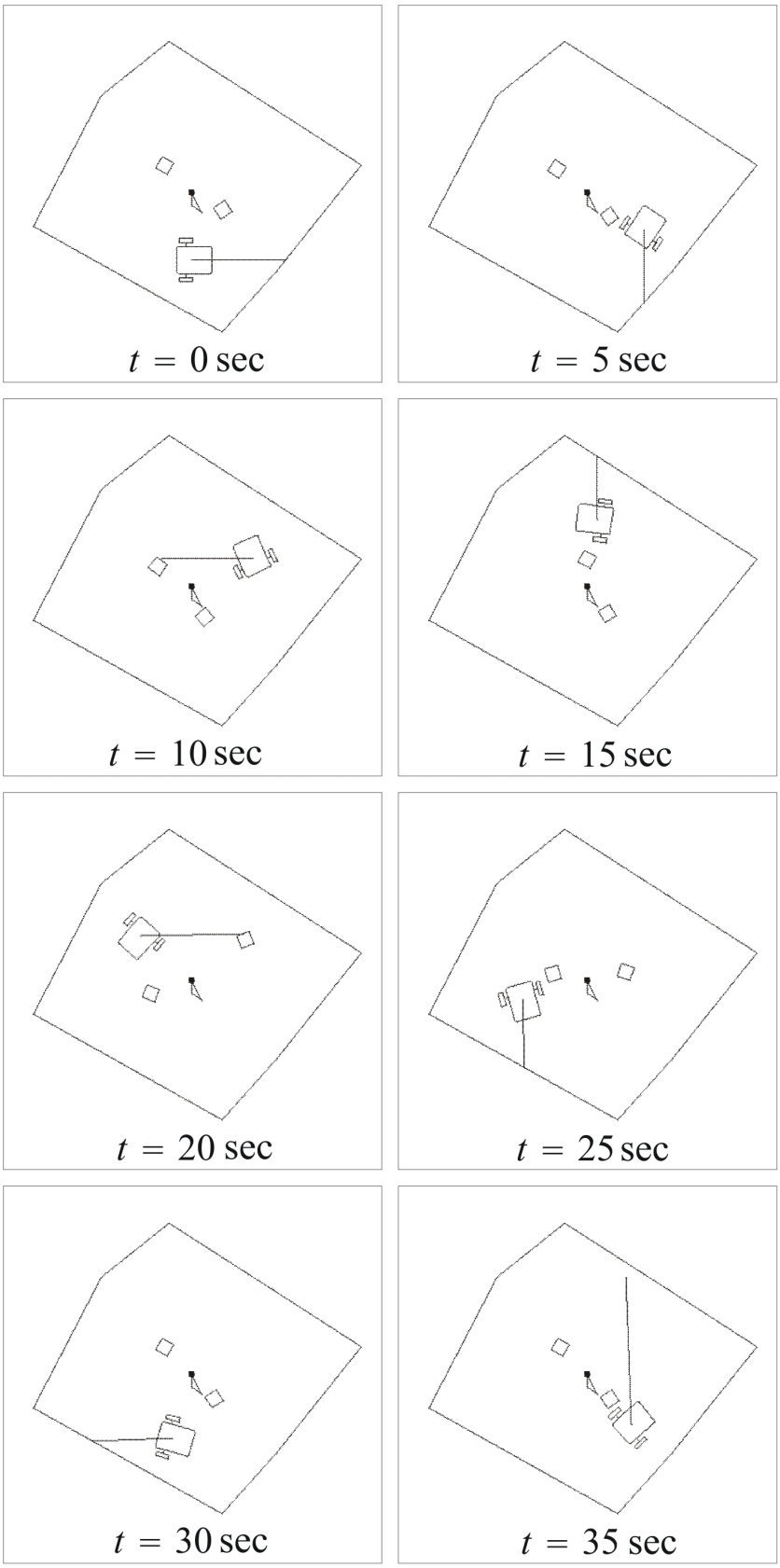


Figure 5.8 – Le robot à différents instants de la simulation.

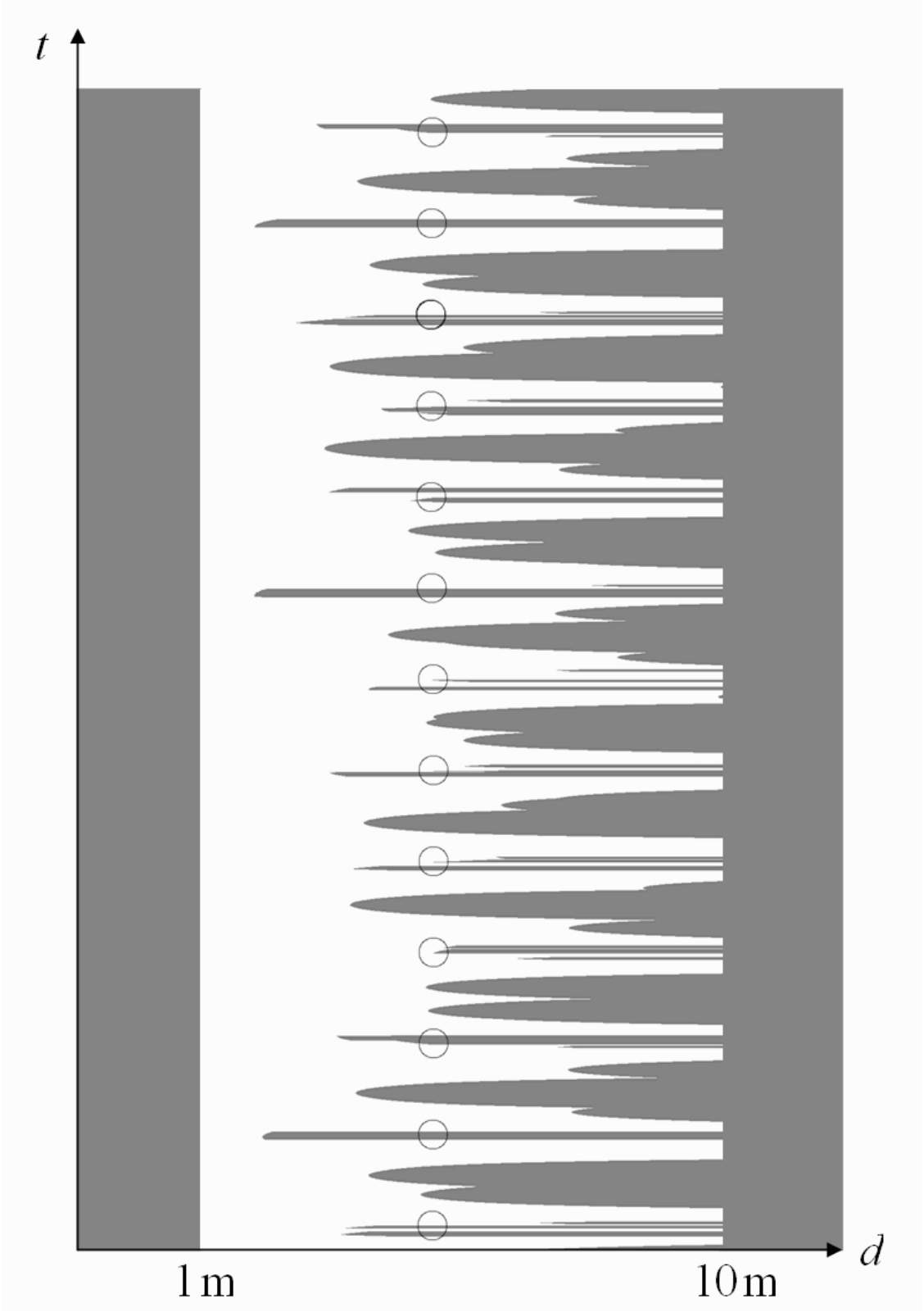


Figure 5.9 – Le robot sait seulement que les points fugaces $(t, g(\mathbf{x}(t))$ sont toujours dans la waterfall $\mathbb{W}(t)$ (en gris) ou (de manière équivalente) qu'aucun point fugace n'est dans la zone blanche.

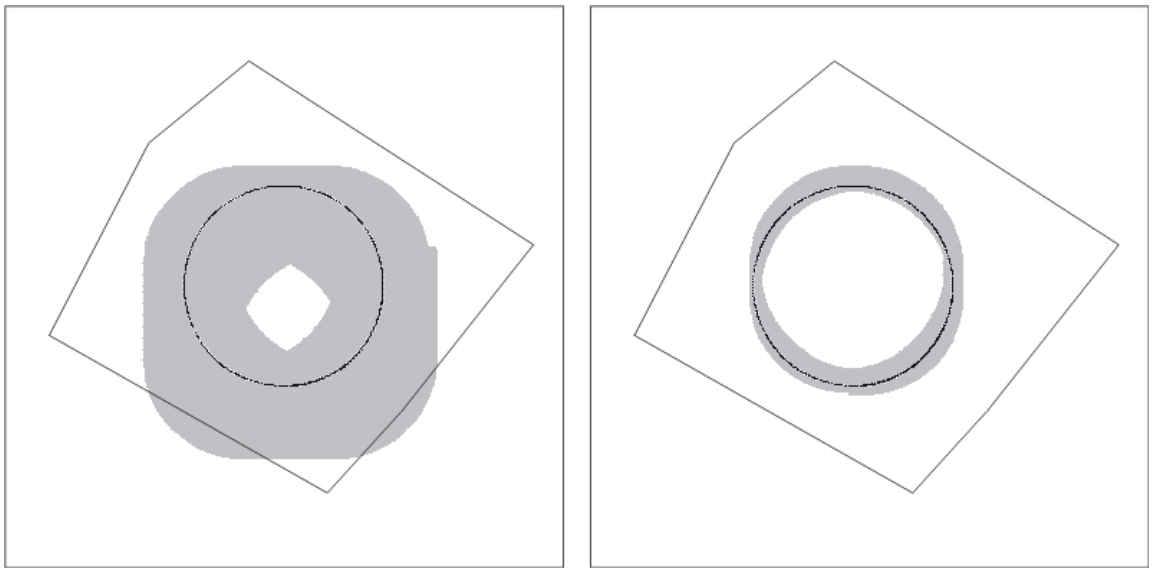


Figure 5.10 – Enveloppe obtenue après une propagation intervalle sans utiliser les données du télémètre (à gauche) ; Enveloppe obtenue après une propagation intervalle en prenant en compte les données du télémètre (à droite). Le trait continu en gris moyen est la forme de la pièce dans laquelle le robot évolue.

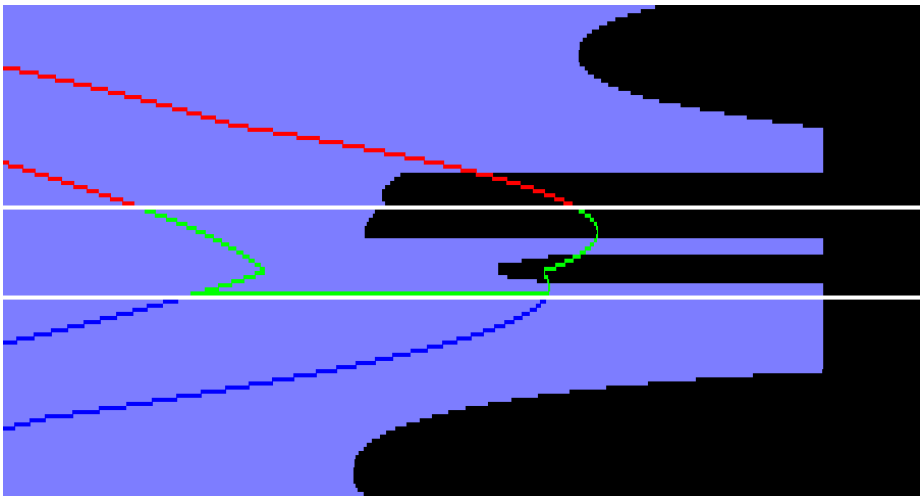


Figure 5.11 – Partie de la waterfall (zone noire) montrant une zone fugace (zone entourée de vert et délimitée par les 2 traits horizontaux blancs), à l'étape 1 de la contraction de $[y](t)$ par rapport à la contrainte de visibilité. Le temps est en ordonnées et la distance au robot en abscisses.

k sélectionné (symbolisé par le trait orange), l'intervalle $[y](k)$ (dans la zone entourée de vert) n'intersecte pas $\mathbb{W}(k)$ (la zone noire). S'il est non-vide comme dans les figures.5.12 et 5.14, on calcule (en jaune) ce que serait le sous-tube $[y](t)$ dans la zone sélectionnée si k était un point fugace. Une fois qu'on a testé tous

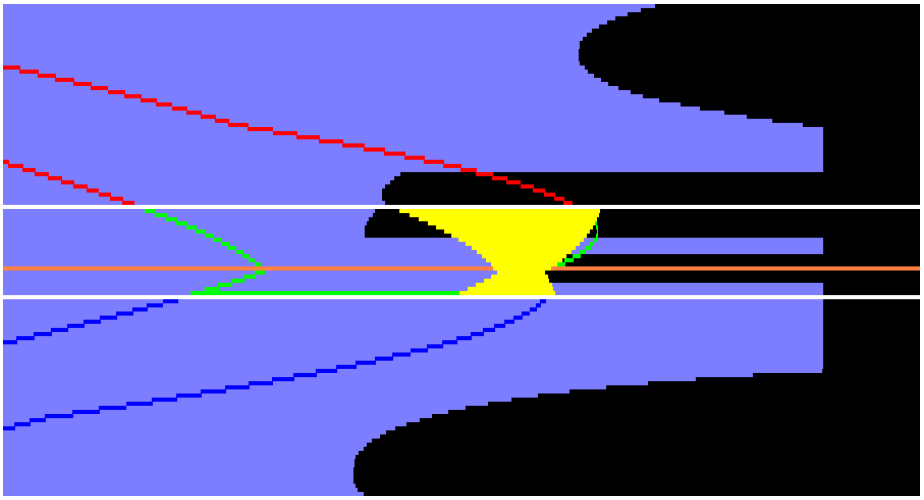


Figure 5.12 – Sous-tube $[y](t)$ obtenu en supposant qu'un point fugace se trouve à l'instant indiqué par le trait horizontal orange.

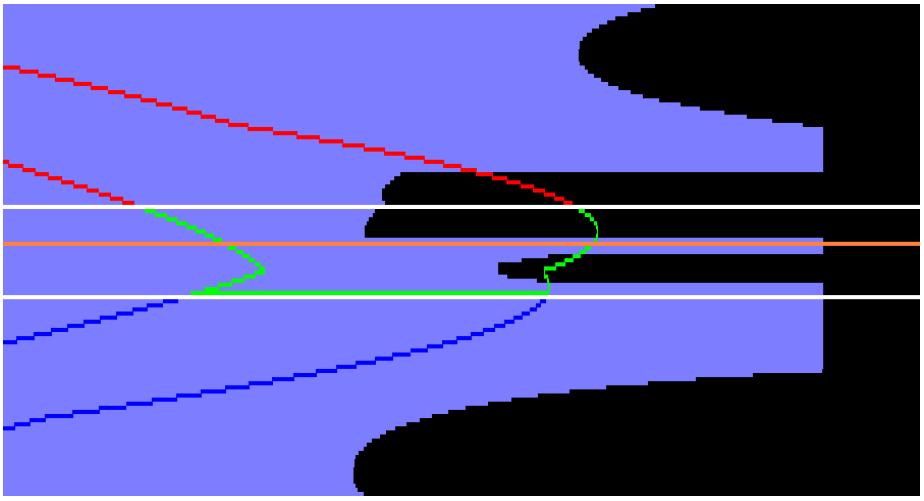


Figure 5.13 – Il ne peut pas avoir de point fugace à l'instant indiqué par le trait horizontal orange car la zone fugace connue jusqu'à maintenant (entourée de vert) n'intersecte pas la waterfall (zone noire).

les k, il n'y a plus qu'à prendre l'union des sous-tubes temporaires $[y](t)$ trouvés pour chaque supposition (car parmi toutes les suppositions qu'on a faites, il y a le point fugace, mais on ne sait toujours pas lequel c'est exactement), puis l'intersecter avec le tube $[y](t)$ d'origine (étape 3) pour contracter $[y](t)$. La figure 5.15 montre le résultat de l'union et intersection (en jaune) par rapport au tube initial $[y](t)$ (entouré de vert),.pour $k \in [k_1 + 1, k_2 - 1]$.

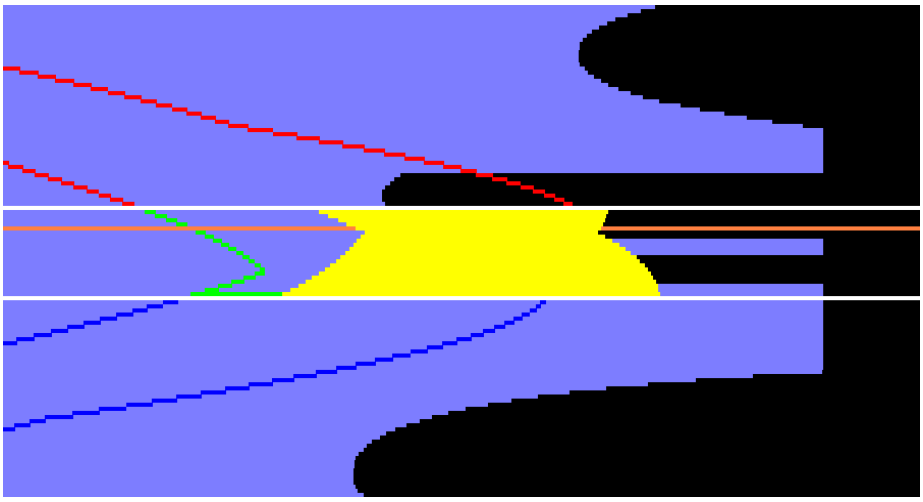


Figure 5.14 – Sous-tube $[y]\,(t)$ obtenu en supposant qu'un point fugace se trouve à un autre instant, indiqué par le trait horizontal orange.

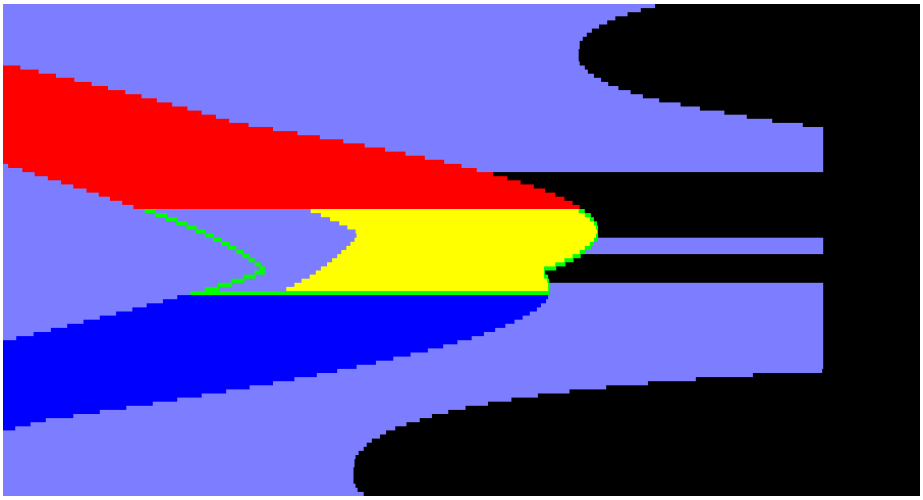


Figure 5.15 – Résultat de l'étape 3 de la contraction de $[y]\,(t)$ par rapport à la condition de visibilité. La zone entourée de vert est la zone fugace initiale et la zone jaune est le résultat de la contraction.

Construction de la carte de l'environnement. En notant $[\mathbf{x}](t)$ les pavés obtenus à la fin de la propagation intervalle et $d(t)$ la distance retournée par le télémètre, on peut obtenir une approximation de l'ensemble

$$\begin{array}{rcll} \mathbb{M} & = & \{ & (z_1, z_2), \exists t \in [0,40], \exists \mathbf{x} \in [\mathbf{x}](t) \\ & & & z_1 = x_1(t) + d(t).\cos(x_3(t) + x_4(t)), \\ & & & z_2 = x_2(t) + d(t).\sin(x_3(t) + x_4(t)) \quad \} \end{array} \tag{5.20}$$

en faisant un simple calcul d'intervalles. L'ensemble calculé est donc celui-ci :

$$\begin{array}{ll} \{ & ([z_1],[z_2]), \exists t \in [0,40], \exists \mathbf{x} \in [\mathbf{x}](t) \\ & [z_1] = [x_1(t)] + [d(t)].[\cos([x_3(t)] + [x_4(t)])], \\ & [z_2] = [x_2(t)] + [d(t)].[\sin([x_3(t)] + [x_4(t)])] \quad \}. \end{array}$$

On obtient l'ensemble des pavés dessinés sur la figure 5.16 (gauche). Cette figure donne une approximation englobant la carte de l'environnement entourant le robot lors de sa mission. La figure 5.16 (droite) représente le centre de tous les pavés.

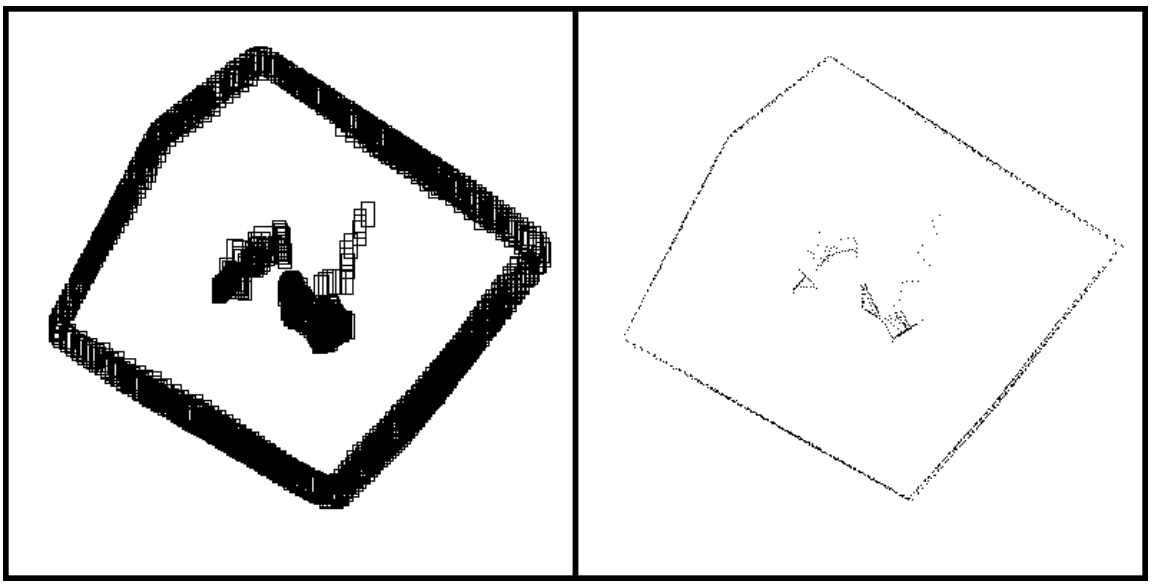


Figure 5.16 – Ensemble des pavés dont l'union contient la carte de l'environnement (à gauche) ; Approximation de la carte en dessinant le centre de ces pavés (à droite).

5.3 Extension au SLAM sous-marin

On peut essayer de formaliser le problème du SLAM offline sous-marin avec données fugaces et sans données aberrantes ([Le Bars et al., 2011b]) en reprenant et modifiant les équations 4.1, 4.2, 4.4, 4.15 et 5.1, 5.3 de

cette manière :

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) & \text{(équation d'évolution)} \\ \mathbf{y} = \mathbf{g}(\mathbf{x}) & \text{(équation des mesures indépendantes de la waterfall)} \\ \mathbf{z}_i = \mathbf{h}(\mathbf{x}, \mathbf{u}, \mathbf{m}_i) & \text{(équation des détections d'amers)} \\ v_i(\mathbf{x}, \mathbf{u}, \mathbf{m}_i) = 0 \Rightarrow \mathbf{z}_i \in \mathbb{W} & \text{(condition de visibilité des amers sur la waterfall).} \end{cases} \tag{5.21}$$

$\mathbf{x}$ est le vecteur d'état du robot à estimer, $\mathbf{f}$ sa fonction d'évolution, $\mathbf{m}_i$ la position de l'amer i à estimer, $\mathbf{u}$ le vecteur des entrées, $\mathbf{y}$ le vecteur des mesures indépendantes de la waterfall $\mathbb{W}$, $\mathbf{g}$ la fonction d'observation correspondante, $\mathbf{z}_i$ est le vecteur des mesures liées aux détections de l'amer i, $\mathbf{h}$ la fonction correspondante, v_i la fonction de visibilité de l'amer i sur la waterfall $\mathbb{W}$, $\mathbb{W}$ une fonction qui à un temps $t \in \mathbb{R}$ associe un sous-ensemble $\mathbb{W}(t) \in \mathbb{R}^2$.

Si on essaye d'appliquer ce formalisme au cas de la *Daurade* et du *Redermor*, on a pour équation d'évolution :

$$\dot{\mathbf{p}} = \mathbf{R}(\varphi, \theta, \psi).\mathbf{v}_r \tag{5.22}$$

et pour équation d'observation :

$$\mathbf{y} = \mathbf{p} \tag{5.23}$$

avec $\mathbf{x} = \mathbf{p}$ et $\mathbf{u} = (\varphi, \theta, \psi, v_r^x, v_r^y, v_r^z)$ (voir les sous-sections 2.2.1 et 4.3.2). Les détections de l'amer i sont définies par :

$$r_i = ||\mathbf{e}_i||$$

et

$$a = z_{mi} - z$$

avec r_i la distance robot-amer mesurée sur l'image sonar et a l'altitude du robot mesurée par le DVL. On a donc $\mathbf{z}_i = (r_i, a)$. Ces relations sont significatives lorsque l'amer i passe dans le champ de vision du sonar latéral, ce qui se produit lorsqu'on a la condition suivante :

$$\left(\mathbf{R}^{\mathrm{T}}.\mathbf{e}_i\right).\mathbf{i}_r = 0$$

Il faut noter qu'ici, la waterfall $\mathbb{W}$ a 2 dimensions : une représentant l'image sonar et l'autre représentant l'altitude du sous-marin. Les figures 5.17 et 5.18 illustrent la condition de visibilité d'un amer sur une partie de waterfall obtenue à partir d'une image sonar.

Nous pouvons résoudre par une propagation de contraintes (suivie éventuellement de bissections) le problème de SLAM fugitif défini. Les étapes de résolution et les contracteurs utilisés pourront être les suivants :

Initialisation. Tous les tubes sont initialisés avec les valeurs déjà connues. Par exemple, si on a une condition initiale sur le vecteur d'état, le tube correspondant $[\mathbf{x}](t)$ sera contracté à $t = 0$. La waterfall $\mathbb{W}(t)$ est supposée connue pour tout t (on suppose qu'on est dans le cas d'un problème d'estimation d'état offline). De plus on peut supposer qu'un opérateur humain a détecté au moins 1 fois chaque amer (pour obtenir le nombre d'amers différents et résoudre le problème d'association des données).

Contracteur d'évolution. Une propagation *forward-backward* par rapport au temps combinée avec une propagation *forward-backward* sur l'expression pour chaque pas de temps sera efficace (voir sous-section 3.4.1) pour contracter les tubes $[\mathbf{x}](t), [\dot{\mathbf{x}}](t)$ par rapport à la contrainte $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$.

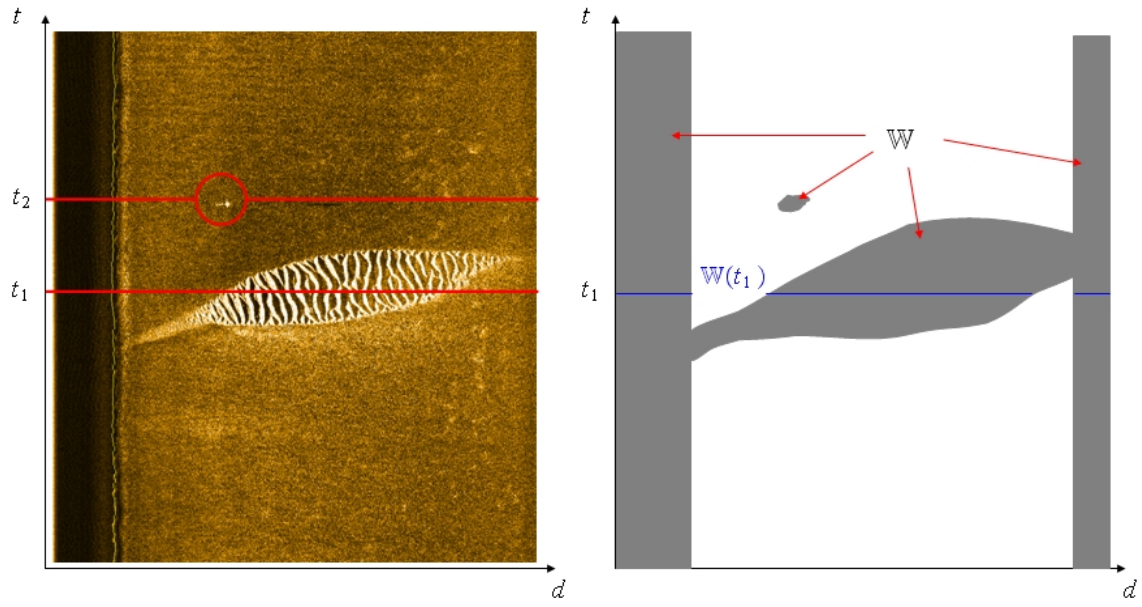


Figure 5.17 – Construction de la waterfall $\mathbb{W}$ sans la dimension liée à l'altitude (à droite) à partir d'une image de sonar latéral (à gauche). Elle est formée à partir des zones brillantes de l'image sonar. Il n'y a pas d'amers dans la zone blanche, mais il y en a peut être dans la zone grise.

Contracteurs d'observation et de détections. Les contractions des tubes $[\mathbf{x}](t), [\mathbf{y}](t), [\mathbf{z}_i](t)$ et pavés $[\mathbf{m}_i]$ par rapport aux contraintes $\mathbf{y}(t) = g(\mathbf{x}(t))$ et $\mathbf{z}_i(t) = \mathbf{h}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{m}_i)$ peuvent être faites avec HC4Revise.

Contracteur de visibilité. La relation $v_i(\mathbf{x}(t), \mathbf{u}(t), \mathbf{m}_i) = 0 \Rightarrow \mathbf{z}_i(t) \in \mathbb{W}(t)$ peut contracter les tubes $[\mathbf{x}](t)$, $[\mathbf{z}_i](t)$ et pavés $[\mathbf{m}_i]$. La contraction de $[\mathbf{z}_i](t)$ est faite en utilisant le théorème 1 et $[\mathbf{x}](t)$, $[\mathbf{m}_i]$ par le théorème 2 (en faisant éventuellement une décomposition similaire à 5.2, voir l'algorithme décrit en 5.2.2).

Comme pour tout algorithme de propagation de contraintes par intervalles, il faut répéter les contractions jusqu'à obtention d'un point fixe. La figure 5.19 illustre schématiquement le processus de contration et propagation sous forme de graphe des contraintes.

Les résultats présentés en 4.3.2 ont déjà montré l'intérêt du formalisme présenté, sauf que dans ce cas la partie "détection des amers sur la waterfall" était faite entièrement par un opérateur humain, au lieu d'un simple seuillage comme dans l'exemple de la section précédente. De plus, le côté fugace était exprimé (voir figure 4.8) par "le fait qu'à un moment donné l'amer était plutôt devant le sous-marin et qu'à un autre moment il se trouve plutôt derrière nous permet (par le théorème des valeurs intermédiaires) de déterminer qu'à un instant t, le sonar du robot aurait dû voir l'amer". Ici, on peut supposer que l'opérateur humain détecte une seule fois chaque amer. En prenant en compte les données sonar de manière automatique, l'algorithme devrait permettre d'améliorer encore plus qu'en 4.3.2 les estimations de positions, tout en minimisant le travail de l'opérateur humain.

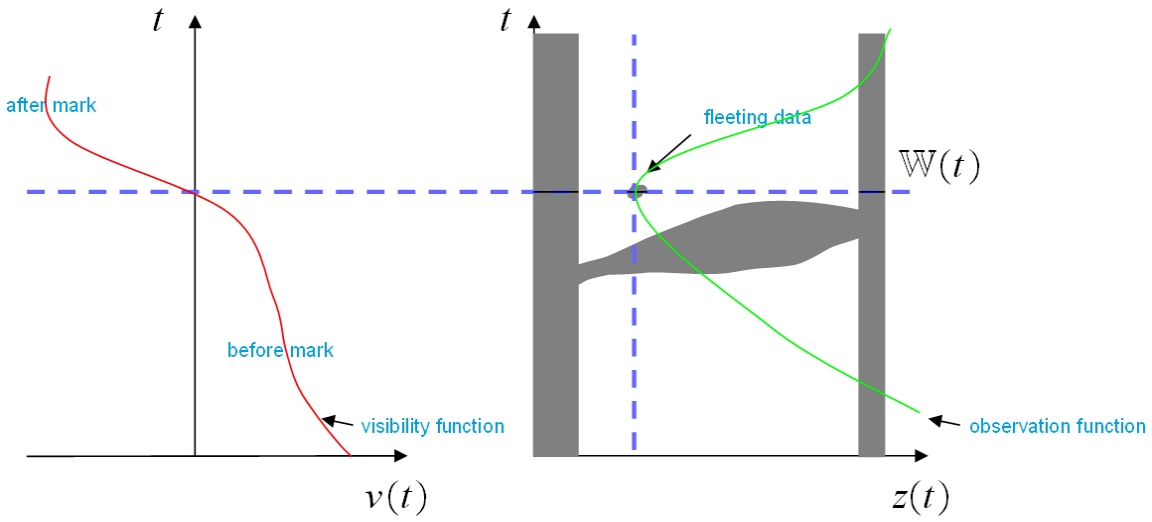


Figure 5.18 – Illustration des fonctions de visibilité et d'observation en SLAM sous-marin. La fonction de visibilité d'un amer est calculée à partir des équations d'état, des données de navigation du sous-marin et d'au minimum une détection de l'amer. Elle modélise le passage de l'amer en face du sonar du robot. Elle peut être définie simplement par l'angle de vue de l'amer par rapport au sonar du sous-marin. On ne peut pas passer par une situation où l'amer est devant puis derrière sans avoir un moment où l'amer tombe en face du sonar du robot. La fonction d'observation d'un amer est calculée à partir des équations d'état, des données de navigation du sous-marin et d'au minimum une détection de l'amer. Elle peut être définie par la distance robot-amer. L'image sonar, à travers la waterfall et la fonction de visibilité vont permettre d'améliorer la connaissance de la fonction d'observation. Inversement, la waterfall et la fonction d'observation vont permettre d'améliorer la connaissance de la fonction de visibilité. L'amélioration de la connaissance de ces 2 fonctions aura pour conséquence d'améliorer l'estimation de la trajectoire du robot et de la position des amers (souvent des mines dans le cadre de la thèse). De plus, lorsqu'elles seront dessinées sur l'image sonar, ces fonctions vont aider l'opérateur humain dans sa recherche de mines.

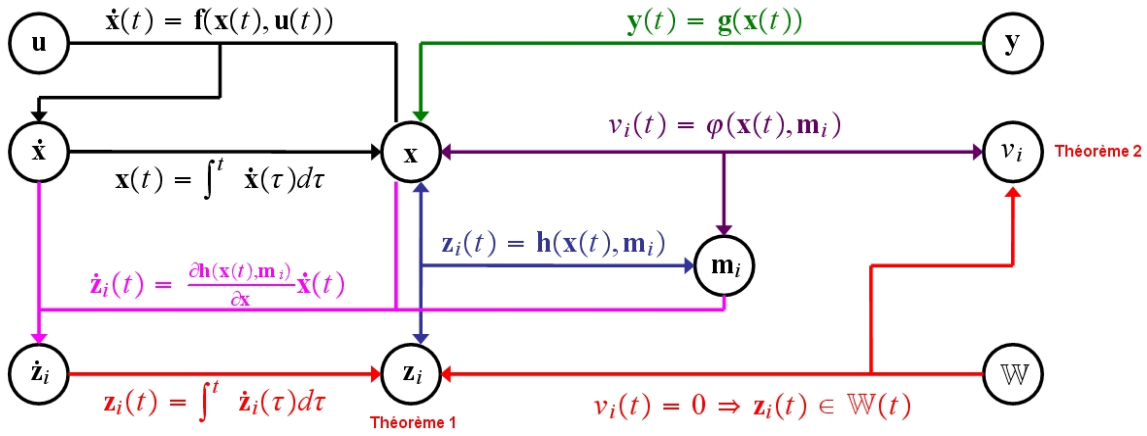


Figure 5.19 – Illustration sous forme de graphe des contraintes du processus de contraction et propagation pour un problème de SLAM avec données fugaces. Ce graphe est formé d'hyperarcs orientés vers les variables qui peuvent être contractées (quand on le sait).

5.4 Conclusion du chapitre

Le problème des données fugaces a été introduit dans ce chapitre. Cette nouvelle façon de formaliser les problèmes mettant en jeu des robots munis de sonar, télémètres ou caméras a ensuite été résolue en utilisant l'approche des tubes (intervalles de trajectoires). L'algorithme résultant a été testé sur une simulation de robot char muni d'un télémètre laser se localisant grâce à un amer connu dans un environnement avec des obstacles mobiles inconnus. Enfin, son application possible au SLAM sous-marin a été expliquée.

Dans ce chapitre, la motivation qui a conduit à considérer le problème des données fugaces a d'abord été indiquée. En effet, dans le SLAM sous-marin étudié jusqu'à maintenant, on ne prenait pas en considération directement les données sonar, seules des détections de mines faites par un opérateur humain sur les images sonar étaient utilisées. La difficulté de ces détections vient notamment du fait que les mines ne sont vues par le sonar que pendant de cours instants, qu'il faut trouver parmi une quantité très importante de données a priori inutiles dans l'image sonar. Pourtant, il semblait intéressant d'essayer de mettre en cohérence les données de navigation et les images sonar pour au moins aider l'opérateur humain dans sa phase de détection (indépendamment du développement d'un algorithme de traitement dédié à la détection de mines sur images sonar). Une formalisation définissant ce qu'est une donnée fugace et le concept de waterfall a donc été introduite. Le fait que le robot voit un amer dans le champ de l'un de ses capteurs a alors été écrit sous la forme d'une condition de visibilité. Une méthode permettant d'utiliser cette condition pour améliorer l'estimation de la trajectoire d'un robot (exemple de la localisation d'un char muni d'un télémètre laser rotatif) et même la position des amers (SLAM sous-marin) a alors été décrite. Ces formalisations et méthodes de prise en compte correspondantes ne semblent pas avoir été déjà définies jusqu'à maintenant.

Chapitre 6

Conclusion et perspectives

Dans ce document, les différents robots utilisés au cours de la thèse ont d'abord été décrits dans le premier chapitre et un parallèle entre les robots terrestres et sous-marins a été fait notamment pour justifier les passages d'un type à l'autre dans les chapitres suivants. La notion du caractère fugace des détections d'amers (qui est nouvelle) a aussi été indirectement introduite avec l'explication du fonctionnement de leurs capteurs (notamment sonars et télémètres) et une première formalisation de leurs équations d'évolution, observation et détection. Tous les robots décrits ont été réutilisés dans divers exemples afin d'illustrer le côté applicatif de la thèse.

Ensuite, les principaux outils théoriques au centre de la thèse ont été présentés dans le deuxième chapitre :

- Le principe du calcul par intervalles.
- Ses extensions aux vecteurs d'intervalles (pavés) et intervalles de trajectoires (tubes), très utiles en robotique. Les tubes, tels qu'ils ont été définis, sont une nouveauté.
- Les techniques de calcul et algorithmes classiques principaux pour résoudre les problèmes mettant en jeu des intervalles.

Le principe du SLAM, les méthodes existantes pour le traiter et le SLAM par intervalles appliqué sur des données réelles de missions de sous-marins ont été décrits dans le chapitre suivant, avec une comparaison entre différentes méthodes de résolution. Ceci a permis de conclure que le SLAM par propagation par intervalles a un intérêt réel de par son bon rapport rapidité de calcul, précision, consistance et réutilisation simple sans adaptations particulières pour différents types de robots.

Enfin, la problématique des données fugaces rencontrée avec les données du sonar dans le cas du SLAM sous-marin a été formalisée et un algorithme de résolution par intervalles a été proposé et appliqué sur un exemple de localisation de robot terrestre. Son extension au SLAM sous-marin a ensuite été présentée.

Plusieurs améliorations et perspectives seraient à étudier et à combiner dans la suite :

- Améliorer encore le côté opérationnel de GESMI pour le SLAM sous-marin (meilleure affichage des données sonar, meilleure automatisation...).
- Appliquer/étendre l'algorithme de traitement des données fugaces à d'autres types d'expériences (robot avec caméra, radar...) et sur du SLAM sans aucune intervention d'opérateur humain.

- Faire du SLAM online temps réel par intervalles. A ma connaissance, ceci n'a pas encore été fait. Le principe pourrait être de résoudre l'équivalent d'un petit SLAM offline sur les n dernières données de la mission.
- Faire du SLAM sans connaître l'association des données liées aux détections. Ceci a déjà été fait avec les données du *Redermor* et par propagation intervalle, voir [Chabert et al., 2009].
- Etre robuste par rapport aux données aberrantes. En pratique, lorsqu'il y a des données aberrantes, les méthodes présentées retournent en général un ensemble vide de solutions, ce qui empêche d'obtenir une estimation de la position du robot ou des amers. Pour obtenir quand même une estimation, il est possible de maximiser le nombre de contraintes à satisfaire (voir [Lahanier et al., 1987]). Ceci peut être fait avec des propagations de contraintes par intervalles comme celles présentées dans [Jaulin and Walter, 2002], [Jaulin, 2009b], [Sliwka et al., 2009] avec la q-intersection, ou [Sliwka et al., 2011b], [Sliwka, 2011] avec le contracteur sur l'image et les polynômes ensemblistes, testés sur notre robot sous-marin autonome *SAUC'ISSE*.
- Gérer des amers non ponctuels et éventuellement mobiles (voir *e.g.* [Wang et al., 2007]). Ceci pourrait être fait en relation avec le point suivant.
- Utiliser une meute de robots sous-marins, gérer les données de plusieurs expériences en même temps. Par exemple, si on a plusieurs robots communiquant ensemble, avec différents types de capteurs, le problème du SLAM peut être transformé en un grand CSP qui pourrait être géré avec les méthodes présentées dans ce document. De plus il serait aussi possible de mélanger de la même façon les données de plusieurs expériences faites par un même sous-marin en concaténant les contraintes liées à chaque expérience pour former un seul CSP (voir [Di Marco et al., 2003] pour des travaux dans ce sens). Des travaux de thèses sur les meutes de robots sont actuellement en cours (Aymeric BETHENCOURT, Clément AUBRY, Thomas SOUSSELIER, Nicolas CARLESI, Thibault GATEAU). Dans la même idée, il serait intéressant d'étudier la parallélisation des algorithmes de SLAM présentés.
- Mélanger des méthodes probabilistes (filtre de Kalman...) avec des méthodes par intervalles. En effet, le SLAM par intervalles est efficace quand on ne peut pas faire d'approximations (telles que des linéarisations) sur l'ensemble de la trajectoire d'un robot. Cependant, les positions trouvées par la méthode intervalle peuvent parfois être très pessimistes (pavés solutions trop grands). Ainsi, lancer une méthode de type Kalman par exemple sur chaque pavé solution (contenant la position du robot et des amers pour chaque pas de temps) retourné par la méthode intervalle pourrait être intéressant car il devrait être plus facile de faire des suppositions ou approximations (nécessaires pour que le filtre de Kalman fonctionne correctement) sur une petite partie de l'expérience et obtenir des estimations plus précises (la propagation par intervalles est une bonne méthode globale, le filtre de Kalman est une bonne méthode locale).
- Créer un programme de SLAM par intervalles plus général que GESMI ou le programme qui a été fait pour la comparaison de méthodes de SLAM pour rendre possible le traitement de ce problème à des non-initiés.

Articles, conférences et rapports

Articles de revue

F. Le Bars, J. Sliwka, and M. Sebela. Autonomous submarine robotic system. *Cybernetic Letters*, http://www.cybletter.com, 2010.

F. Le Bars, J. Sliwka, O. Reynet, and L. Jaulin. Set-membership state estimation with fleeting data. Accepted by *Automatica*, 2011.

J. Sliwka, F. Le Bars, and L. Jaulin. Student autonomous underwater robotics at ENSIETA. Accepted by *IJME (International Journal of Maritime Engineering)*, 2011.

Conférences avec comité de sélection

F. Le Bars, J. Sliwka, and L. Jaulin. Analyse par intervalles pour le lancé de rayon et pour l'analyse de stabilité. In *JD-JN-MACS 2009*, Angers, France, 2009.

J. Sliwka, F. Le Bars, and L. Jaulin. Calcul ensembliste pour la localisation et la cartographie robustes. In *JD-JN-MACS 2009*, Angers, France, 2009.

F. Le Bars, A. Bertholom, J. Sliwka, and L. Jaulin. Interval SLAM for underwater robots - a new experiment. In *NOLCOS 2010*, Bologna, Italy, 2010.

J. Sliwka, F. Le Bars, O. Reynet, and L. Jaulin. Using interval methods in the context of robust localization of underwater robots. In *NAFIPS 2011*, El Paso, USA, 2011.

Conférences sans comité de sélection

F. Le Bars, J. Sliwka, and L. Jaulin. Ray tracing and stability analysis of parametric systems. In *SWIM 2009*, Lausanne, Switzerland, 2009.

J. Sliwka and F. Le Bars. Robotics at ENSIETA. In *AIM'09*, Brno, Czech Republic, 2009.

F. Le Bars, A. Bertholom, J. Sliwka, and L. Jaulin. GESMI, an interval-based software for submarine SLAM.

In *SWIM 2010*, Nantes, France, 2010.

F. Le Bars and J. Sliwka. Calcul par intervalles et robotique à l'ENSIETA. In *Journées Jeunes Chercheurs en Robotique 2010*, Paris, France, 2010.

F. Le Bars, J. Sliwka, and L. Jaulin. SAUC'ISSE, un robot sous-marin autonome. In *Journées Démonstrateurs 2010*, Angers, France, 2010.

F. Le Bars and L. Jaulin. State estimation with fleeting data. In *Journée MEA 2010*, Paris, France, 2010.

F. Le Bars and J. Sliwka. SAUC'ISSE et SARDINE, 2 robots sous-marins autonomes. In *2ème conférence de l'association DAKODOC des doctorants de l'ED SICMA*, Brest, France, 2011.

F. Le Bars, J. Sliwka, O. Reynet, and L. Jaulin. SLAM with fleeting detections. In *SWIM 2011*, Bourges, France, 2011.

Rapports

F. Le Bars, J. Sliwka, and L. Jaulin. Réalisation d'un robot sous-marin autonome pour le concours SAUC-E. *Compte-rendu final pour l'année 2007-2008 pour la MRIS*, 2008.

F. Le Bars, J. Sliwka, and L. Jaulin. Réalisation d'un robot sous-marin autonome. *Fourniture 1 pour la MRIS*, 2009.

J. Sliwka, F. Le Bars, and L. Jaulin. Réalisation d'un robot sous-marin autonome. *Fourniture 2 pour la MRIS*, 2009.

F. Le Bars, J. Sliwka, and L. Jaulin. Réalisation d'un robot sous-marin autonome. *Fourniture 3 pour la MRIS*, 2009.

F. Le Bars, J. Sliwka, and L. Jaulin. Réalisation d'un robot sous-marin autonome. *Fourniture 4 pour la MRIS*, 2009.

F. Le Bars and L. Jaulin. Validation des méthodes de géoréférencement par une méthode de propagation de contraintes sur les intervalles. *Réponses au poste 4 "Module extraction d'amers" de l'étude demandée par le GESMA*, 2010.

J. Sliwka, F. Le Bars, and L. Jaulin. Construction d'un robot sous-marin pour le concours SAUC-E. *Rapport d'avancement T0+6 pour la MRIS*, 2010.

F. Le Bars, J. Sliwka, and L. Jaulin. Construction d'un robot sous-marin pour le concours SAUC-E. *Rapport d'avancement T0+9 pour la MRIS*, 2010.

F. Le Bars, J. Sliwka, and L. Jaulin. Construction d'un robot sous-marin pour le concours SAUC-E. *Rapport d'avancement T0+12 pour la MRIS*, 2010.

J. Sliwka, F. Le Bars, and L. Jaulin. Construction d'un robot sous-marin pour le concours SAUC-E. *Rapport d'avancement T0+6 pour la MRIS*, 2011.

Autres

F. Le Bars, J. Sliwka, and L. Jaulin. SAUC-E 2009 Journal paper ENSIETA. In *SAUC-E 2009*, Gosport, UK, 2009.

F. Le Bars, J. Sliwka, and L. Jaulin. SAUC-E 2010 Journal paper ENSIETA. In *SAUC-E 2010*, La Spezia, Italy, 2010.

F. Le Bars, J. Sliwka, I. Torres-Tamanaja, E. Campos-Mercado, M. S. Ibn Seddik, C. Aubry, and L. Jaulin. SAUC'ISSE and SARDINE, 2 AUVs for SAUC-E 2011. In *SAUC-E 2011*, La Spezia, Italy, 2011.

Résumé des contributions principales

Mes contributions principales au cours de la thèse ont été les suivantes :

- Amélioration du programme GESMI (voir 4.3.2) existant pour qu'il puisse traiter les données de la *Daurade* (rajout de la gestion de la partie bâbord des images sonar...), faciliter l'interprétation des résultats (affichage des erreurs en position...) et aider au maximum l'opérateur humain dans sa recherche de mines (amélioration des indications de positions estimées de mines directement sur l'image sonar...). Ce travail faisait partie d'un contrat entre l'ENSTA Bretagne et le GESMA.
- Présentation et utilisation des tubes (intervalles de trajectoires, voir 3.3.3).
- Formalisation du problème des données fugaces et test d'un algorithme de résolution (utilisant les tubes) sur une simulation de localisation de robot terrestre avec obstacles mobiles inconnus (voir 5.2).
- Lien entre données fugaces et SLAM sous-marin (voir 5.3).
- Prolongement d'une comparaison existante (voir [Joly, 2010]) entre algorithmes de SLAM intervalles et probabilistes (voir 4.4).
- Participation au concours de robots sous-marins SAUC-E avec les sous-marins de l'ENSTA Bretagne (et avec un financement par la MRIS) et aide au test d'un algorithme de localisation robuste (voir 2.2.2, [Sliwka et al., 2011b] et [Sliwka, 2011]).

Activités annexes

Au cours de ma thèse, je me suis beaucoup occupé des robots de l'ENSTA Bretagne. J'ai fait de nombreuses présentations des activités robotiques de l'école avec démonstrations de robots lors de visites de personnes extérieures. Les robots sur lesquels j'ai principalement travaillé sont les 2 AUVs (Autonomous Underwater Vehicle) *SAUC'ISSE* et *SARDINE*. J'ai participé aux concours SAUC-E (Student Autonomous Underwater Challenge - Europe) 2007 (3ème place, j'y étais en tant qu'étudiant et c'était la 1ère participation de l'école au concours), SAUC-E 2008 (2ème place, j'ai participé à la préparation du robot et de l'équipe qui est allée au concours), SAUC-E 2009 (2ème place) et SAUC-E 2010 (3ème place) avec ces sous-marins. La réalisation de ces 2 robots était financée par la MRIS (Mission pour la Recherche et l'Innovation Scientifique), à qui j'ai envoyé certains rapports d'avancement.

J'ai enseigné en automatique, robotique (sur la meute de robots JOGs par exemple) et informatique (environ 150 heures au total sur 3 ans) et participé à l'encadrement de stagiaires ou élèves (notamment dans le cadre du club robotique de l'école) :

- En faisant des mini-cours/TP (en bénévole, le soir) d'initiation à la programmation C/C++, le traitement d'images de webcams, la communication entre capteurs, actionneurs et ordinateurs...
- En gérant leurs achats de matériel (capteurs et actionneurs de robots, ordinateurs embarqués...).
- En leur donnant des conseils techniques pendant leurs projets (informatique, électronique, gestion de l'étanchéité...).

J'ai aidé les personnes travaillant sur le défi CAROTTE 2010 (CArtographie par RObоT d'un TErritoire), le challenge Microtransat et le concours WRSC 2009 (courses de mini-voiliers autonomes).

J'ai aussi été impliqué dans un contrat avec le GESMA (Groupe d'Etudes Sous-Marines de l'Atlantique) à travers le traitement des données de leur AUV *Daurade*.

Plus d'informations sont disponibles sur http://www.ensta-bretagne.fr/lebars.

Index

écho-sondeur, 17, 74
3BCID, 63, 68, 101, 102
3G, 24, 25

accéléromètre, 24
AGPS, 24
AHRS, 24
altimètre, 25
altitude, 28, 31
amer, 17, 28, 31, 74, 77, 126
association des données, 18, 74
AUV, 27, 33, 77

bearing-only, 74, 98
bissection, 18, 51, 53, 63, 64, 72, 76, 135
boussole, 24, 43, 48, 49, 126

caméra, 25, 31, 38, 40, 42–44, 49, 116
capteur de pression, 17, 25, 31, 38, 40, 74
CAROTTE, 44
cartographie, 17, 74, 134
centrale inertielle, 17, 38, 40, 74
compass, 24
connecteur étanche, 38, 39
contracteur, 40, 60, 76, 135
contrainte, 52, 60, 121
CSP, 59, 60, 62, 76, 120, 121

DGPS, 24, 27
donnée aberrante, 18, 40, 52, 74, 134
donnée fugace, 18, 21, 43, 50, 51, 115, 117, 120, 134, 139
DVL, 25, 28, 31, 89

ECA, 27
effet d'enveloppe, 53, 62
effet de dépendance, 53, 55, 62
encoder, 24

ENSTA Bretagne, 21, 33, 43, 44
estimation d'état, 75, 76, 115, 117, 120
estimation de paramètres, 63, 65, 75, 76

fleeting, 18, 115, 117, 120, 134, 139
fonction d'inclusion, 55
forward-backward, 62, 63, 77, 102, 123, 135
FPGA, 43, 53

GESMA, 21, 27, 38
GPS, 17, 24, 27, 38, 74, 77, 89
GPU, 53
GSM, 25
gyromètre, 24, 38

HC4Revise, 62, 68, 77, 100, 123, 135

IMU, 24
inclinomètre, 24
INS, 24, 31, 89
intervalle, 17, 40, 43, 51, 53, 139

Kalman, 27, 52, 75, 77, 89, 101, 108

lidar, 25
localisation, 17, 40, 50, 51, 53, 74, 126, 139
loch, 25
loch Doppler, 17, 25, 74

magnétomètre, 24
multi-occurrence, 53, 55

odomètre, 24, 43, 44, 48, 49
offline, 18, 74, 98, 117, 134
online, 74

pavé, 53, 56
PC/104, 38, 40
ping, 28, 38
portée, 28, 38, 43, 48

propagation de contraintes, 18, 51, 52, 59, 64, 68, 76, 89, 100, 135
propulseur, 37, 41
PWM, 38, 44

radar, 25, 116
range and bearing, 74
range-only, 74
rangescale, 28
ROV, 37

SAUC-E, 33
SHOM, 27
SIVIA, 63
SLAM, 17, 21, 31, 43, 50–53, 63, 73, 75, 98, 115, 134, 139
sonar, 17, 25, 27, 28, 31, 38, 40, 42, 43, 50, 74, 87, 115–117
Strangle, 63, 101
surestimation, 53

télémètre, 25, 43, 44, 48, 50, 116, 126
tape, 36, 38
treillis, 56
tube, 18, 51, 57, 60, 68, 117

variateur, 38, 44, 48

water column, 28
waterfall, 28, 117, 118, 128
waypoint, 40, 41
Wifi, 24, 25, 38, 40, 49, 50

Bibliographie

[Abdallah et al., 2008] Abdallah, F., Gning, A., and Bonnifait, P. (2008). Box particle filtering for nonlinear state estimation using interval analysis. *Automatica*, 44(3) :807–815.

[Aubry et al., 2011] Aubry, C., Desmare, R., and Jaulin, L. (2011). Loop detection in a mobile robot trajectory using interval analysis. In *SWIM 2011*, Bourges, France.

[Baffet and Jaulin, 2011] Baffet, G. and Jaulin, L. (2011). Setdemo. http://www.ensta-bretagne.fr/jaulin/demo.html.

[Bazeille, 2008] Bazeille, S. (2008). *Vision sous-marine monoculaire pour la reconnaissance d'objets*. PhD thesis, Université de Bretagne Occidentale.

[Becis-Aubry et al., 2011] Becis-Aubry, Y., Aubry, D., and Ramdani, N. (2011). Multisensor set-membership state estimation of nonlinear models with potentially failing measurements. In *SWIM 2011*, Bourges, France.

[Benhamou et al., 1999] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F. (1999). Revising Hull and Box Consistency. In *ICLP*, pages 230–244.

[Berz et al., 1996] Berz, M., Bischof, C., Corliss, G., and A., G., editors (1996). *Computational Differentiation : Techniques, Applications and Tools*. SIAM, Philadelphia, Penn.

[Berz and Makino, 1998] Berz, M. and Makino, K. (1998). Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4) :361–369.

[Caiti et al., 2002] Caiti, A., Garulli, A., Livide, F., and Prattichizzo, D. (2002). Set-membership acoustic tracking of autonomous underwater vehicles. *Acta Acustica united with Acustica*, 5(88) :648–652.

[Castellanos and Tardos, 1999] Castellanos, J. A. and Tardos, J. D. (1999). Mobile robot localization and map building : A multisensor fusion approach. *Kluwer*.

[Chabert, 2007a] Chabert, G. (2007a). *IBEX library, available at http://www.emn.fr/z-info/ibex/*. ENSIETA-INRIA.

[Chabert, 2007b] Chabert, G. (2007b). *Techniques d'intervalles pour la résolution de systèmes d'équations*. PhD dissertation, Université de Nice-Sophia Antipolis, Nice-Sophia Antipolis, France. Available at : www.emn.fr/z-info/gchabe08/.

[Chabert and Jaulin, 2009] Chabert, G. and Jaulin, L. (2009). Contractor Programming. *Artificial Intelligence*, 173 :1079–1100. WOS.

[Chabert and Jaulin, 2011] Chabert, G. and Jaulin, L. (2011). Propagation de contraintes sur les intervalles pour la planification d'expérerience en SLAM. *Soumis à Revue d'Intelligence Artificielle*.

[Chabert et al., 2009] Chabert, G., Jaulin, L., and Lorca, X. (2009). A constraint on the number of distinct vectors with application to localization. In Gent, I., editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 196–210. Springer Berlin / Heidelberg.

[Combastel, 2005] Combastel, C. (2005). A state bounding observer for uncertain non-linear continuous-time systems based on zonotopes. In *CDC-ECC '05*.

[Dao et al., 2004] Dao, M., Baguenard, X., and Jaulin, L. (2004). *Proj2d, disponible sur* `www.istia.univ-angers.fr/~dao/`. LISA, ISTIA, Université d'Angers.

[Delanoue, 2006] Delanoue, N. (2006). *Algorithmes numériques pour l'analyse topologique*. PhD dissertation, Université d'Angers, Angers, France. Available at : www.istia.univ-angers.fr/~delanoue/.

[Delanoue et al., 2006] Delanoue, N., Jaulin, L., and Cottenceau, B. (2006). Using interval arithmetic to prove that a set is path-connected. *Theoretical Computer Science, Special issue : Real Numbers and Computers*, 351(1) :119–128.

[Di Marco et al., 2003] Di Marco, M., Garulli, A., Giannitrapani, A., and Vicino, A. (2003). Simultaneous localization and map building for a team of cooperating robots : a set membership approach. *IEEE Transactions on Robotics and Automation*, 2(19) :238–249.

[Di Marco et al., 2004] Di Marco, M., Garulli, A., Giannitrapani, A., and Vicino, A. (2004). A set theoretic approach to dynamic robot localization and mapping. *Autonomous Robots*, 16(1) :23–47.

[Di Marco et al., 2001] Di Marco, M., Garulli, A., Lacroix, S., and Vicino, A. (2001). Set membership localization and mapping for autonomous navigation. *International Journal of Robust and Nonlinear Control*, 7(11) :709–734.

[Dissanayake et al., 2001] Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H. F., and Csorba, M. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions Robotics and Automation*, 3(17) :229–241.

[Drevelle and Bonnifait, 2010] Drevelle, V. and Bonnifait, P. (2010). Robust positioning using relaxed constraint-propagation. In *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, Taipei, China.

[Drocourt et al., 2005] Drocourt, C., Delahoche, L., E. Brassart, B. M., and Clerentin, A. (2005). Incremental construction of the robot's environmental map using interval analysis. *Global Optimization and Constraint Satisfaction : Second International Workshop, COCOS 2003*, 3478 :127–141.

[Dufourd, 2005] Dufourd, D. (2005). *Des cartes combinatoires pour la construction automatique de modèles d'environnement par un robot mobile*. PhD dissertation, Institut National Polytechnique de Toulouse, Toulouse, France.

[Durieu et al., 1996] Durieu, C., Polyak, B., and Walter, E. (1996). Ellipsoidal state outer-bounding for MIMO systems via analytical techniques. In *Proceedings of the IMACS—IEEE–SMC CESA'96 Symposium on Modelling and Simulation*, volume 2, pages 843–848, Lille, France.

[Durrant-Whyte and Bailey, 2006a] Durrant-Whyte, H. and Bailey, T. (2006a). Simultaneous Localization And Mapping : Part I. *IEEE Robotics and Automation Magazine*, pages 99–108.

[Durrant-Whyte and Bailey, 2006b] Durrant-Whyte, H. and Bailey, T. (2006b). Simultaneous Localization And Mapping : Part II. *IEEE Robotics and Automation Magazine*, pages 108–117.

[Eustice et al., 2005] Eustice, R., Singh, H., Leonard, J., Walter, M., and Ballard, R. (2005). Visually navigating the rms titanic with slam information filters. In *Proceedings of Robotics : Science and Systems (RSS)*, Cambridge, MA, USA.

[Ferris et al., 2007] Ferris, B., Fox, D., and Lawrence, N. (2007). Wifi-slam using gaussian process latent variable models. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2480–2485, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Flórez, 2008] Flórez, J. (2008). *Improvements in the ray tracing of implicit surfaces based on interval arithmetic*. PhD thesis, Universitat de Girona.

[Glynn, 2007] Glynn, J. (2007). *Acoustic calibration and bathymetric processing with a KLEIN 5410 sidescan sonar*. PhD thesis, University of New Hampshire, US.

[Gning and Bonnifait, 2006] Gning, A. and Bonnifait, P. (2006). Constraints propagation techniques on intervals for a guaranteed localization using redundant data. *Automatica*, 42(7) :1167–1175.

[Goldsztejn and Jaulin, 2006] Goldsztejn, A. and Jaulin, L. (2006). Inner and outer approximations of existentially quantified equality constraints. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming, (CP 2006)*, Nantes (France).

[Guyonneau et al., 2011] Guyonneau, R., Lagrange, S., Hardouin, L., and Lucidarme, P. (2011). Interval Analysis for Kidnapping Problem using Range Sensors. In *SWIM 2011*, Bourges, France.

[Halbwachs and Meizel, 1996] Halbwachs, E. and Meizel, D. (1996). Bounded-error estimation for mobile vehicule localization. *CESA'96 IMACS Multiconference (Symposium on Modelling, Analysis and Simulation)*, pages 1005–1010.

[Hyvönen, 1992] Hyvönen, E. (1992). Constraint reasoning based on interval arithmetic ; the tolerance propagation approach. *Artificial Intelligence*, 58(1-3) :71–112.

[IXSEA, 2011] IXSEA (2011). IXSEA. http://www.ixsea.com/.

[Jaulin, 2002a] Jaulin, L. (2002a). Nonlinear bounded-error state estimation of continuous-time systems. *Automatica*, 38 :1079–1082.

[Jaulin, 2002b] Jaulin, L. (2002b). Propagation de contraintes sur les intervalles pour l'estimation ensembliste. *Numéro spécial de JESA : Identification des systèmes*, 36(3) :383–395.

[Jaulin, 2009a] Jaulin, L. (2009a). A Nonlinear Set-membership Approach for the Localization and Map Building of an Underwater Robot using Interval Constraint Propagation. *IEEE Transaction on Robotics*, 25(1) :88–98.

[Jaulin, 2009b] Jaulin, L. (2009b). Robust set membership state estimation ; application to underwater robotics. *Automatica*, 45(1) :202–206.

[Jaulin, 2011] Jaulin, L. (2011). Range-only SLAM with occupancy maps ; A set-membership approach. *Accepted in IEEE Transaction on Robotics*.

[Jaulin and Bazeille, 2009] Jaulin, L. and Bazeille, S. (2009). Image shape extraction using interval methods. In *Sysid 2009*.

[Jaulin et al., 2007] Jaulin, L., Bertholom, A., Dabe, F., and Legris, M. (2007). A set approach to the simultaneous localization and map building ; application to underwater robots. *ICINCO 2007*.

[Jaulin and Chabert, 2008] Jaulin, L. and Chabert, G. (2008). QUIMPER : un langage de programmation pour le calcul ensembliste ; Application à l'automatique. In *CIFA 2008*, Bucharest, Romania.

[Jaulin et al., 2001a] Jaulin, L., Kieffer, M., Braems, I., and Walter, E. (2001a). Guaranteed nonlinear estimation using constraint propagation on sets. *International Journal of Control*, 74(18) :1772–1782.

[Jaulin et al., 2001b] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001b). *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer-Verlag, London.

[Jaulin et al., 2006] Jaulin, L., Legris, M., and Dabe, F. (2006). GESMI, un logiciel pour l'aide à localisation de mines sous-marines. In *JIME 2006 (Journées Identification et Modélisation Expérimentale)*, Poitiers (France).

[Jaulin and Walter, 1993] Jaulin, L. and Walter, E. (1993). Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4) :1053–1064.

[Jaulin and Walter, 2002] Jaulin, L. and Walter, E. (2002). Guaranteed robust nonlinear minimax estimation. *IEEE Transaction on Automatic Control*, 47(11) :1857–1864.

[Joly, 2010] Joly, C. (2010). *Contributions aux méthodes de localisation et cartographie simultanées par vision omnidirectionnelle*. PhD dissertation, Ecole Nationale Supérieure des Mines de Paris, Paris, France.

[Joly and Rives, 2008] Joly, C. and Rives, P. (2008). Bearing-only SLAM : comparaison entre une méthode probabiliste et une méthode déterministe. *Rapport de recherche 6602*.

[Joly and Rives, 2009] Joly, C. and Rives, P. (2009). Contribution au SLAM avec caméra omnidirectionnelle. In *Ecole d'été en traitement du signal et des images*, Peyresq, France.

[Kearfott and Kreinovich, 1996] Kearfott, R. B. and Kreinovich, V., editors (1996). *Applications of Interval Computations*. Kluwer, Dordrecht, the Netherlands.

[Knüppel, 1993] Knüppel, O. (1993). *PROFIL/BIAS (Programmer's Runtime Optimized Fast Interval Library / Basic Interval Arithmetic Subroutines) library, available at* http://www.ti3.tu-harburg.de/. Institute for Reliable Computing.

[Kurzhanski and Valyi, 1997] Kurzhanski, A. and Valyi, I. (1997). *Ellipsoidal Calculus for Estimation and Control*. Birkhäuser, Boston, MA.

[L-3 Klein Associates, Inc, 2011] L-3 Klein Associates, Inc (2011). L-3 Klein Associates, Inc. http://www.l-3klein.com/.

[Lahanier et al., 1987] Lahanier, H., Walter, E., and Gomeni, R. (1987). OMNE : a new robust membership-set estimator for the parameters of nonlinear models. *Journal of Pharmacokinetics and Biopharmaceutics*, 15 :203–219.

[Le Bars et al., 2010a] Le Bars, F., Bertholom, A., Sliwka, J., and Jaulin, L. (2010a). Interval SLAM for underwater robots - a new experiment. In *NOLCOS 2010*, Bologna, Italy.

[Le Bars et al., 2009] Le Bars, F., Sliwka, J., and Jaulin, L. (2009). Analyse par intervalles pour le lancé de rayon et pour l'analyse de stabilité. In *JD-JN-MACS 2009*, Angers, France.

[Le Bars et al., 2010b] Le Bars, F., Sliwka, J., and Jaulin, L. (2010b). SAUC'ISSE, un robot sous-marin autonome. In *Journées Démonstrateurs 2010*, Angers, France.

[Le Bars et al., 2011a] Le Bars, F., Sliwka, J., Reynet, O., and Jaulin, L. (2011a). Set-membership state estimation with fleeting data. *Accepted by Automatica*.

[Le Bars et al., 2011b] Le Bars, F., Sliwka, J., Reynet, O., and Jaulin, L. (2011b). SLAM with fleeting detections. In *SWIM 2011*, Bourges, France.

[Leblond, 2006] Leblond, I. (2006). *Recalage à long terme d'images sonar par mise en correspondance de cartes de classification automatique des fonds*. PhD dissertation, Université de Bretagne Occidentale, Brest, France.

[Leonard and Durrant-Whyte, 1992] Leonard, J. J. and Durrant-Whyte, H. F. (1992). *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer, Boston.

[Lydoire and Poignet, 2003] Lydoire, F. and Poignet, P. (2003). Nonlinear predictive control using constraint satisfaction. In *In 2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS)*, pages 179–188.

[Magnetic Declination, 2011] Magnetic Declination (2011). Magnetic Declination. http://www.magnetic-declination.com/.

[Mei and Rives, 2007] Mei, C. and Rives, P. (2007). Cartographie et localisation simultanée avec un capteur de vision. In *JNRR (Journée Nationales de la Recherche en Robotique) 2007*, Obernai, France.

[Meizel et al., 2002] Meizel, D., Lévêque, O., Jaulin, L., and Walter, E. (2002). Initial localization by set inversion. *IEEE Transactions on Robotics and Automation*, 18(6) :966–971.

[Meizel et al., 1996] Meizel, D., Preciado-Ruiz, A., and Halbwachs, E. (1996). Estimation of mobile robot localization : geometric approaches. In Milanese, M., Norton, J., Piet-Lahanier, H., and Walter, E., editors, *Bounding Approaches to System Identification*, pages 463–489. Plenum Press, New York, NY.

[Milanese et al., 1996] Milanese, M., Norton, J., Piet-Lahanier, H., and Walter, E., editors (1996). *Bounding Approaches to System Identification*. Plenum Press, New York, NY.

[Montemerlo et al., 2003] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2003). FastSLAM 2.0 : An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico. IJCAI.

[Moore, 1979] Moore, R. E. (1979). *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, PA.

[olsrd, 2011] olsrd (2011). olsrd. http://www.olsr.org/.

[Piskorski and Lacassagne, 2006] Piskorski, S. and Lacassagne, L. (2006). Efficient 16-bit floating-point interval processor for embedded systems and applications. *Scientific Computing, Computer Arithmetic and Validated Numerics, International Symposium on*, 0 :23.

[Porta, 2005] Porta, J. (2005). Cuikslam : A kinematics-based approach to slam. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2436–2442, Barcelona (Spain).

[Raissi et al., 2004] Raissi, T., Ramdani, N., and Candau, Y. (2004). Set membership state and parameter estimation for systems described by nonlinear differential equations. *Automatica*, 40 :1771–1777.

[Raka and Combastel, 2011] Raka, S.-A. and Combastel, C. (2011). Set-membership methods and tools for the pre-sizing of mechatronic systems : focus on parametric uncertaintie and bounded inputs under linear dynamicsSet-membership methods and tools for the pre-sizing of mechatronic systems : focus on parametric uncertaintie and bounded inputs under linear dynamics. In *SWIM 2011*, Bourges, France.

[Revol, 2001] Revol, N. (2001). Introduction à l'arithmétique par intervalles. *Rapport de recherche 4297*.

[Reynet, 2011] Reynet, O. (2011). Distributed Localization Using Interval Contractors. In *SWIM 2011*, Bourges, France.

[Reynet and Jaulin, 2010] Reynet, O. and Jaulin, L. (2010). A new interval-based method to characterize estimability. *International Journal of Adaptive Control and Signal Processing*, 9999(9999) :n/a+.

[Reynet et al., 2010] Reynet, O., Le Lann, J.-C., and Clément, B. (2010). JOG : une approche haut niveau des systèmes embarqués via Armadeus et Java. In *Journées Démonstrateurs en Automatique 2010*.

[Ribas et al., 2010] Ribas, D., Ridao, P., and Neira, J. (2010). *Underwater SLAM for Structured Environments Using an Imaging Sonar*. Number 65 in Springer Tracts in Advanced Robotics. Springer Verlag, Heidelberg, Alemania.

[Rives, 2009] Rives, P. (2009). Méthodes de SLAM visuel. In *Réunion du GT2 - Véhicules terrestres*, Paris, France.

[Rodriguez et al., 2009] Rodriguez, S., Frémont, V., and Bonnifait, P. (2009). An Experiment of a 3D Real-Time Robust Visual Odometry for Intelligent Vehicles. In *12th International IEEE Conference on Intelligent Transportation Systems (ITSC '09)*, United States.

[Ruiz et al., 2004] Ruiz, I., de Raucourt, S., Petillot, Y., and Lane, D. (2004). Concurrent mapping and localization using sidescan sonar. *IEEE Journal of Oceonic Engineering*, 39(2) :442–456.

[Sam-Haroud, 1995] Sam-Haroud, D. (1995). *Constraint consistency techniques for continuous domains*. PhD dissertation 1423, Swiss Federal Institute of Technology in Lausanne, Switzerland.

[Shettar et al., 2007] Shettar, R., Banakar, R., and Nataraj, P. (2007). Implementation of interval arithmetic algorithms on FPGAs. *Computational Intelligence and Multimedia Applications, International Conference on*, 2 :196–200.

[Sliwka, 2011] Sliwka, J. (2011). Set valued polynomials and their application to robust localization of an underwater robot. In *SWIM 2011*, Bourges, France.

[Sliwka et al., 2009] Sliwka, J., Le Bars, F., and Jaulin, L. (2009). Calcul ensembliste pour la localisation et la cartographie robustes. In *JD-JN-MACS 2009*, Angers, France.

[Sliwka et al., 2011a] Sliwka, J., Le Bars, F., Jaulin, L., and Torres-Tamanaja, I. (2011a). Student autonomous underwater robotics at ENSIETA. *Accepted by IJME (International Journal of Maritime Engineering)*.

[Sliwka et al., 2011b] Sliwka, J., Le Bars, F., Reynet, O., and Jaulin, L. (2011b). Using interval methods in the context of robust localization of underwater robots. In *NAFIPS 2011*, El Paso, Texas, USA.

[Teledyne RD Instruments, 2011] Teledyne RD Instruments (2011). Teledyne RD Instruments. http://www.rdinstruments.com/.

[Telle, 2003] Telle, B. (2003). *Méthode ensembliste pour une reconstruction 3D garantie par stéréovision*. PhD dissertation, LIRMM, Montpellier, France.

[Thrun et al., 2005] Thrun, S., Bugard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press, Cambridge, M.A.

[Trombettoni and Chabert, 2007a] Trombettoni, G. and Chabert, G. (2007a). CID : Disjonction constructive sur intervalles. In *Journées Francophones de Programmation par Contraintes 2007*, Rocquencourt, France.

[Trombettoni and Chabert, 2007b] Trombettoni, G. and Chabert, G. (2007b). Constructive interval disjunction. In *CP'07 - 13th International Conference on Principles and Practice of Constraint Programming*, USA.

[van Emden, 1999] van Emden, M. (1999). Algorithmic power from declarative use of redundant constraints. *Constraints*, 4(4) :363–381.

[Videau et al., 2009] Videau, G., Raïssi, T., and Zolghadri, A. (2009). Guaranteed state estimation for nonlinear continuous-time systems based on qLPV transformations. In *Proceedings of European Control Conference (ECC'09)*, Budapest, Hungary.

[Vinas et al., 2006] Vinas, P. H., Sainz, M. A., Vehi, J., and Jaulin, L. (2006). Quantified set inversion algorithm with applications to control. *Reliable computing*, 11(5) :369–382.

[Vrignaud and Meyrat, 2009] Vrignaud, C. and Meyrat, J. (2009). Use of a DVL in a autonomous underwater vehicle for a rapid environmental assessment. In *ADCP in Action*, San Diego.

[Walter and Pronzato, 1997] Walter, E. and Pronzato, L. (1997). *Identification of Parametric Models from Experimental Data*. Springer-Verlag, London, UK.

[Wang et al., 2007] Wang, C., Thorpe, C., Thrun, S., Hebert, M., and Durrant-Whyte, H. (2007). Simultaneous localization, mapping and moving object tracking. *International Journal of Robotics Research*, 26(9) :889–916.

[Williams et al., 2001] Williams, S., Dissanayake, G., and Durrant-Whyte, H. (2001). Towards terrain-aided navigation for underwater robotics. *Advanced Robotics*, 15(5) :533–549.

Résumé

Cette thèse étudie le problème de la localisation et cartographie simultanées des robots sous-marins, et ses méthodes de résolution utilisant le calcul par intervalles.

Le principe du SLAM (Simultaneous Localization And Mapping, ou cartographie et localisation simultanées) est le suivant: un robot sous-marin connait en général sa position initiale (lorsqu'il est à la surface grâce à un GPS), son modèle de déplacement (très approximativement) et possède quelques capteurs l'aidant à estimer sa position (capteur de pression pour mesurer sa profondeur, loch Doppler pour mesurer sa vitesse et sa distance au fond, centrale inertielle pour mesurer son orientation) et voir ce qui l'entoure (sonar). Pourtant, malgré tous ces capteurs, plus il avance, plus ses erreurs d'estimation de position s'accumulent: le robot se perd. En passant et repassant devant plusieurs objets (ou éléments remarquables quelconques de son environnement), il va pouvoir évaluer leur position (plus ou moins précisément) une première fois à partir de la sienne (cartographie grâce à sa localisation), puis recalculer sa trajectoire en les prenant comme repère les fois suivantes quand il est perdu (localisation grâce à sa cartographie).

Les mesures de capteurs ou variables utilisées pour décrire les robots étant souvent entachées d'erreurs, elles peuvent être représentées de différentes façons: distributions probabilistes, nuages de points, ensembles continus... En général, les données constructeur des capteurs ou actionneurs du robot nous indiquent des bornes (liées à la précision...). On peut donc représenter ces valeurs sous forme d'intervalles. Les méthodes ensemblistes telles que l'analyse par intervalles permettent d'obtenir des résultats à partir d'équations sur des intervalles. L'avantage de ces méthodes est qu'on est sûr de ne perdre aucune solution (compte tenu des hypothèses faites), contrairement à celles utilisant l'approche probabiliste, où on n'obtient parfois que les solutions les plus probables.

Dans cette thèse, l'utilisation du calcul par intervalles dans le cadre du SLAM appliqué aux robots sous-marins et une comparaison entre plusieurs méthodes existantes seront étudiées. De plus, une nouvelle méthode permettant de mieux gérer le problème des données fugaces (données seulement significatives à des instants bien précis et inconnus), rencontré notamment avec des données provenant de sonars, sera proposée.

Les applications de ces travaux concernent par exemple le développement de robots sous-marins autonomes (souvent appelés AUVs pour Autonomous Underwater Vehicles ou UUVs pour Unmanned Underwater Vehicles). En effet, contrairement aux robots téléguidés par des humains, ceux-ci doivent eux-mêmes être capables de se repérer dans leur environnement pour effectuer leur travail. Ces robots peuvent avoir des missions variées: relevé de données hydrographiques, localisation d'épaves (bateau, avion...) ou objets dangereux (mines...), surveillance (détection de pollution, vérification de l'état de pipelines...)... Actuellement, ces tâches sont principalement réalisées par des humains, directement avec des plongeurs, ou indirectement avec des sous-marins téléopérés pour les travaux les plus dangereux ou difficiles d'accès.

Mots-clés : SLAM, intervalles, sous-marins, robotique, données fugaces

Abstract

This thesis studies the simultaneous localization and mapping problem for submarine robots, and its resolution methods using interval analysis.

The principle of SLAM (Simultaneous Localization And Mapping) is the following: a submarine robot usually knows its initial position (when it is at the surface thanks to a GPS), its moving model (approximately) and has sensors enabling it to estimate its position (pressure sensor to get its depth, DVL to get its speed and measure its distance to the sea floor, inertial navigation system to get its orientation) and see its surrounding environment (sonar). However, in spite of all these sensors, the more it moves, the more its position estimation errors increase: the robot is lost. By going next to the same objects (or any distinguishable mark in its environment) several times, it should be able to evaluate their position (with a given accuracy) the first time from its own position (cartography from localization), then compute and correct its trajectory evaluation by taking them as mark next time when it is lost (localization from cartography).

Because measurements from sensors or variables used to describe the behaviour of robots are often erroneous, they can be represented using different ways: probabilistic distributions, particles, continuous sets... Sensors or actuators manufacturers usually provide bounds (related to precision, accuracy...). Therefore, we can represent these values as intervals. Set-membership methods such as interval analysis enable to obtain results from equations involving intervals. The main advantage of these methods is that it is sure that no solution is lost (taking into account the assumptions made), contrary to probabilistic approaches, where the most probable solutions are obtained.

In this thesis, the use of intervals computations for the SLAM of underwater robots and a comparison between several existing methods are studied. Additionally, a new method to handle better the problem of fleeting data (data that are only significant during short and unknown time intervals), often met with data from sonars, will be proposed.

Applications of this work are for example in the development of autonomous submarine robots (often called AUVs for Autonomous Underwater Vehicles or UUVs for Unmanned Underwater Vehicles). Indeed, contrary to teleoperated robots, they must be able to localize themselves in their environment to do their work. These robots can do several missions: hydrographic data collection, shipwrecks or objects (sea mines...) localization, surveillance (pollution detection, pipelines surveys...)... Nowadays, those tasks are mostly done by humans, directly with divers or indirectly using teleoperated submarines when the conditions are dangerous or difficult.

Keywords : SLAM, intervals, submarines, robotics, fleeting data