REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Génie Electrique
Spécialité Automatique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme
D'Ingénieur d'Etat en Automatique

Thème

Navigation référencée capteurs d'un robot d'intérieur en environnement dynamique

Realisé par : **Mlle GABOUR Nour El Houda**
Mr BRIEDJ Fawzi

Encadré par :

Mr CHEKIREB Hachemi
Mr OUADAH Nour Eddine

Examinateurs :

Mr ABDELOUEL Lahcene
Mr ILLOUL Rachid

Centre d'accueil : **CDTA**
Centre de Développement des Technologies Avancées.

Juin 2016

Ecole Nationale Supérieure Polytechnique, 10, AV. Hassen Badi, El-Harrach, Algérie

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Génie Electrique
Spécialité Automatique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme
D'Ingénieur d'Etat en Automatique

Thème

Navigation référencée capteurs d'un robot d'intérieur en environnement dynamique

Realisé par : **Mlle GABOUR Nour El Houda**
Mr BRIEDJ Fawzi

Encadré par :

Mr CHEKIREB Hachemi
Mr OUADAH Nour Eddine

Examinateurs :

Mr ABDELOUEL Lahcene
Mr ILLOUL Rachid

Centre d'accueil : **CDTA**
Centre de Développement des Technologies Avancées.

Juin 2016

Ecole Nationale Supérieure Polytechnique, 10, AV. Hassen Badi, El-Harrach, Algérie

Remerciements

Tous d'abord on remercie Dieu de nous avoir accordé la connaissance, donner le courage, la patience et la santé pour réaliser notre projet de fin d'étude.

Nos remerciements les plus vifs s'adressent tout particulièrement à Mr CHEKIREB Hachemi professeur à l'Ecole Nationale Polytechnique ENP, et Mr OUADAH Noureddine, pour avoir accepté de diriger ce travail, et à Mr KHELOUFI Abdellah, Mlle MAHTOUT Imene et Mlle BELARBI Abir pour nous avoir aidé, conseillé et encouragé tout au long de nos recherches.

Nos remerciements s'adressent également aux membres du jury qui ont accepté de lire et d'évaluer ce mémoire.

Nous tenons aussi à remercier

Le personnel du Centre de développement des technologies avancées CDTA qui nous ont fourni les informations nécessaires à la réalisation du présent mémoire.

Enfin, nous remercions

Toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

Dédicace

À la femme la plus chère au monde, qui a fait de moi ce que je suis aujourd'hui

Ma mère

À mon idole qui m'a appris à être autonome et aimer le travail

Mon père

A mes chères cousines : Zineb et Soumya et Sarah

A ma petite cousine : Mouna Lyna

A tous les membres de ma famille : oncles, tantes, cousins et cousines,

À toute la Famille GABOUR et LARBI

À mon binôme et ami : BRIEDJ Fawzi

À tous mes enseignants,

À tous ceux que j'aime et qui m'aiment.

GABOUR Nour El Houda.

Dédicace

À Ma mère qui m'a aimé, soutenu et encouragé durant toutes mes études

À mes frères qui sont restés à mes côtés

À mes amis proches que j'ai retrouvés durant les épreuves les plus difficiles

À mon binôme et amie Nour El Houda pour sa patience et son courage

À ma grande famille qui m'a aidé à me consacrer à mes études

Aux enseignants qui ont su faire naître chez moi l'amour du savoir

À mon père, pour m'avoir rendu curieux d'apprendre, pour sa gentillesse, et pour les valeurs qu'il m'a transmises.

Fawzi BRIEDJ.

ملخص

العمل المقدم في هذه الأطروحة يتمحور حول الروبوتات المتنقلة في الأماكن المغلقة، في إطار ابحار روبوت دليل في محيط داخلي مزدحم، عن طريق المراقبة البصرية لتتبع هدف متحرك بغاية تحسين استقلالية المنصة المتحركة (الروبوت) من خلال تعزيز تفاعله مع البيئة المحيطة به عن طريق مختلف أجهزة الاستشعار الموجودة على متنه.

كلمات البحث: الروبوتات، المراقبة البصرية، يعتم، وتجنب العقبات.

Abstract

The work presented in this thesis deals with the indoor mobile robotics, it is placed under the navigation of an inner guide robot in a cluttered environment by performing a visual servoing for tracking a dynamic target in the purpose of improving the performance of a mobile platform by enhancing its interaction with the environment surrounding it, through its various on-boardsensors.

Key words: robotics, visual servoing, occlusion, obstacle avoidance.

Résumé

Le travail présenté dans ce mémoire traite de la robotique mobile d’intérieur, il est placé dans le cadre de la navigation d’un robot guide d’intérieur dans un environnement encombré en effectuant un asservissement visuel pour le suivi d’une cible dynamique dans le but de l’amélioration de l’autonomie d’une plateforme mobile en améliorant son interaction avec le milieu l’entourant à travers ses différents capteurs embarqués.

Mots clés : robotique, asservissement visuel, occultation, évitement d’obstacles.

Table des Matières

CHAPITRE 4 : Evitement d'obstacles et gestion de l'occultation

CHAPITRE 5 : Lois de commande et résultats expérimentaux

Liste des figures

Liste des figures

Introduction Générale

et

Cahier des Charges

Introduction Générale

La robotique, apparue dans les années 1960, consiste à concevoir et à étudier des machines intelligentes, c'est-à-dire des machines capables de réaliser des tâches avec un certain degré d'autonomie. Les disciplines de recherche associées à cette thématique sont articulées autour de la perception (traitement du signal, reconnaissance de formes), de la décision (intelligence artificielle) et de l'action (automatique). Dans un premier temps, les chercheurs se sont intéressés à la robotique de manipulation et à ses applications notamment dans l'industrie manufacturière. A partir des années 1980, en raison de l'augmentation de la puissance de calcul embarquée et de l'amélioration des capteurs, la robotique mobile devient un thème de recherche important avec comme objectif la création de robots capables d'explorer des zones dangereuses pour l'homme en ce qui concerne la robotique d'extérieur, en ce qui concerne la robotique d'intérieur, l'objectif est la création de robots accompagnateurs capables d'interagir avec des humains : aide aux personnes âgées, robots guide dans une institution (musée, hôpital…).

Cahier des charges

Le travail, qui nous a été assigné dans le cadre de notre PFE, s'intègre dans un projet global relatif à « **la navigation d'un robot guide d'intérieur dans un environnement encombré** ». Au moment de la prise en main de ce projet son état d'avancement était comme suit :

1. Le robot peut, grâce au programme « Open NI tracker » distinguer un individu **H1**, donner les coordonnées de plusieurs points de son squelette (torse, épaules…) et suivre son mouvement. Le programme a, malgré tout, des failles concernant le tracking quand, dans l'environnement, il y a des reflets (un miroir par exemple) ou quand la luminosité est élevée.
2. Le robot ne peut pas retrouver **H1** si celui-ci disparaît du champ de vision, quand ce dernier réapparaît dans le champ de vision du robot, il lui est affecté l'identifiant **H2** (ne sachant pas que c'est la même personne de départ).
3. Le robot peut poursuivre **H1** si celui-ci se déplace en ligne droite, ou s'il dévie avec une vitesse angulaire faible. Si **H1** tourne brusquement, le robot adopte une trajectoire aberrante.

4. L'évitement d'obstacles implémenté n'est pas adapté au suivi d'une cible. Il consiste à faire naviguer le robot sur un chemin quelconque et de lui faire l'apprentissage de ce dernier en stockant une succession d'images clés du chemin ; ensuite le robot est renvoyé sur ce même chemin mais cette fois-ci en présence d'obstacles que celui-ci a pour mission d'éviter.

Notre propre tâche consiste à atteindre les objectifs suivants:

1- Résoudre le problème d'occultation en implémentant un filtre de Kalman pour prédire la trajectoire de la personne suivie (**H1**) et ainsi de la retrouver si celle-ci est occultée par un autre objet.
2- Résoudre les problèmes concernant la poursuite dus à un changement brutal d'angle, et ceci, en changeant complètement de stratégie de commande (celle-ci sera décrite plus en détails dans le Chapitre 2).
3- Adapter l'évitement d'obstacles déjà implémenté au contexte de poursuite de la cible (**H1**).

Le travail présenté dans ce mémoire traite de la robotique mobile d'intérieur, il est placé dans le cadre de l'amélioration de l'autonomie de la plateforme mobile b21r grâce à une meilleure interaction avec le milieu l'entourant à travers ses capteurs embarqués.

Le mémoire est structuré en une introduction générale suivie de cinq chapitres dont le contenu est présenté comme suit :

Dans le chapitre 1, nous présentons un aspect global sur les robots mobiles ; leur intérêt et leur classification selon le degré d'autonomie avant de présenter notre catégorie de robots qui sont les robots mobiles à roues, pour ensuite passer à une présentation des parties traitées dans notre travail qui sont : la commande visuelle, la gestion des occultations et enfin l'évitement d'obstacles.

Pour la première partie, les différentes méthodes d'asservissement visuel sont présentées ; pour les deux dernières parties un bref historique des travaux déjà effectués en fait l'objet.

Dans le chapitre 2, nous présentons les différentes méthodes de détection et de suivi des personnes basées sur le système de vision utilisé qui est la Kinect ainsi que le télémètre laser sur laquelle se base la détection d'obstacles.

Concernant le chapitre 3, il traite de la partie logicielle du mémoire, une présentation de l'OS utilisé ROS (Robot Operating System) en fait l'objet pour la première partie ; suivi d'une présentation de la hiérarchie des différents programmes implémentés pour la seconde partie.

Ensuite, les chapitres 4 et 5 regroupent les algorithmes clés du mémoire, à savoir ; l'évitement d'obstacles et la gestion des occultations en ce qui concerne le chapitre 4, et la loi de commande avec ses trois variantes, en ce qui est de la première partie du chapitre 5.

Enfin, la deuxième partie du chapitre 5 a été dédiée aux résultats expérimentaux des différents tests effectués et à leurs interprétations ;

Pour finir, une conclusion générale qui clôt notre mémoire.

CHAPITRE 1

Généralités sur les robots mobiles

et leurs asservissements Visuels

➢ Introduction

Dans ce premier chapitre, nous présentons l'aspect général du travail qui est divisé en quatre parties :

- Une première partie dédiée aux généralités sur les robots mobiles
- Une deuxième partie qui traite de l'asservissement visuel dans son aspect référencé vision
- Une troisième partie qui présente la problématique de gestion des occultations d'un point de vue historique
- Enfin, une quatrième partie qui présente également la problématique de l'évitement d'obstacles

1.1 Les robots mobiles :

1.1.1 Généralités sur les robots mobiles :

Un robot est un système alimenté en énergie qui évolue dans un environnement statique ou dynamique, il est formé d'un microcontrôleur ainsi que d'un ou plusieurs capteurs et actionneurs. La conception d'un robot se base sur son cahier des charges. Elle comprend l'analyse du comportement souhaité pour le robot et sa synthèse théorique, à l'aide notamment des théories d'asservissement, ainsi que l'implémentation logicielle et matérielle du robot.

L'ensemble des problèmes particuliers liés à la conception de tels robots sont :

- La conception mécanique liée à la mobilité,
- La détermination de la position et de la latitude (orientation),
- La détermination du chemin optimal pour atteindre le lieu de la tâche.

La structure d'un robot est contrôlée de manière à effectuer un ensemble de tâche. Ce contrôle inclut trois phases distinctes qui se répètent en boucle : la perception, le traitement et l'action. Un robot fonctionne par l'exécution continue d'un programme informatique constitué d'algorithmes. Ce programme est écrit dans un langage de programmation dont la nature est choisie par le constructeur.

La phase de perception est assurée par l'utilisation de capteurs. Les capteurs donnent une information à propos de l'environnement ou des composants internes (ex : position d'un moteur ou d'un vérin, état d'une LED, etc.). Cette information est utilisée pour calculer l'ordre approprié à envoyer aux actionneurs.

La phase de traitement est assurée par un microcontrôleur ou un ordinateur embarqué, elle peut varier en complexité. À un niveau réactif, un robot peut traduire l'information brute d'un capteur directement en commande d'un actionneur (ex : un arrêt d'urgence ; si un obstacle est détecté alors arrêt des moteurs). Avec des tâches plus sophistiquées, il faut utiliser des algorithmes.

La phase d'action est réalisée à l'aide d'actionneurs.

1.1.2 Intérêts des robots mobiles :

Les robots mobiles ont une place particulière en robotique. Leur intérêt réside dans leur mobilité qui ouvre des applications dans de nombreux domaines. Comme les robots manipulateurs, ils sont destinés à assister l'homme dans les tâches pénibles (transport de charges lourdes), monotones ou en ambiance hostile (nucléaire, marine, spatiale, lutte contre l'incendie, surveillance…). L'aspect particulier de la mobilité impose une complexité technologique et méthodologique qui s'ajoute en général aux problèmes rencontrés par les robots manipulateurs. La résolution de ces problèmes passe par l'emploi de toutes les ressources disponibles tant au niveau technologique (capteurs, motricité, énergie) qu'à celui du traitement des informations par l'utilisation des techniques de l'intelligence artificielle ou de processeurs particuliers. L'autonomie du robot mobile est une faculté qui lui permet de s'adapter ou de prendre une décision dans le but de réaliser une tâche malgré des informations préliminaires absentes ou éventuellement erronées.

1.1.3 Classification des robots mobiles selon leur degré d'autonomie :

Une classification est proposée dans la littérature se basant sur le degré d'autonomie du robot mobile. On distingue :

• Véhicule télécommandé par un opérateur lui imposant chaque tâche élémentaire à réaliser.

• Véhicule télécommandé par un opérateur lui imposant des tâches plus complexes.

• Véhicule semi-autonome réalisant sans l'aide de l'opérateur des tâches prédéfinies.

• Véhicule autonome qui réalise des tâches semi-définies. Ce type de véhicule pose des problèmes d'un niveau de complexité élevé de représentation des connaissances, de capacité décisionnelle et de génération de plans qui sont résolus à bord dans la mesure du possible.

1.1.4 Classification des robots mobile à roue :

On trouve généralement deux grandes catégories de robots mobiles à roues :

a) Robots omnidirectionnels :

Dit aussi Holonome, ce terme se réfère à la relation entre les degrés commandabilité et le degré de liberté d'un robot. Si le degré de commandabilité et de liberté sont égaux, alors le robot est dit holonome. Un robot construit sur roues ou Omni-roues est un bon exemple derobot holonomes car il peut se déplacer librement dans toutes les directions. L'image montre une roue pivotante qui peut tourner dans les deux axes X et Y.

Fig.1.1 : roue d'un robot holonome.

b) Robots non holonomes :

Si le degré de commandabilité est inférieur au degré de liberté, alors le robot est considéré comme non-holonome. Une voiture a trois degrés de liberté ; sa position est calculée dans deux axes et son orientation. Cependant, il y a seulement deux degrés de liberté contrôlables qui sont l'accélération (ou le freinage) et l'angle de braquage du volant. Cela rend difficile pour le conducteur de tourner la voiture dans toutes les directions.

1.2 Commande visuelle :

1.2.1 Commande référencée capteurs :

La commande référencée capteurs consiste à exprimer les tâches robotiques à réaliser, non plus dans l'espace des configurations, mais directement dans l'espace du capteur sous la forme d'une relation locale entre le robot et son environnement. On cherche ainsi à définir des commandes en boucle fermée sur la base des informations provenant des capteurs et non plus

sur la base de la configuration du robot. De ce fait, les lois de commande généralement synthétisées dans le cadre de cette approche sont de type *retour de sortie* et permettent de réaliser les tâches désirées de manière plus précise. Ainsi, toute tâche pouvant s'exprimer sous la forme d'une relation entre le robot et l'environnement sera réalisable par le biais de la commande référencée capteurs. Toutefois, cette approche requiert l'utilisation de capteurs performants, capables de fournir les mesures à une fréquence compatible avec la fréquence d'échantillonnage de l'asservissement.

1.2.2 Commande référencée vision :

La commande référencée vision consiste à contrôler les mouvements d'un système robotique en utilisant des informations visuelles issues d'une ou plusieurs caméras (ou plus généralement d'un capteur de vision) embarquées ou non sur le système. De nombreux travaux sont basés sur l'exploitation de ces données pour réaliser différents objectifs tels que le positionnement face à un objet, son suivi, sa saisie, etc. La première utilisation de la vision en boucle fermée est due à Shirai et Inoue qui décrivirent comment un capteur de vision pouvait augmenter la précision du positionnement [Shi 73]. On parlait alors de retour par vision (*visual feedback*). Mais c'est à Hill et Park que l'on doit l'apparition du terme asservissement visuel (*visual servoing*) [Hil 79]. Plusieurs approches ont depuis vu le jour : Sanderson et Weiss ont proposé ainsi deux grandes classes selon l'utilisation des informations visuelles donc on a l'asservissement visuel 3D (ou *position-based control*) et l'asservissement visuel 2D (ou *image-based control*) [San 80]. Notons qu'il existe aussi des approches intermédiaires plus récentes telles que l'asservissement visuel 2D½ ou d2D/dt.

a) Asservissement visuel 2D :

On parle d'asservissement visuel **2D** lorsque l'information visuelle est définie dans le plan image d'où sont extraites les primitives de l'image. Les primitives sont des formes géométriques élémentaires (point, segment de droite, portion d'ellipse…) associées à l'image. Ceci permet de modéliser la projection de la cible dans le plan image.

Une fois les indices visuels sélectionnés, l'asservissement **2D** consiste à contrôler le mouvement du robot de façon à déplacer les primitives observées dans l'image, de la position courante $\boldsymbol{s}$ vers la position désirée $\boldsymbol{s}^*$. Ceci est obtenu**,** en synthétisant une commande assurant la convergence vers zéro d'une fonction de tâche définie dans l'image qui doit permettre la

convergence de la valeur courante des primitives visuelles $\boldsymbol{s}$ vers leurs valeurs désirées $\boldsymbol{s}^*$. On choisit généralement **N** primitives de type point avec **N≥4**.

Afin de synthétiser la commande en asservissement visuel 2D, il est nécessaire d'établir la matrice Jacobienne entre le torseur cinématique du repère de la caméra et les vitesses de déplacement des primitives dans l'image.

b) Asservissement visuel 3D :

Dès 1979 et pour améliorer la précision de la tache de positionnement des robots Tamtsia, Shirai et Hill ont exploité les techniques de l'asservissement visuel 3D [Tam 13].

Contrairement à l'asservissement visuel 2D qui utilise les cordonnées de la projection de la cible dans le plan de l'image, l'asservissement 3D se base sur une mesure 3D de la cible dans la scène.

Historiquement, l'obtention des mesures 3D nécessitait une reconstruction 3D de l'information visuelle. Cette dernière est basée sur une estimation de la position relative entre la caméra et l'objet dans la scène. Cette estimation est déduite à partir des données visuelles extraites de l'image observée par conséquent, la connaissance d'un modèle tridimensionnel de la cible est nécessaire.

De ce fait, la reconstruction 3D engendre une complexité de calcul s'accroissant avec la complexité de la cible. Cependant le développement des capteurs extéroceptifs (laser, ultrason,…) et surtout la camera RGB_D appelée Kinect a permis de résoudre ce problème aisément. En effet, ce dispositif fournit en ligne la mesure de la profondeur grâce aux algorithmes de perception embarqués sur ce dispositif.

En utilisant les informations 3D, l'asservissement 3D repose sur plusieurs techniques qui se différencient par le type des signaux à réguler et la manière de les exploiter. Parmi ces techniques, on distingue celles utilisant les primitives géométriques extraites de la cible (point, segment de droit, portion d'ellipse…) dans la plupart des cas il s'agit de points. Tout d'abord la fonction de la tâche est définie en fonction des coordonnées 3D de ces points puis la régulation a pour objet d'assurer la convergence à zéro de la fonction tache donc les points courants sont amenés vers les points désirés. Cette technique est souvent appelée « asservissement visuel 3D point ».

D'autres méthodes privilégient la régulation de la pose de la cible dans la scène par rapport à la Kinect (position et orientation). Pour ces méthodes l'objectif de commande consiste à faire

converger le repère lié à la cible vers un repère désiré ou encore le repère de la Kinect vers un repère désiré.

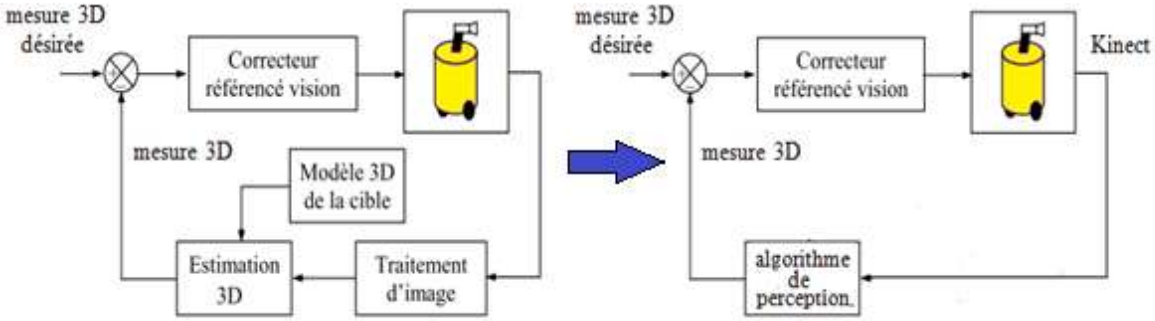


Fig.1.2Boucle d'asservissement 3D

c) Asservissement visuel 2D$\frac{1}{2}$:

L'asservissement visuel 2D$\frac{1}{2}$ a été introduit par ***Chaumette et al*** [Cha02] et il est considéré comme une variante de l'asservissement 2D. En effet, il utilise une combinaison d'informations ou certaines d'entre elles sont dans l'image et d'autres dans l'espace capteur. Cette technique est particulièrement intéressante car elle ne présente pas les inconvénients du 2D ni ceux du 3D.

d) Asservissement visuel d2D/dt :

Ce type d'asservissement visuel se base sur la régulation de la vitesse relative entre la cible et la caméra. La référence est définie comme un champ de vitesse désiré des indices visuels dans le plan image. Le principe de la commande consiste alors à contrôler les mouvements de la caméra de telle sorte que le mouvement 2D mesuré atteigne un champ de vitesse désiré d'où l'appellation d'asservissement visuel d2D/dt. Contrairement aux techniques présentées précédemment qui contraignent les vitesses du robot pour pouvoir asservir les positions liées à la cible, cette technique contraint, en plus des vitesses, les accélérations afin d'asservir la vitesse des primitives.

1.3 Gestion des occultations :

Le mot occultation, est emprunté du latin *occultatio* : action de cacher ; en astronomie, il signifie la disparition passagère d'un astre caché par un autre.

Habituellement, l'occultation se définit comme étant un phénomène de recouvrement d'un élément par un autre. En asservissement visuel, cela se traduit par la perte des informations sensorielles généralement liée au masquage du motif visuel par un élément de l'environnement. Dans le cadre de notre étude, le fait que la cible est susceptible de sortir du champ de vision de la caméra, est également assimilé à une occultation.

Le suivi de personnes est devenu essentiel pour le service robotique puisque le robot est supposé interagir et naviguer dans un environnement encombré de personnes. La vision référencée capteurs est basée sur l'asservissement visuel, il opère dans l'espace image en se basant sur la caméra pour accomplir différentes tâches.

Les approches de l'asservissement visuel ont été initialement appliquées à un bras manipulateur à 6 degrés de liberté. L'application a été étendue aux robots mobiles depuis 1990. Dans le contexte de la navigation des robots mobiles, trois tâches critiques doivent être prises en considération et plus particulièrement pour le cas des robots non holonomes :

- Les tâches de positionnement tout en prenant en compte les propriétés non holonomes du robot.
- La combinaison entre l'asservissement visuel dans l'image et d'autres tâches tel que l'évitement d'obstacles.
- La visibilité des primitives de l'image pendant l'asservissement plus particulièrement lorsque le capteur de vision est fixé sur la plateforme mobile.

Pour la première tâche, des techniques (d'asservissement visuel) ont été abordées initialement dans ce même chapitre.

Pour la deuxième tâche, la question sera détaillée dans le chapitre 4 (évitement d'obstacles).

Pour la troisième tâche, assurer la visibilité reste un continuel défi. La visibilité nécessite qu'un nombre minimum d'informations visuelles reste dans le champ de vision de la caméra pendant l'asservissement. Si ces informations viennent à être perdues, l'asservissement visuel échoue. La perte des informations est causée par l'occlusion de la cible ou par l'application d'une loi de commande pour une tâche spécifique telle que l'évitement d'obstacles. Des méthodes ont été développées pour résoudre cette question à travers l'évitement de

l'occultation de la cible et ainsi garder les informations visuelles dans le dit champ de vision pendant l'asservissement. Ces méthodes ont été dédiées aux bras manipulateurs avec 6 degrés de liberté et tirent avantage des redondances. Dans le cas de la robotique mobile Chirubini et al. [Chi 08] proposent une approche basée sur la redondance en rajoutant des degrés de liberté pour la caméra. Cette approche permet de garder la cible dans le champ de vision pendant l'évitement d'obstacles. Folio et al proposent une méthode pour prendre en compte la visibilité et les collisions [Fol 05]. Contrairement à la préservation de tous les indices visuels, une première approche permet le changement de ces derniers durant l'asservissement, par contre cette méthode est limitée aux changements partiels de la visibilité, mais encore si le nombre des indices visuels qui apparaissent est plus faible que celui requit, la méthode échoue également [Gar 05].

Dans le but de traiter la question de perte totale des indices visuels, une méthode est introduite, elle applique un algorithme d'estimation des informations visuelles. Elle utilise l'information précédente et l'entrée de commande afin d'estimer la prochaine information nécessaire lorsque celle-ci est perdue (la cible est sortie du champ de vision). Cependant la méthode est sujette aux erreurs d'estimation [Intelligent Robots and Systems, 2008].

Dans le cadre de nos travaux, nous nous intéressons à l'estimation des indices visuels en se basant sur un estimateur, le Filtre de Kalman, tout en utilisant les informations envoyées précédemment par la caméra (Kinect).

1.4 Evitement d'obstacles :

Dans le cas de la navigation autonome, une tâche importante est l'évitement d'obstacles, qui consiste soit à générer une trajectoire sans collision avec l'obstacle [Min 08], ou de décélérer pour éviter une collision lorsque le robot se retrouve dans une impasse [Wad 09]. La plupart des techniques d'évitement d'obstacles, en particulier ceux qui utilisent la planification de trajectoires [Lat 91], reposent sur un modèle global de l'environnement avec ses obstacles.

Au lieu d'utiliser un tel modèle global de l'environnement, qui porterait atteinte au paradigme *l'action suivant la perception* [Sci 00], le travail est fait dans un cadre d'évitement d'obstacle avec exécution simultanée d'une tâche de l'asservissement visuel [Cha 06].

La tâche visuelle est basée sur la navigation référencée vision, [Šeg 08, Che 09, Dio 11]. Dans ce cadre, le chemin est un graphe topologique, représenté par une base de données d'images

clés. La tâche traitée est purement basée sur l'image, [Bec 10] et elle est divisée en une série de tâches secondaires, constitués chacune d'entraînement du robot vers l'image clé suivante dans la base de données.

L'évitement d'obstacles a été intégré dans de nombreux systèmes de navigation à base de modèles. Une large gamme de vision monoculaire permet la navigation dans un environnement. [Yan 03]. La trajectoire souhaitée est déformée pour éviter les obstacles détectés. [Lam 04]. Les auteurs de [Ohy 08] utilisent un système de vision basé sur un modèle avec la correction de position rétroactive. Le suivi de trajectoire et l'évitement d'obstacles simultanés est présenté dans [Lap 07], où la géométrie de la trajectoire (une courbe sur le sol) est parfaitement connue. Dans [Lee 10], le rayon des obstacles (supposé cylindrique) est connu a priori. En pratique, toutes ces méthodes sont basées sur un modèle de l'environnement 3D, y compris, par exemple, des murs et des portes, ou sur la géométrie de la voie. En revanche, le cadre du travail actuel repose sur un système de navigation qui ne nécessite ni le modèle de l'environnement, ni celui de l'obstacle. Une des techniques les plus courantes pour l'évitement d'obstacles sans modèle de l'environnement est la méthode des *champs de potentiels*, initialement introduite dans [Kha 85]. Le travail de [Qui 93] consiste à utiliser la planification de trajectoire globale et le contrôle à base de capteurs en temps réel : un chemin sans collisions, déformable, dont la forme initiale est générée par un planificateur, puis déformé en temps réel selon les données détectées. De même, un ensemble de trajectoires (arcs de cercles ou "tentacules") est évalué pour la navigation. [Bon 01, Hun 08]

Le travail actuel se base sur ce problème : un véhicule à roues, équipé d'une caméra sténopé actionnée et avec un télémètre laser, doit suivre un chemin représenté par des images clés, sans entrer en collision avec les obstacles. La caméra détecte les caractéristiques requises pour la navigation, tandis que le laser détecte les obstacles.

Le travail est inspiré de celui de Chirubini [Che11]. Pour l'évitement d'obstacles, le domaine de potentiels classiques a été remplacé par une nouvelle technique basée sur les tentacules. [Bon 01] et [Hun 08].
L'asservissement visuel est réalisé au final tel qu'il doit inclure une loi de commande assurant la convergence des primitives visuelles vers les consignes désirées tout en évitant les obstacles rencontrés en cours de chemin. Enfin, nous proposons de faire l'estimation de la position de la personne en cas de perte de cette dernière par le biais d'un programme de gestion de l'occultation.

➢ Conclusion

Ce premier chapitre a englobé des généralités sur la robotique mobile et plus particulièrement sur les robots mobiles à roues avec une brève classification de ces derniers avant de passer à la commande référencée vision et aux différentes méthodes d'asservissement visuel.

Enfin dans les deux dernières parties nous avons présenté les problématiques de gestion des occultations et d'évitement d'obstacles et la façon dont ces deux questions ont été traitées dans le cadre de la navigation autonome.

CHAPITRE 2

Méthode de Détection et de Suivi des Personnes basée sur la Kinect et la détection d'obstacles basée sur le laser

CHAPITRE 2 : Méthode de Détection et de Suivi des Personnes basée sur la Kinect et la détection d'obstacles basée sur le laser

➢ Introduction :

Dans ce deuxième chapitre, nous allons présenter les capteurs utilisés dans notre travail :

- Une première partie est consacrée à la présentation du système de vision utilisé qui est la Kinect de Microsoft, nous présentons ses caractéristiques matérielles suivies de la méthode dont se fait son calibrage.
- Dans la deuxième partie, nous présentons la reconnaissance physique basée sur la Kinect ainsi que le suivi de personnes basé sur la vision artificielle.
- Enfin, la troisième partie est dédiée à la présentation du télémètre laser pour la détection des obstacles.

2.1 Système de vision :

2.1.1 Description matérielle de la Kinect de Microsoft :

Le mot « Kinect » est issu des mots anglais « *kinetic* » (qu'on peut traduire par « cinétique ») et « *connect* » (qu'on peut traduire par « connecter »).

La Kinect est une caméra utilisant des techniques d'interaction par commande vocale, reconnaissance de mouvement et d'image.

Ce périphérique de Microsoft est constitué d'une barre horizontale connectée à sa base via un petit moteur. La barre horizontale constitue l'élément principal de la technologie Kinect. Elle est équipéed'une série de multi-microphones, d'une caméra RGB, et enfin d'un capteur de profondeur. L'exploitation de ces trois dispositifs permet la reconnaissance vocale et faciale de l'utilisateur, ainsi que la « capture » de l'ensemble de la pièce (et donc du corps) en 3D.

Caractéristiques en détails

- Capteur :
    - Lentilles détectant la couleur et la profondeur
    - Micro à reconnaissance vocale
    - Capteur motorisé pour suivre les déplacements
- Champ de vision :
    - Champ de vision horizontal : 57 degrés

- Champ de vision vertical : 43 degrés
- Marge de déplacement du capteur : ± 27 degrés
- Portée du capteur : 1,2 m – 3,5 m (à partir de 50 cm pour la version Kinect pour Windows)

- Flux de données :
  - 320 × 240 en couleur 16 bits à 30 images par seconde
  - 640 × 480 en couleur 32 bits à 30 images par seconde
  - Audio 16 bits à 16 kHz
- Système de reconnaissance physique :
  - Jusqu'à 6 personnes et 2 joueurs actifs (4 joueurs actifs avec le SDK 1.0)
  - 20 articulations par squelette
  - Application des mouvements des joueurs sur leurs avatars Xbox Live
- Audio :
  - Chat vocal Xbox Live et chat vocal dans les jeux
  - Suppression de l'écho
  - Reconnaissance vocale multilingue

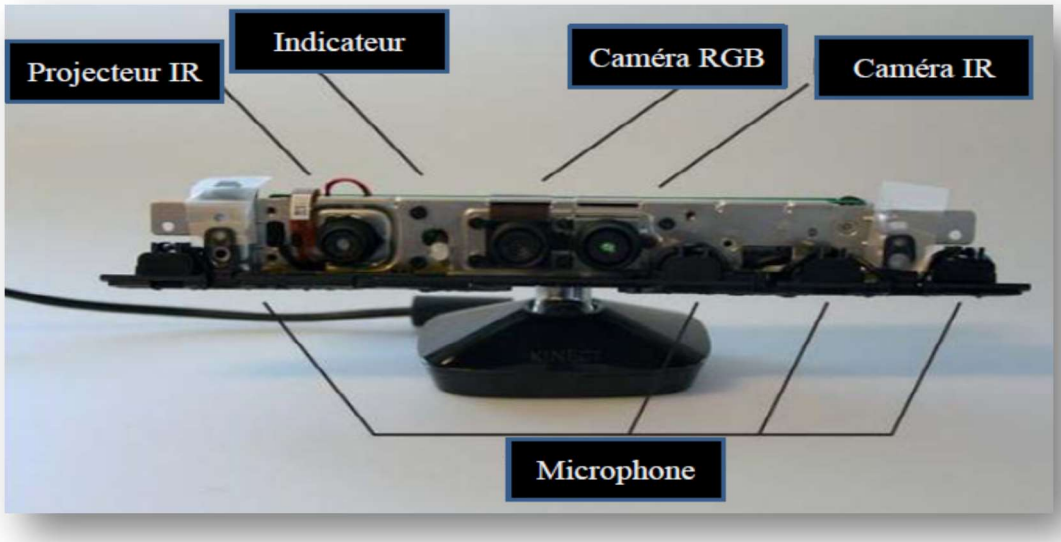


Fig.2.1 : Kinect sans cache.

2.1.2 Calibrage d'un capteur de vision stéréoscopique :

L'opération de calibration d'une caméra revient à modéliser le processus de formation des images, il consiste à déterminer la relation mathématique existant entre les coordonnées des points 3D de la scène observée et les coordonnées 2D de leur projection dans l'image (points-image). Cette étape de calibrage constitue le point initial pour plusieurs applications de la vision artificielle, comme par exemple la reconnaissance et la localisation d'objets, le contrôle dimensionnel de pièces, la reconstruction de l'environnement pour la navigation d'un robot mobile, etc.

Plusieurs modèles décrivant le processus de formation des images existent. Le plus simple est le modèle du sténopé ou modèle du trou d'épingle (*pinhole* dans la littérature anglo-saxonne). Ce dernier est couramment utilisé en traitement d'image. Ce modèle est développé en détail en **Annexe B**.

2.1.3 Généralités sur la Reconnaissance de Personnes dans une Scène :

La vision par ordinateur (aussi appelée vision artificielle ou vision numérique) est une branche de l'intelligence artificielle dont le principal but est de permettre à une machine d'analyser, traiter et comprendre une ou plusieurs images prises par un système d'acquisition (par exemple : caméras).

La détection de personnes est un domaine de la vision par ordinateur consistant à détecter un humain dans une image numérique. C'est un cas particulier de détection d'objet, où l'on cherche à détecter la présence et la localisation précise, dans une image, d'une ou plusieurs personnes, en général dans une posture proche de celle de la situation debout ou demarche. On parle également de détection de piéton, en raison de l'importance des applications en vidéosurveillance et pour les systèmes de vision embarqués dans des véhicules.

Étudiée à partir de la fin des années 1990, la détection de personne s'est révélée être un sujet assez difficile, en raison de la grande variété d'apparences des personnes, de l'articulation du corps humain et des phénomènes d'occultations. Bénéficiant des progrès méthodologiques réalisés en détection de visage, la détection de personne a inspiré des méthodes spécifiques, comme les histogrammes de gradient orienté, particulièrement performants. Les méthodes les

plus efficaces construisent des modèles statistiques par apprentissage supervisé, à partir de caractéristiques de forme ou d'apparence, calculées sur de nombreux exemples d'images de personnes.

2.1.4 Reconnaissance physique basée sur la Kinect :

L'exécution de la tache de suivi d'une cible par asservissement visuel passe par l'étape de la reconnaissance des points décrivant la cible parmi le nuage 3D observé par la Kinect. Il est à noter que la Kinect détecte un squelette (ensemble de points particuliers liés à une personne) en guise de personne. La Kinect, mise à notre disposition, peut détecter au maximum jusqu'à six squelettes de personnes assises ou debout. Cette caractéristique représente l'une des principales limites actuelles du périphérique. Parmi ces six squelettes, deux seulement peuvent être reconnus totalement : on parle alors de squelettes « trackés » et les quatre autres sont dits squelettes « non-trackés ». L'image de la **Figure 2.2** illustre cette fonctionnalité.

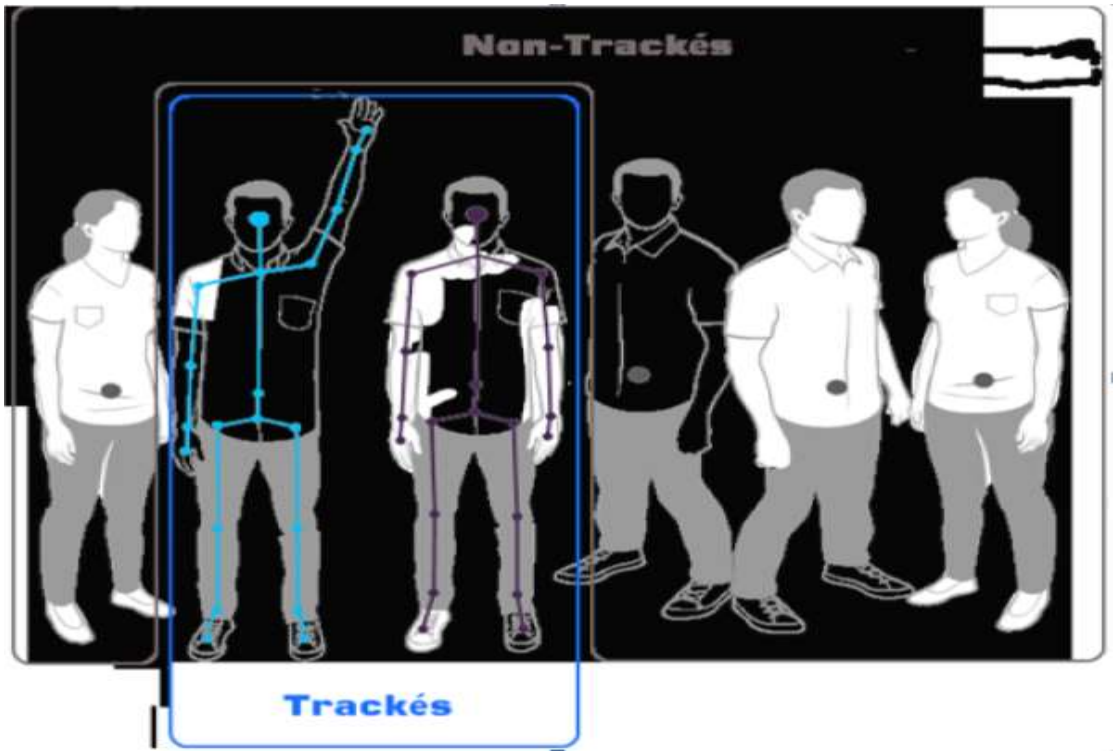


Fig.2.2 Squelettes « trackés » et « non-trackés »

De la Figure 2.2, nous constatons que les squelettes « non-trackés » sont gérés différemment par rapport à ceux « trackés ». Dans le premier cas, la Kinect reconnait un seul point

correspondant à la position courante du squelette dans son repère alors que dans le second cas, elle détecte 20 points représentant le squelette de la personne détectée : c'est pourquoi nous parlons de squelettes et non de personnes.

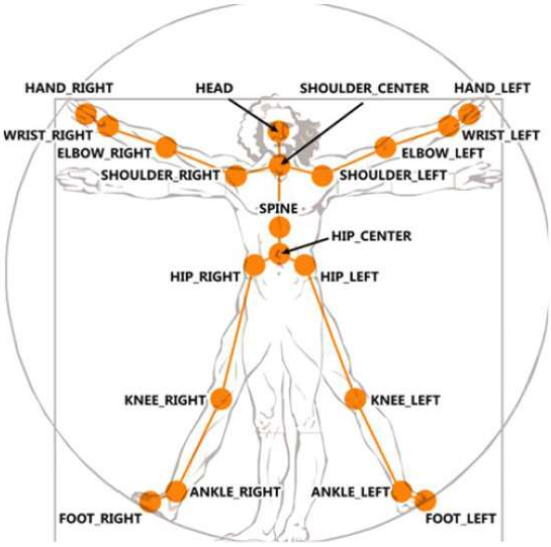


Fig.2.3 Représentation d'un squelette tracké

Ces points sont appelés « joints ». Ils correspondent pour la plupart d'entre eux aux articulations de l'homme et constituent le squelette de la personne.

La **Figure 2.3** détaille les 20 joints d'un squelette tracké. Dans le cas d'un squelette non-tracké, un seul joint est pris en compte : HIP_CENTER. Chaque joint, associé à un squelette, est identifié par sa position en X, Y et Z dans le repère 3D ainsi que son orientation par les quaternions dans ce même repère 3D.

2.2 Le suivi de personnes :

2.2.1 Suivi basé sur la vision artificielle :

Le suivi d'objets est une tâche fréquemment rencontrée en vision par ordinateur. Le suivi d'objets a pour objectif la détermination des trajectoires de ces objets dans le plan image et d'assigner à chaque objet de la scène une étiquette consistante dans le temps. Le suivi d'humains est une tâche difficile pour plusieurs raisons particulières :

- les personnes suivies peuvent avoir des mouvements complexes et difficilement prévisibles ;
- le corps humain est fortement articulé ;

- De nombreuses occultations peuvent survenir (de la personne par elle-même, par les autres objets en mouvement ou par des objets de l'arrière-plan) ;
- Des changements d'illumination de la scène peuvent entraîner une non-consistance des valeurs des pixels représentant une personne.

D'une manière générale, il existe dans la littérature deux approches majeures pour réaliser le suivi. La première approche consiste à rechercher dans l'image courante l'objet présent sur les images précédentes [Com 00]. Il se pose alors le problème de l'initialisation et de la terminaison du chemin. La deuxième approche consiste à détecter les objets sur l'image courante et à les associer avec les objets présents à l'instant précédent [Har 00, Sta 99]. Ces méthodes peuvent être également classées en fonction de leur environnement de travail, de leur système de vision (monoculaire ou stéréovision [Tan 00]), du spectre utilisé par la caméra (visible ou infrarouge [Yas 05])... Néanmoins, ces méthodes se décomposent toutes en deux parties : le choix d'une représentation des personnes suivies et une méthode de suivi.

2.2.2 Les différentes représentations des personnes :

Les représentations possibles sont nombreuses (**Fig.2.4**), nous pouvons citer par exemple :

- Le centroïde de la personne [Dav 93],
- Un ensemble de points [Meu 07, Tsu 09],
- Des primitives géométriques plus ou moins complexes, par exemple une simple boîte englobante,
- La silhouette ou le contour de la personne [Kim 06, Xia 08],
- Un modèle articulé [Zha 06]; dans cette représentation, plusieurs parties du corps sont modélisées par une primitive géométrique (par exemple une ellipse) et sont reliées entre elles par des connexions,
- Un squelette [Har 00],
- Une représentation statistique avec des densités de probabilité, par exemple utilisation d'un histogramme des couleurs [Per 02, Ziv 04],
- Des descripteurs basés sur l'avant-plan, par exemple les moments de Zernike [Hun 95],
- Le mouvement, avec par exemple l'utilisation du flot optique [Den 07].

Le choix de la représentation est crucial car les modèles de déplacement qui seront utilisées en dépendront. Les informations récupérées à partir du déplacement d'un centroïde ou des déplacements des parties d'un modèle articulé n'auront pas la même importance.

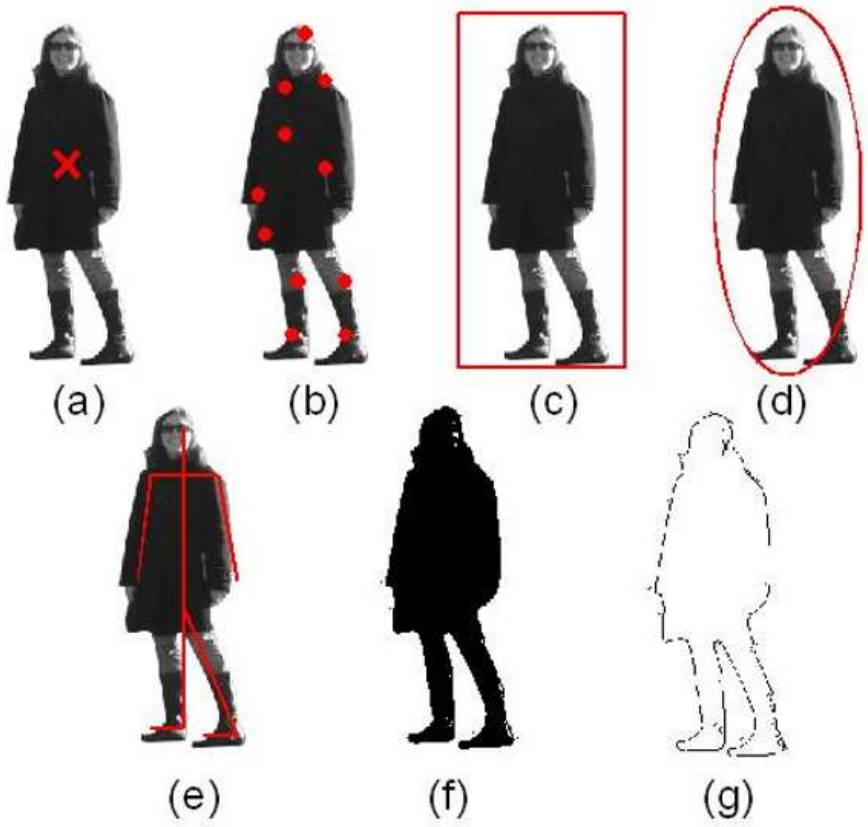


Fig.2.4Illustration de quelques représentations utilisées pour le suivi : (a) le centroïde ; (b) un ensemble de points caractéristiques ; (c) une boîte englobante ; (d) une ellipse ; (e) un squelette ; (f) un masque (ou silhouette) ; (g) un contour.

2.2.3 Méthodes de suivi :

Les méthodes de suivi dépendent directement de la représentation de la personne. Pour les méthodes où la représentation choisie est un ensemble de points, la première façon de faire la correspondance entre les points sur l'image à l'instant t et les points présents sur l'image à l'instant t-1 est de formuler le problème comme étant un problème d'optimisation [Ran 91, Vee 01, Sha 05]. On recherche alors pour chaque point son correspondant optimal. Les contraintes utilisées peuvent être la distance, la régularité de la vitesse, la rigidité (les mouvements des points proches doivent être similaires), l'homogénéité des mouvements des

points d'un même objet etc. Il est également possible de faire la correspondance de manière statistique (et non plus déterministe comme expliqué ci-dessus). La méthode qui est la plus utilisée est sans conteste le filtre de Kalman [Liu 07]. Partant de l'hypothèse que le bruit de mesure suit une distribution gaussienne, le filtre de Kalman est un algorithme récursif en deux étapes, la prédiction et la correction. Des versions modifiées comme le filtre de Kalman étendu [Nic 01] ont également été proposées dans la littérature et permettent de prendre en compte des modèles non-linéaires. Il a récemment été proposé une méthode basée sur les estimateurs à horizon glissant dont la particularité est de prendre en compte une fenêtre temporelle pour l'estimation et également de pouvoir gérer les occultations comme étant des contraintes visuelles [Bru 09]. Le filtre à particules [Jun 08] permet lui de considérer les cas où les variables ne suivent pas une distribution normale.

Si la représentation utilisée est basée sur la forme ou sur des densités de probabilité, la méthode la plus simple, mais aussi la plus coûteuse en terme de calcul est le simple *templatematching* [Lin 07] en calculant par exemple l'inter-corrélation. On peut également dans ce cas utiliser la méthode du *mean shift* [Com 00], cette méthode présentant l'avantage par rapport au *templatematching* d'optimiser la phase de recherche du correspondant optimal. En se basant sur le même raisonnement, si on utilise une représentation du contour, il est possible de réaliser une correspondance simple entre les contours en calculant par exemple la distance de *Hausdorff* [Aya 00]. Ensuite, Shi et al. [Shi 94] présentent une méthode basée sur le flot optique. Se basant sur des contraintes d'absence de variation de la luminance, ils calculent le vecteur déplacement d'un pixel ou l'étendent à une région de pixels.

2.3 Télémètres à Laser :

2.3.1 Généralités :

Les capteurs de distance Hokuyo sont des détecteurs d'obstacles 2D. Ils peuvent être utilisés dans des endroits fixes pour détecter l'apparition d'un objet dans leur zone de numérisation, et peuvent être embarqués sur des plates-formes mobiles pour leur assurer la détection et de l'évitement d'obstacles.

Il existe deux catégories de capteurs : les capteurs extérieurs sont étanches et ils ont une portée jusqu'à 30 mètres (ou 80 mètres); et les capteurs d'intérieur plus petits/légers mais ils sont d'une plus grande précision.

Selon le modèle, la communication peut se faire soit par USB, RS232, Ethernet, et/ou des signaux d'entrée/sortie numériques simples, et la tension d'alimentation peut être de 5V, 12V, 12/24V, 10-30V, ou celle de l'USB.

2.3.2 Présentation du Télémètre Laser :

a) Définition et différentes classes :

Un télémètre laser est un appareil permettant de mesurer les distances. En effet, un rayon modulé en fréquence est projeté sur une cible. La cible renvoie le rayon réfléchie vers l'appareil. Le temps, mis par le rayon pour revenir au boitier émetteur, est mesuré lequel va servir pour calculer la distance séparant le boitier émetteur de la cible.

Le laser utilisé est un laser de classe 1, c'est à dire qu'il ne pose aucun problème de sécurité.

Rappel de la classification des lasers :

- Classe I : Lasers sans dangers, regroupant les produits tels que les imprimantes lasers, les lecteurs de CD et la plupart du temps les produits où l'émetteur laser est inséré dans le produit.
- Classe II : Lasers à rayonnement visible où la protection de l'œil est assurée par le réflexe palpébral (ex : lecteurs de code barre).
- Classe III : Lasers dont la vision directe est dangereuse pour l'œil immédiatement (class3b) ou dans le cas d'une vision directe supérieur à 0,25s (classe 3a). On retrouve dans ces classes certains pointeurs lasers.
- Classe IV : Lasers dangereux pour la vision (en vision directe ou diffuse), pouvant créer des lésions et également susceptibles de générer des incendies. On retrouve là des Lasers industriels et des lasers pour la recherche.

b) Principe de fonctionnement :

Une onde lumineuse se propage en ligne droite, rebondit sur un obstacle et revient vers le télémètre laser. Celui-ci compare l'onde reçue et l'onde émise et calcule la différence de phase entre ces deux ondes. Cette différence de phase est en effet proportionnelle au temps mis par la lumière pour aller du capteur vers l'obstacle et en revenir et ce temps est lui-même proportionnel à la distance parcourue. La technologie laser est très efficace car l'onde

lumineuse se réfléchit sur toutes les surfaces solides avec une diffusion réduite quel que soit la nature de l'obstacle.

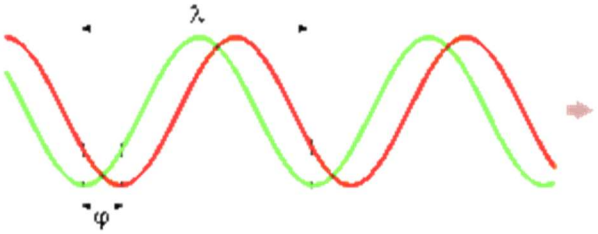


Fig.2.5 Déphasage de deux ondes de même fréquence

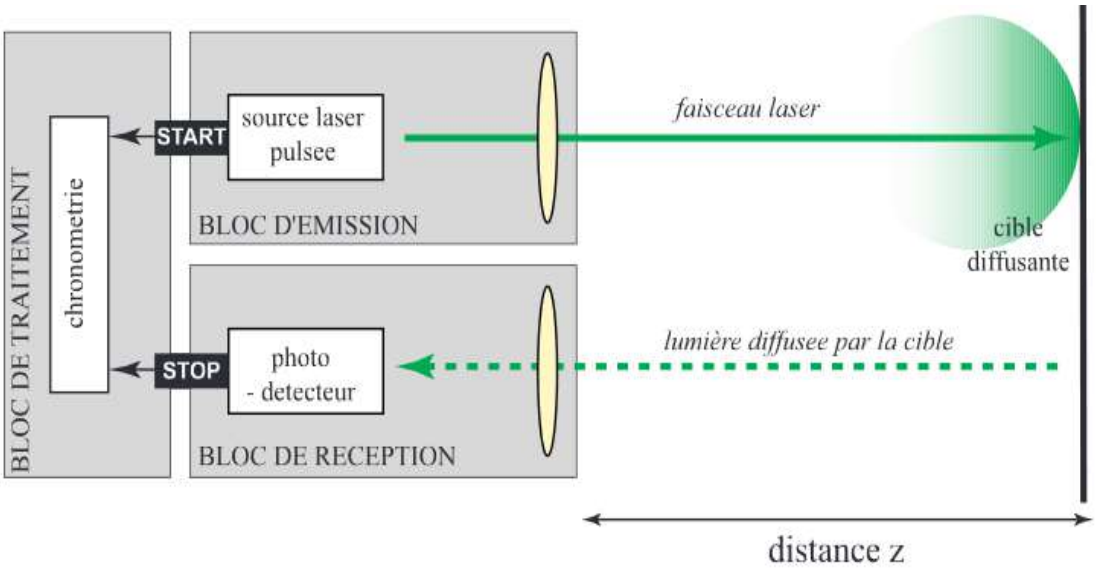


Fig.2.6 Principe d'un télémètre laser basé sur le temps de vol

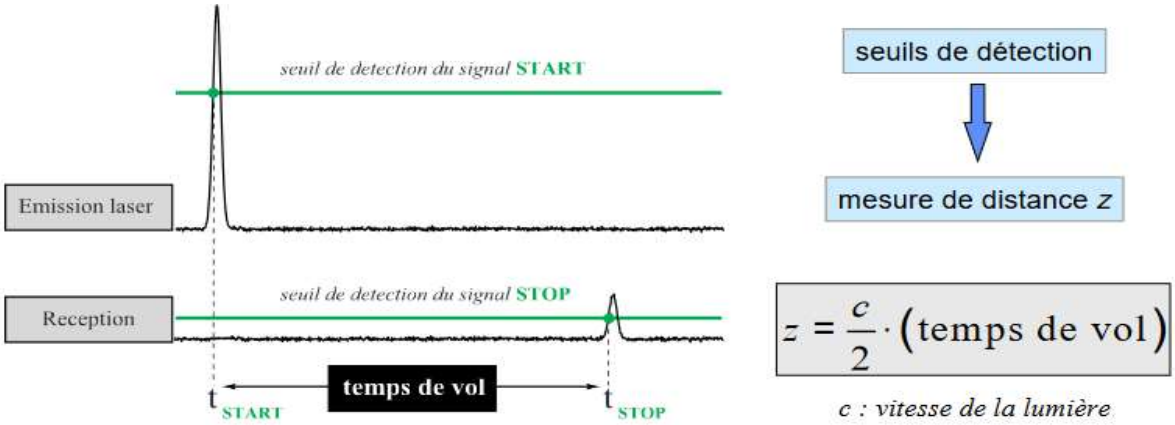


Fig.2.7 Technique de calcul de la distance

2.3.3 Détection sur un plan à l'aide du télémètre laser :

Le capteur émet des faisceaux laser pulsés dans un champ de vision de 270°. Lorsque les faisceaux laser émis sont réfléchis par un objet, sa distance est mesurée en appliquant la technique basée sur le temps de vol (TOF). Il fait un balayage du champ de vision avec un pas de 0.5°. Afin de prendre des mesures suivant un plan, le capteur intègre un petit miroir pouvant pivoter de 0.5° entre chaque mesure.

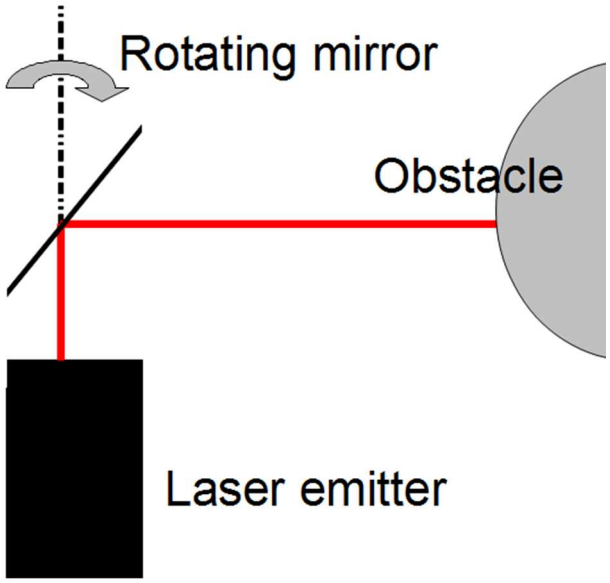


Fig.2.8 Balayage suivant le plan du laser

L'utilisateur peut présélectionner la région de détection au niveau du capteur. Le champ balayé par le capteur est segmenté en trois zones. Lorsque le capteur détecte un objet dans une zone, la sortie correspondante est activée à l'état OFF.

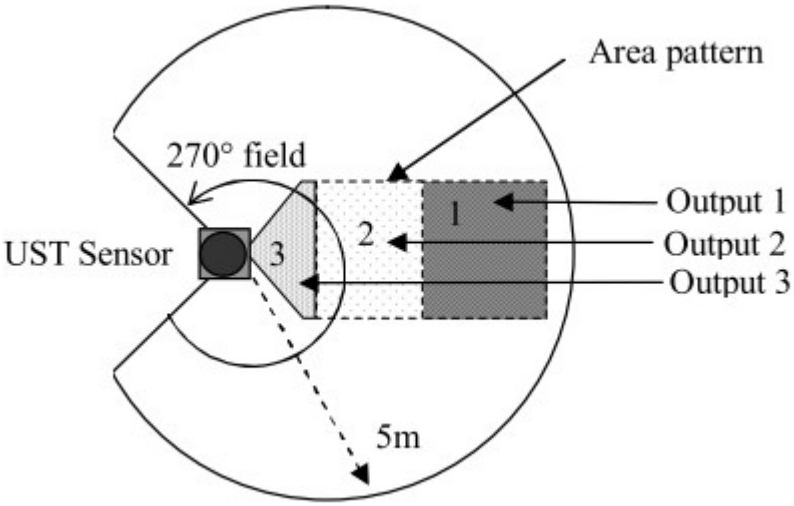


Fig.2.9 Champ de balayage du laser

a) Configuration de la région :

L'utilisateur peut configurer des zones au niveau du capteur en utilisant un logiciel d'application fourni par le fabricant. Ce logiciel doit être installé dans un système d'exploitation qui le prend en charge et la connexion du capteur se fait à l'aide d'un câble Ethernet.

b) Dysfonctionnement de la sortie :

Le capteur a une fonction d'auto-diagnostic. Il commute la sortie de défaut à un état OFF lorsque des erreurs sont détectées dans ses composants internes.

c) Hystérésis de la zone de détection :

Lorsque des objets sont présents sur la limite d'une zone, le capteur ne peut pas les détecter en continu. Dans de tels cas, le signal de sortie oscille souvent entre l'état ON et l'état OFF. Le capteur disposant de la fonction d'hystérésis peut alors augmenter temporairement la taille de la zone pour empêcher une telle oscillation. Le ratio d'augmentation de la zone peut être spécifié en utilisant le logiciel d'application.

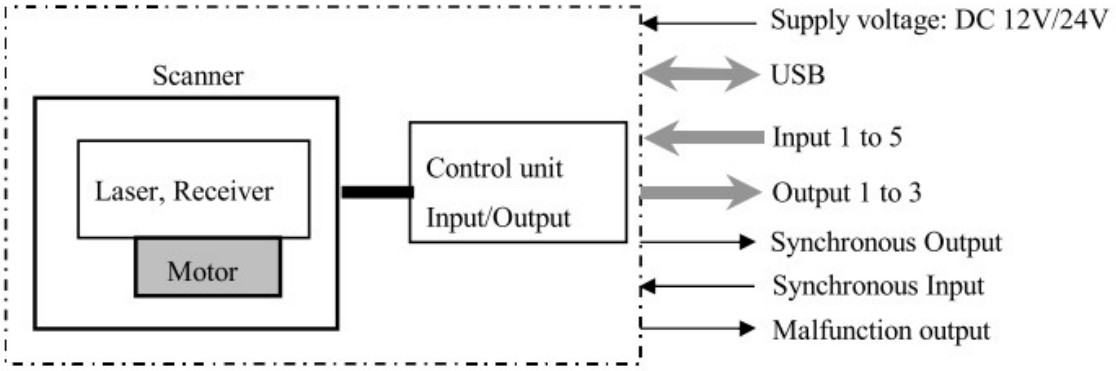


Fig.2.10 Diagramme de la structure du laser

➢ Conclusion

L'exécution de la tache de suivi d'une cible par asservissement visuel passe par l'étape de la reconnaissance des points décrivant la cible parmi le nuage 3D observé par la Kinect. Dans notre cas, le suivi de la cible est basé sur la détection du squelette de cette dernière dont nous collectons les coordonnées des articulations.

Aussi, le télémètre laser est un appareil permettant de mesurer les distances, et plus particulièrement dans notre contexte, il va permettre de mesurer la distance qui sépare le robot des obstacles qu'il va rencontrer au cours de l'asservissement.

CHAPITRE 3

Architecture logicielle et Implémentation des programmes

➢ Introduction :

Dans ce chapitre, nous allons décrire dans une première partie, l'architecture logicielle de la plateforme robotique B21r et le système qui la contrôle, à savoir le rFlex, pour ensuite passer à une représentation d'ensemble de l'OS utilisé ROS (Robot Operating System).

Dans la deuxième partie, nous avons présenté la hiérarchie des programmes implémentés qui répondent aux problématique traitées : asservissement visuel, gestion de l'occultation et enfin l'évitement d'obstacle.

3.1 Architecture software du B21r :

3.1.1 Système rFLEX :

La plate-forme B21r est contrôlée par le système rFLEX de la société *iRobot* fournissant des contrôles standardisés et une interface utilisateur facile à utiliser et conviviale permettant l'interaction avec l'utilisateur à travers un ensemble de menus permettant la gestion des capteurs ultrasonores, les moteurs et autres systèmes embarqués. Ce système est considéré comme le système nerveux central de ce robot mobile où les câbles, nœuds et Hubs son facile à configurer et son réseau est très optimisé pour lier les capteurs embarqués et les actionneurs à son ordinateur de traitement central (cerveau).

Le contrôle software se fait par *Mobility*, logiciel distribué, une boite à outils orientée objet pour la construction de programmes de contrôle d'un seul ou plusieurs robot de la famille *iRobot*. Il consiste en :

- Un ensemble d'outils logiciel,
- Un modèle objet pour le logiciel du robot mobile,
- Un ensemble de modules de contrôle de base du robot mobile,
- Une structure de classes orientées objet pour simplifier le développement du code.

Le logiciel *Mobility* utilisé basé sur CORBA, collecte les données en provenance des capteurs embarquées, les interprète et les exécute. En utilisant CORBA tous les capteurs et actionneurs peuvent être vus comme des ressources Internet. C'est une librairie composée de programmes écrits en C et C++ contrôlant le hardware des robots mobiles.

La connexion avec les différents capteurs embarqués se fait en exécutant le programme serveur qui réagit réciproquement avec le capteur. Ce denier, se connectera à son tour au name server.

Le capteur peut être donc connecté et contrôlé par un simple programme écrit en C incluant les fonctions et procédures fournis avec*Mobility*.

L'exécution de notre tâche nécessite un PC externe ayant des caractéristiques lui assurant d'intégrer les informations issues de la Kinect, du laser et l'utilisation de l'outil robotique ROS. A ce PC externe a été connecté la Kinect via une connexion USB, le laser via une connexion Ethernet et le système rFlex via une liaison RS232.

Pour ce faire, il a fallu adapter l'architecture software afin d'intégrer les programmes assurant la communication avec le système rFlex. Ce dernier transmet, à son tour, les vitesses élaborées par les programmes de commande aux cartes de commande des moteurs afin que le robot puisse exécuter la tâche d'asservissement.

Le schéma suivant met en évidence les différents composants de la plate-forme robotique

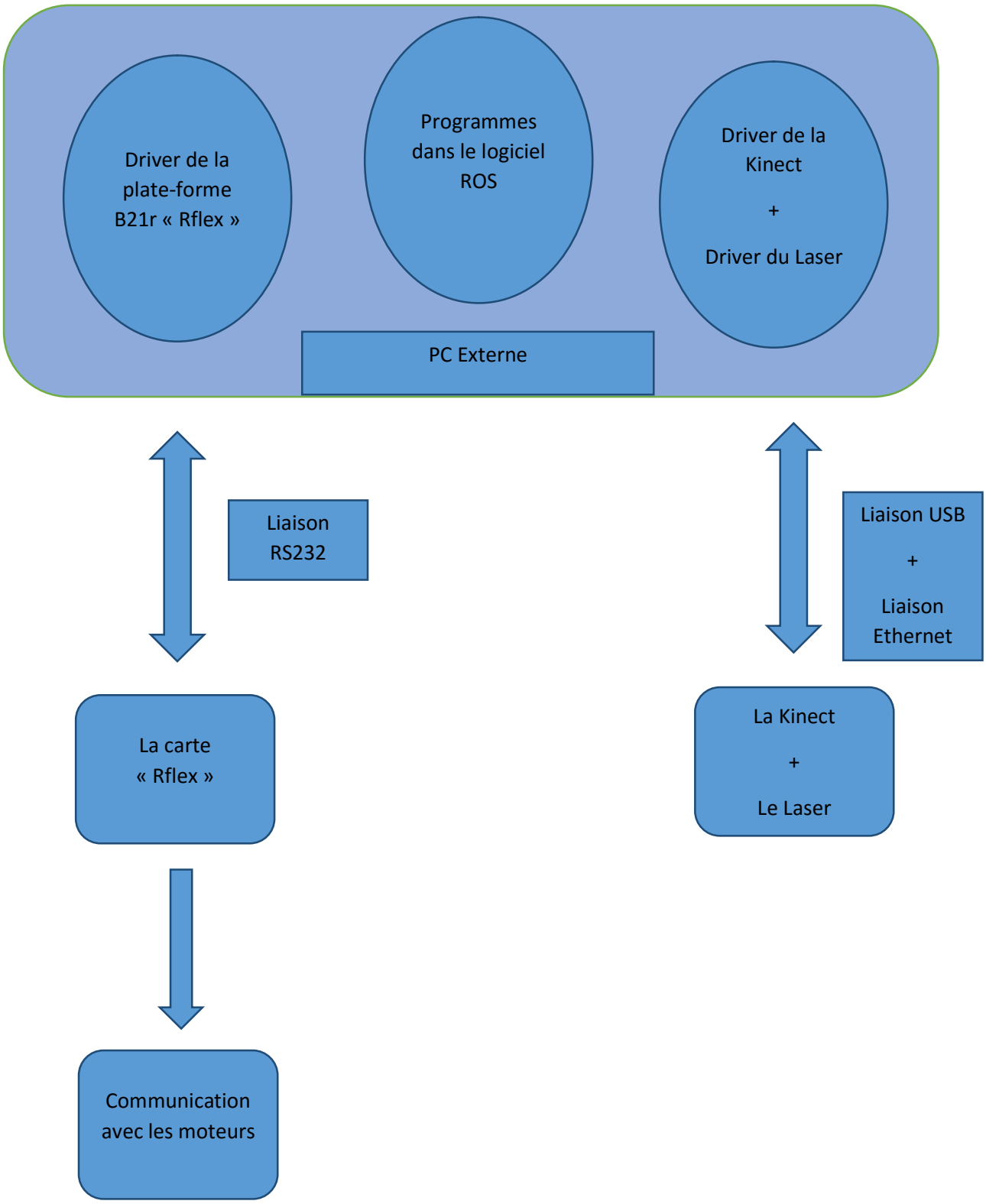


Fig 3.1 Organigramme de l'architecture globale du robot

3.2 Présentation du logiciel ROS :

Robot Operating System (ROS) est une plateforme de développement logicielle pour robot. Il s'agit d'un méta-système d'exploitation (entre un OS et un middleware) qui peut fonctionner sur un ou plusieurs ordinateurs et qui fournit plusieurs fonctionnalités : abstraction du matériel, contrôle des périphériques de bas niveau, mise en œuvre de fonctionnalités couramment utilisées, transmission de messages entre les processus et gestions des packages installés.

Le logiciel ROS peut être séparé en trois groupes :

1. les outils utilisés pour créer, lancer et distribuer des logiciels basés sur ROS (roscore, roslaunch, catkin)
2. les clients ROS pour des langages : roscpp (C++) et rospy (python)
3. les packages contenant des programmes pour ROS utilisant un ou plusieurs clients ROS

De nombreux outils très utiles dans ROS

Comme nous l'avons dit plus haut, ROS est une collection d'outils et d'algorithmes. Certains sont très utilisés lors de la programmation, de la simulation ou de l'exécution des comportements des robots. Citons quelques outils ou algorithmes que le programmeur de ROS retrouvera souvent :

- Stage : un simulateur 2D
- Gazebo : un simulateur 3D
- Rviz : un système de visualisation 3D (contrairement à Gazebo, il n'inclut pas de moteur physique)
- Le package tf qui permet de manipuler des coordonnées et des transformations (si vous avez déjà eu à faire de la cinématique inverse et à manipuler des matrices, alors ce package va grandement vous faciliter la vie).
- Opencv : traitement d'images
- PointCloudLibrary : reconstruction d'environnement 3D à partir de mesures d'un laser

3.2.1 Architecture de communication

ROS offre une architecture souple de communication interprocessus et inter-machine. Les processus ROS sont appelés des nœuds et chaque nœud peut communiquer avec d'autres via des topics. La connexion entre les nœuds est gérée par un master et suit le processus suivant :

1. Un premier nœud averti le master qu'il a une donnée à partager
2. Un deuxième nœud averti le master qu'il souhaite avoir accès à une donnée
3. Une connexion entre les deux nœuds est créée
4. Le premier nœud peut envoyer des données au second

La figure (3.2) illustre ce processus.

Un nœud qui publie des données est appelé un Publisher et un nœud qui souscrit a des données est appelé un subscriber. Un nœud peut être à la fois Publisher et subscriber. Les messages envoyés sur les topics sont pour la plupart standardisés ce qui rend le système extrêmement flexibles.

ROS permet une communication inter-machine, des nœuds s'exécutant sur des machines distinctes, mais ayant connaissance du même master peuvent communiquer de manière transparente pour l'utilisateur.

Le principe de base d'un OS robotique est de faire fonctionner en parallèle un grand nombre d'exécutable qui doivent pouvoir échanger de l'information de manière synchrone ou asynchrone. Par exemple, un OS robotique doit interroger à une fréquence définie les capteurs du robot (capteur de distance à ultrasons ou infrarouge, capteur de pression, capteur de température, gyroscope, accéléromètre, caméras, microphones…), récupérer ces informations, les traiter (faire ce que l'on appelle la fusion de données), les passer à des algorithmes de traitement (traitement de la parole, vision artificielle, localisation et cartographie simultanée,…) et enfin contrôler les moteurs en retour. Tout ce processus s'effectue en continue et en parallèle. D'autre part, l'OS robotique doit assurer la gestion de la concurrence afin d'assurer l'accès efficace aux ressources du robot.

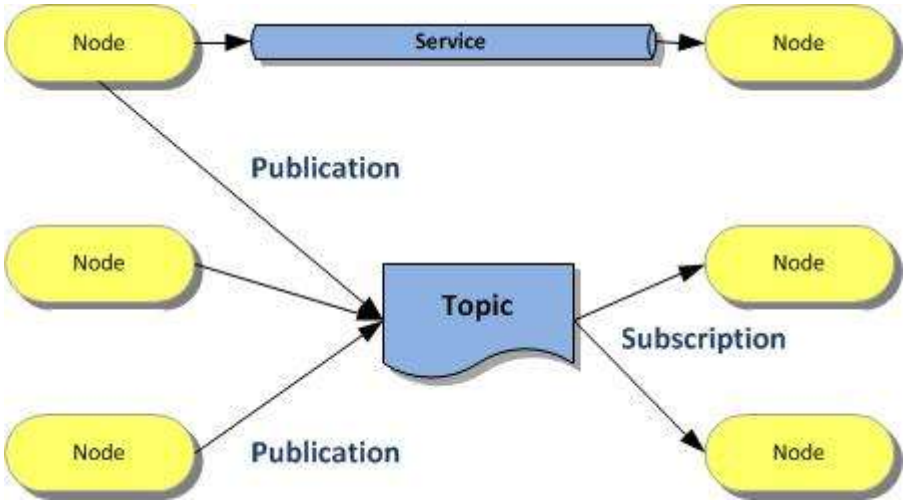


Fig3.2 Structuration de l'environnement ROS

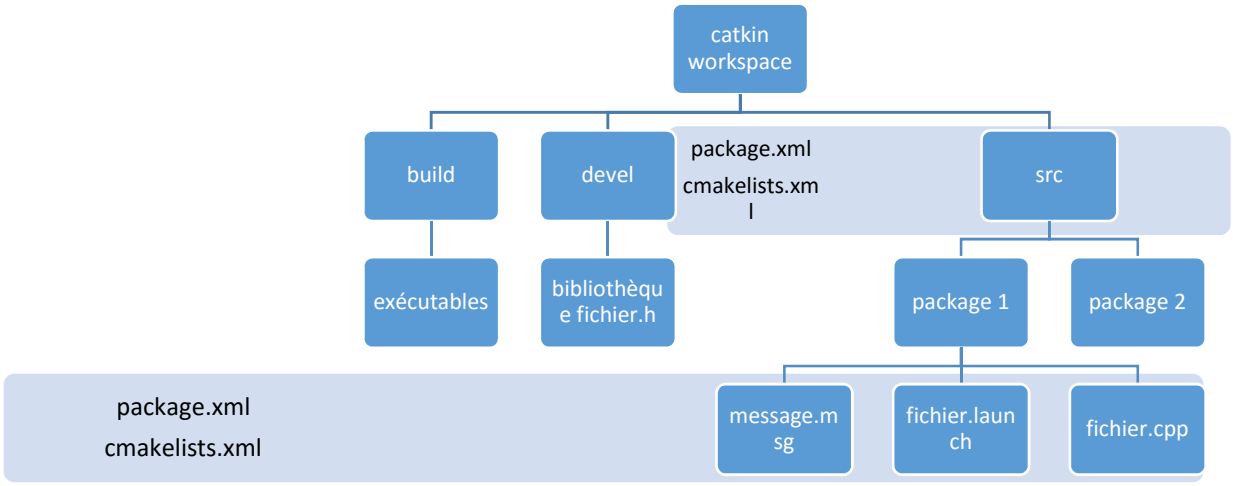


Fig3.3 Schéma récapitulatif de la hiérarchie de ROS

3.3 Programmes implémentés :

Nous présentons dans cette partie les différents programmes conçus et utilisés.

3.3.1 Le package Kinect_suivi_av :

Ce package traite la partie perception dans la hiérarchie des programmes, il a été fait en modifiant le package d'openni_tracker ; il contient un programme, kinect_suivi.cpp qui est chargé de collecter les données de la Kinect et de fournir les coordonnées 2D, métrique et 3D du squelette humain perçu par cette dernière.

Dans ce programme, un topic a été créé et nommée Kalman_Filter afin de publier un message nommée position_k qui contient les coordonnées 2D (x et z) du torse de la cible, vers le package filtre_de_kalman (dont il sera détaillé le rôle ci-après).

3.3.2 Le package urg_node

Ce package permet de traduire les données du laser hokuyo, à travers le programme urg_node.cpp de collecter l'emplacement des obstacles dans la cham scannée par le laser, ce balayage du champ par le laser est transféré au package tentacles_avoidance afin d'appliquer la tâche d'évitement d'obstacles.

Dans ce programme, le topic Scan, permet de publier ces données collectées à travers un message du même nom.

3.3.3 Le package hector_slam :

Ce package est utilisé pour récupérer la position du robot au cours de la navigation. Cette position est nécessaire et est utilisée dans le package filtre_de_kalman afin de pouvoir faire la prédiction de la position de la cible (algorithme du filtre de kalman).

Cette position est publiée à travers le topic odom dans un message nommé nav_msgs, le message contient les coordonnées (x, y, z) ainsi que les quaternions (Q_x, Q_y, Q_z, w) qui décrive la translation et la rotation du robot.

Le package filtre_de_kalman récolte cette information et l'utilise dans son algorithme d'estimation.

3.3.4 Le package filtre_de_kalman :

Ce package est la base de la hiérarchie des programmes, puisqu'il est le package qui traite la gestion des occultations de la cible si cette dernière est occultée sinon, il se chargera de transférer les positions perçues par al Kinect sans avoir à les traiter. Il contient le programme, kalman_2d.cpp, qui est chargé de recevoir les positions du torse, à travers le message position_k, et, soit de les transférer sans traitement au package de la loi de commande, si la cible est dans le champ de vision de la caméra. Soit, d'utiliser ces même coordonnées pour faire une estimation de la position suivante de la cible, et de les transférer aussi au package de la loi de commande, si la cible venait de sortir du champ de vision ou est occulter par un obstacle, qui ne permet pas de détecter le torse.

Dans ce programme, trois topics ont été utilisés afin de recevoir les données de la Kinect du package hector_slam d'une part, et de publier les positions de la cible dans le package commande_av d'une autre part.

Les topics utilisés sont nommés comme suit :

- Kalman_filter : une inscription à ce topic permet de récupérer les coordonnées 2D du point torse de la personne suivie, il s'agit du même topic cité dans le package kinect_suivi_av, qui dans l'autre cas publie les données.
- odom : une inscription à ce topic permet de récupérer la position du robot au cours de la navigation ce dernier, cette position est utilisée dans l'algorithme d'estimation de la position de la cible au cours du temps.
- Loi_filtre : ce topic permet la publication d'un message nommée position qui contient la position de la cible après exécution de l'algorithme (filtre de kalman), et qui va être lu par le package commande_av.

3.3.5 Le package commande_av :

Dans ce package, la loi de commande conçue au chapitre 2 est implémentée. Il contient un programme, loi_de_commande.cpp, dont le rôle est de calculer et d'envoyer les vitesses de rotation et de translation au robot en se basant sur les données reçues par le package filtre_de_kalman.

Le package utilise les topics suivant :

- Loi_filtre : une inscription à ce topic permet de récupérer la position de la cible qui a été calculée dans le package filtre_de_kalman, afin de l'utiliser dans l'algorithme de calculs des vitesses.
- Visual_vel : ce topic permet la publication des vitesses de rotation et de translation du robot à travers un message nommé Twist vers le package tentacles_avoidance, un package qui traite l'évitement d'obstacles. Ces vitesses vont être suivies dans le cas où le champ est libre (pas d'obstacles qui gênent le suivi de la cible) comme cela a été expliqué au chapitre précédent, dans la partie évitement d'obstacles.

3.3.6 Le package tentacles_avoidance :

Dans ce package, le principe d'évitement d'obstacles expliqué au chapitre 2, est implémenté. Il contient un programme nommé tentacles_avoidance.cpp qui récupère des données du laser (package urg_node) et de la loi de commande (package commande_av) afin d'exécuter l'algorithme, pour ensuite publier les vitesses adéquates qui vont soit être celle déjà calculées par la loi de commande du package commande_av, soit de nouvelles vitesses calculées à cause de la présence d'obstacles, ces dernières sont transférées vers le package Rflex qui communique avec les actionneurs du robot.

Le package utilise les topics suivant :

- Scan : l'inscription à ce topic permet de récupérer les données du laser à travers un message nommé LaserScan et qui va permettre de détecter les obstacles.
- Visual_vel : l'inscription à ce topic permet de récupérer les vitesses calculées par la loi de commande dans le package commande_av et qui vont être directement publiée vers le package rflex si jamais il n y a pas d'obstacles en vue.
- Cmd_vel : ce topic permet la publication des vitesses à travers un message nommée Twist, ces vitesses sont calculées en présence d'obstacles qui vont gêner le suivi de la cible, elles vont causer la sortie du robot de la trajectoire de suivi.

3.3.7 Le package rFlex :

Ce package a été développé par la communauté ROS afin d'assurer la communication entre les différents nœuds contenues dans le PC externe et la carte rFlex pour que cette dernière puisse piloter le robot. Ce package contient essentiellement deux nœuds rFlex_driver.cppet b21_node.cppdont la fonction est de récupérer et d'envoyer les données à différents sous-

systèmes à travers plusieurs topics. Par exemple, le sous-système des moteurs reçoit les commandes en vitesses de rotation et de translation via le topic cmd_vel, et le sous-système capteurs (ces deux nœuds) récupère l'odométrie du robot et les publie dans le topic odom. Dans notre travail, on a utilisé le sous-système des moteurs et le topiccmd_vel.

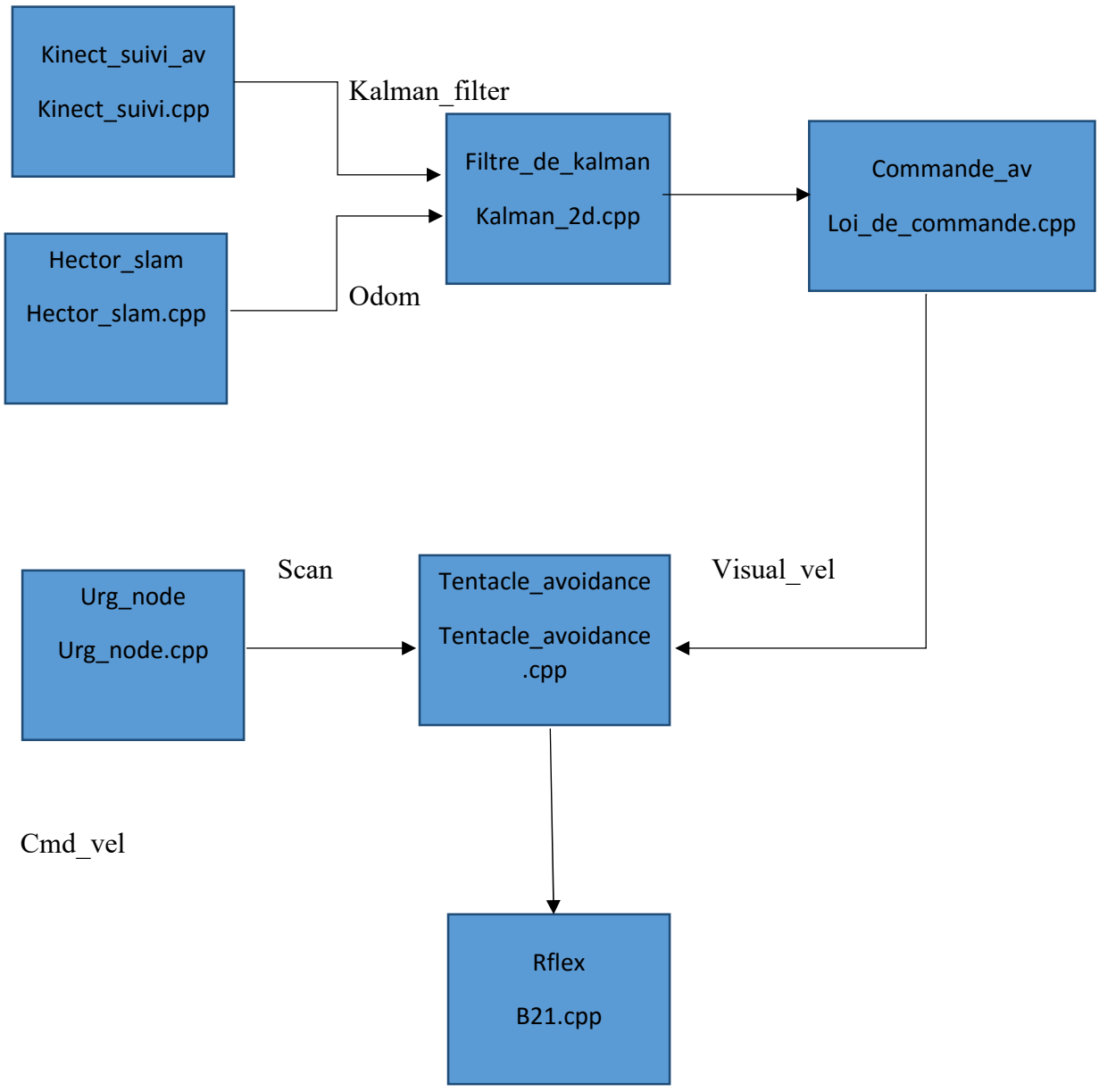


Fig3.4 Schéma récapitulatif des programmes implémentés

➢ Conclusion

L'une des principales raisons d'avoir opté pour l'OS ROS est le fait qu'il puisse faire fonctionner en parallèle un grand nombre d'exécutable qui doivent pouvoir échanger de l'information de manière synchrone ou asynchrone.

Les programmes implémentés permettent d'une manière globale de faire :

- La détection des personnes
- La détection des obstacles
- Le suivi d'une cible tout en estimant sa position si elle est occultée et en évitant les obstacles s'ils se trouvent en cours de chemin.

CHAPITRE 4

Evitement d'Obstacles et Gestion de l'Occultation

➢ Introduction :

Dans ce chapitre nous allons présenter les algorithmes de l'évitement d'obstacles ainsi que celui de la gestion de l'occultation qui vont permettre au robot de naviguer dans son environnement tout en déviant en présence d'obstacles et de gérer la cas où la cible sera momentanément hors du champ de vision de la caméra.

4.1 Évitement d'obstacles par la Méthode des tentacules

4.1.1 Objectif et principe général :

Dans le cadre de notre projet, l'objectif de l'évitement d'obstacles est que le robot puisse suivre la cible tout en évitant les différents obstacles qui peuvent gêner son mouvement ou l'endommager.
Le programme de l'évitement aura donc pour entrées les vitesses angulaires et linéaires données par la loi de commande, et devra calculer des vitesses comme sorties.
Si la trajectoire menant à la cible n'est pas encombrée (il n'y a aucun obstacle) la sortie du programme d'évitement sera la même que son entrée. Si, par ailleurs, certains obstacles « dangereux » (ce terme sera quantifié dans les paragraphes suivants) se retrouvent dans la trajectoire, le programme calculera des vitesses différentes afin d'éviter cet obstacle ou éventuellement s'arrêter si le robot se trouve dans une impasse.

La stratégie suivie pour concevoir le programme d'évitement est comme suit :

- Etablir une carte (ou grille) indiquant la position de chaque obstacle dans un domaine délimité.
- Calculer différentes trajectoires (tentacules) en forme de demi-cercles de courbure différentes.
- Choisir la meilleure trajectoire de sorte à éviter les obstacles tout en restant le plus proche possible de la direction initiale (donnée par la loi de commande).

Ce travail se base sur [Cha. Et Che. 2012]

4.1.2 Conception de la grille d'occupation :

4.1.2.1 Généralités :

Dans la représentation de l'environnement sous forme d'une grille d'occupation, l'espace est partitionné en un ensemble de cellules distinctes. Un vecteur d'attributs (éventuellement un seul nombre ou un seul bit dans le cas de cartes binaires) est attaché à chacune des cellules pour représenter ses propriétés : souvent, il s'agit du degré d'encombrement par un obstacle (indice indiquant que la cellule correspondante est occupée ou non par un obstacle). Les grilles d'occupation constituent une représentation surfacique simple et populaire. Dans ce type de modèle, l'espace est discrétisé selon une grille régulière en cellules carrées ou rectangulaires de même taille. Chaque cellule contient un indice (binaire, probabilité, histogramme, etc.) indiquant si l'espace correspondant est plutôt libre ou occupé.

L'avantage principal des grilles d'occupation est leur capacité à représenter l'espace de manière très dense, en fonction du pas de discrétisation de la grille. Elles sont adaptées à des environnements de forme quelconque. De plus, elles fournissent des informations d'occupation et donc sur les positions des obstacles. Ainsi, elles sont souvent utilisées lorsque l'application visée repose sur la connaissance de l'espace libre, en particulier pour la planification de trajectoires.

Elles sont en général relativement faciles à interpréter par l'homme.

4.1.3 La représentation des obstacles par la grille d'occupation

Dans ce travail, les obstacles sont détectés grâce à un balayage du laser dans un plan parallèle au sol durant la navigation du robot. Une grille d'occupation est utilisée à la Figure 4.1a : elle est liée à (R_k) (Repère lié à la kinect) avec des côtés de cellules parallèles à X_k et Z_k. La grille est limitée suivant les deux axes ($X_m \leq X \leq X_M$ et $Z_m \leq Z \leq Z_M$), l'algorithme ignore les obstacles qui sont trop loin et qui ne pourront pas mettre en péril le robot. La taille de la grille devrait augmenter avec la vitesse du robot, afin de garantir un temps suffisant pour l'évitement d'obstacles. Un choix approprié pour | Z_m | est la longueur du robot, car les obstacles ne peuvent pas heurter le robot par derrière s'il les a déjà devancé. Dans ce travail, il est utilisé : $X_M = Z_M$ = 10 m, Z_m = -2 m, X_m = -10 m. Toute cellule de grille *c* est considéré comme occupée si un obstacle a été détecté dans *c*. Les cellules ont une taille de 0,2m x 0,2 m. Pour les cellules situées entièrement dans la zone de balayage, seule la lecture du scanner en cours est prise en compte. Pour toutes les autres cellules de la grille, des lectures antérieures sont utilisées.

Il est défini l'ensemble des cellules occupées de la grille : $O = \{c_1, ..., c_n\}$, un ensemble de chemins carrossables (les tentacules) où chaque tentacule j est un demi-cercle qui commence au niveau de la Kinect, est tangent à Z_k, et se caractérise par sa courbure K_j, qui appartient à K, un ensemble uniformément échantillonné : $K_j \in K = \{-K_M,\ldots, 0, ..., K_M\}$. La courbure $K_M > 0$ maximale sera déterminée en fonction du nombre de tentacules et de la précision désirée de l'échantillonnage, ces deux paramètres influent sur la complexité de l'algorithme. Un compromis entre le coût de calcul et de contrôle de précision doit être atteint pour régler la taille de K.

Dans la figure 4.1, un total de 3 tentacules est utilisé. Chaque tentacule j est caractérisé par trois zones (collision C_j, centrale dangereuse D_j et externe dangereuse E_j), qui sont obtenues en déplaçant de manière rigide, le long du tentacule, trois boîtes rectangulaires, avec une taille croissante. Les boîtes sont partout estimées par rapport aux dimensions réelles du robot. Les cellules appartenant à l'ensemble central dangereux, ne sont pas considérés comme appartenant à l'ensemble externe dangereux, de sorte que $D_j \cap E_j = \emptyset$. Les ensembles O, C, D et E sont utilisés pour calculer les variables requises pour le calcul des vitesses plus particulièrement, les zones de D et E sont utilisés pour sélectionner le tentacule le plus sûr et le risque qui lui est lié, alors que la zone C détermine la décélération nécessaire éventuelle (cas de la collision).

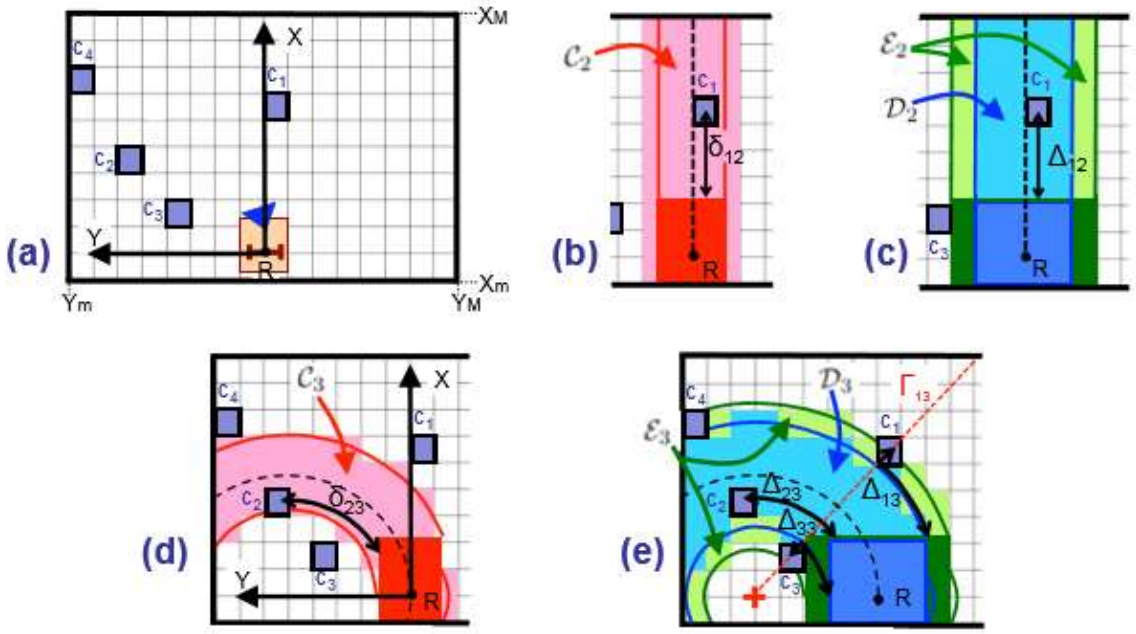


Fig.4.1 Illustration de la méthode des tentacules

4.1.4 La fonction risque et le meilleur tentacule

La fonction risque notée H est une fonction qui associe à tout tentacule « j » un nombre dans [0,1] exprimant le caractère dangereux du tentacule (i.e. de la trajectoire), 0 représentant le risque nul, et 1 le risque maximal.

Pour calculer la fonction risque H, qui nous permet de choisir le meilleur tentacule, une fonction de risque candidat $H_j \in [0,1]$ est d'abord calculée pour chaque tentacule. Chaque H_j est déduite de la « distance de risque » de toutes les cellules occupées dans les zones C_j, D_j et E_j. Cette distance est notée $\Delta_{ij} \geq 0$ pour chaque $c_i \in O \cap (D_j \cup E_j)$.

Pour les cellules occupées dans l'ensemble central D_j, Δ_{ij} est la distance couverte le long du tentacule j entre le robot et le centre de la cellule occupée c_i.

Pour les cellules occupées dans E_j, seul un sous-ensemble $\overline{\boldsymbol{E_J}}$ est pris en compte : $\overline{\boldsymbol{E_J}} \subseteq O \cap E_j$. Ce sous-ensemble ne contient que des cellules qui réduisent l'espace libre du tentacule dans la direction normale à leur trajectoire (c'est-à-dire, qui rendent la trajectoire « étroite »). Pour chaque cellule de E_j occupée, on note Γ_{ij} le rayon à partir du centre du tentacule (du demi-cercle) et passant par c_i. La cellule c_i est ajoutée à $\overline{E_J}$ si, et seulement si, $D_j \cup E_j$, contient au moins une cellule occupée traversé par Γ_{ij} de l'autre côté du tentacule.

Exemple : dans la **Figure 4.1e**, $O \cap E3 = \{C_1, C_3, C_4\}$, alors que $\overline{E_3} = \{C_1, C_3\}$. La cellule C_4 n'a pas été considérée comme dangereuse, puisqu'elle est externe, et ne possède pas une contrepartie de l'autre côté du tentacule. Ensuite, pour les cellules dans $\overline{E_J}$, Δ_{ij} est la somme de deux termes : la distance du centre de c_i à sa projection normale sur le périmètre de la zone dangereuse centrale D_j, et la distance entre le robot et la projection normale le long du tentacule j. Le calcul de Δ_{ij} est illustré sur la Fig. 4.1, pour les quatre cellules occupées. Lorsque toutes les distances de risque pour le tentacule j sont calculées, on calcule Δ_j comme :

$$\Delta_j = \inf_{C_i \in (O \cap D_j) \cup \overline{E_J}} \Delta_{ij} \qquad (4.1)$$

Dans l'exemple de la figure 4.1, $\Delta_2 = \Delta_{12}$ et $\Delta_{33} = \Delta_3$. Nous utilisons ensuite Δ_j et deux seuils de distance Δ_d et Δ_s ($0 < \Delta_d < \Delta_s$), pour concevoir la fonction risque du tentacule :

$$H_j = \begin{cases} 0 & if\ \Delta_j \geq \Delta_s \\ \frac{1}{2}\left[1 + \tanh\left(\frac{1}{\Delta_j - \Delta_d} + \frac{1}{\Delta_j - \Delta_s}\right)\right] & if\ \Delta_d < \Delta_j < \Delta_s \\ 1 & if\ \Delta_j \leq \Delta_d \end{cases} \qquad (4.2)$$

Si $H_j = 0$, le tentacule est étiqueté comme *libre*. En pratique, le seuil Δ_s représente la distance limite pour laquelle le contexte cesse d'être sûr de sorte que le robot commence à quitter le chemin principal, qui représente l'asservissement visuel (suivi de la cible). D'autre part, Δ_d représente la distance pour laquelle le contexte devient dangereux, de sorte que le robot suive le meilleur tentacule pour faire le tour de l'obstacle.

Dans ce travail, les valeurs choisis sont : $\Delta_s = 6$ m et $D_d = 4{,}5$ m. La fonction de risque H_j correspondant à ces valeurs est représentée graphiquement à la figure 4.2

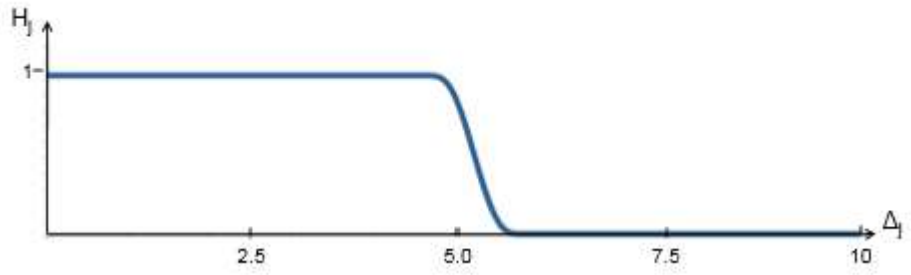


Fig.4.2 Evolution de la fonction risque

Cette fonction a l'avantage de permettre un passage progressif de l'état sûr à l'état dangereux.

Les H_j de tous les tentacules sont ensuite comparés, afin de déterminer H. Dans un premier temps, nous calculons la courbure $k \in IR$ que le robot suivrait s'il n'y avait pas d'obstacles. La courbure étant l'inverse du rayon R du demi-cercle du tentacule, sachant que V_z et ω sont les vitesses calculées par la loi de commande, nous avons :

$V_z = R.\omega$

$k = 1/R = \omega / V_z$

k est limité à l'intervalle de courbures possibles $[-K_M , K_M]$. Puis, les deux voisins de k sont déduits parmi toutes les courbures tentacule existants :

$K_n, K_{nn} \in K$ tel que $k \in [K_n , K_{nn}]$.

K_n est la courbure du tentacule qui se rapproche le mieux de la trajectoire sûre.

Une fonction de risque notée H_v est ensuite calculée par interpolation linéaire :

$$H_v = \frac{(H_{nn} - H_n)K + H_n K_{nn} - H_{nn} K_n}{K_{nn} - K_n} \qquad (4.3)$$

En pratique, H_v mesure le risque sur la voie considérée, en ne considérant que les obstacles sur les tentacules au voisinage de *k*. Dans le but de suivre la cible et de réaliser la tâche d'évitement, une condition suffisante est que les tentacules voisins soient libres (Hn = Hnn = 0). De cette façon, les obstacles sur les côtés ne dévieront pas le robot loin du chemin voulu.

La stratégie de détermination du meilleur rayon de courbure K_b est illustrée dans les exemples présentés dans la figure 4.3, où 5 tentacules sont utilisés. Dans la figure, les cellules dangereuses (à savoir, pour chaque tentacule j, les cellules appartenant à D_j U E_j) associées au tentacule j et au meilleur tentacule sont respectivement présentés en gris clair et gris foncé. Les cellules occupées sont en noir. Le meilleur tentacule est dérivé des fonctions de risque de tentacule définies précédemment. Si $H_v = 0$ (comme dans la figure 4.3 (a).), Le tentacule j peut être suivi (ici suivi de la cible) : il est mis $k_b = k_n$, et la commande est appliquée avec H=0. En revanche, si $H_v \neq 0$, un tentacule libre (H = 0) est recherché. Tout d'abord, pour éviter les changements brusques de contrôle, le tentacule est recherché entre le tentacule j et le meilleur tentacule à l'itération précédente, son rayon de courbure est noté K_{pb}, en gris dans la figure. S'il existe plusieurs tentacules qui sont considérés *libres*, le plus proche du tentacule j est choisi, comme sur la figure 4.3 (b). Si aucun des tentacules avec une courbure dans [K_n, K_{pb}] n'est libre, la recherche continue parmi les autres tentacules. Encore une fois, le meilleur tentacule sera celui qui est libre et est le plus proche de K_n et, en cas d'ambiguïté, le plus proche de K_{nn}. Si un tentacule clair est trouvé (comme dans la figure. 4.3 (c)), ce dernier est sélectionné et H est mis égale à 0. Dans le cas contraire, si aucun tentacule

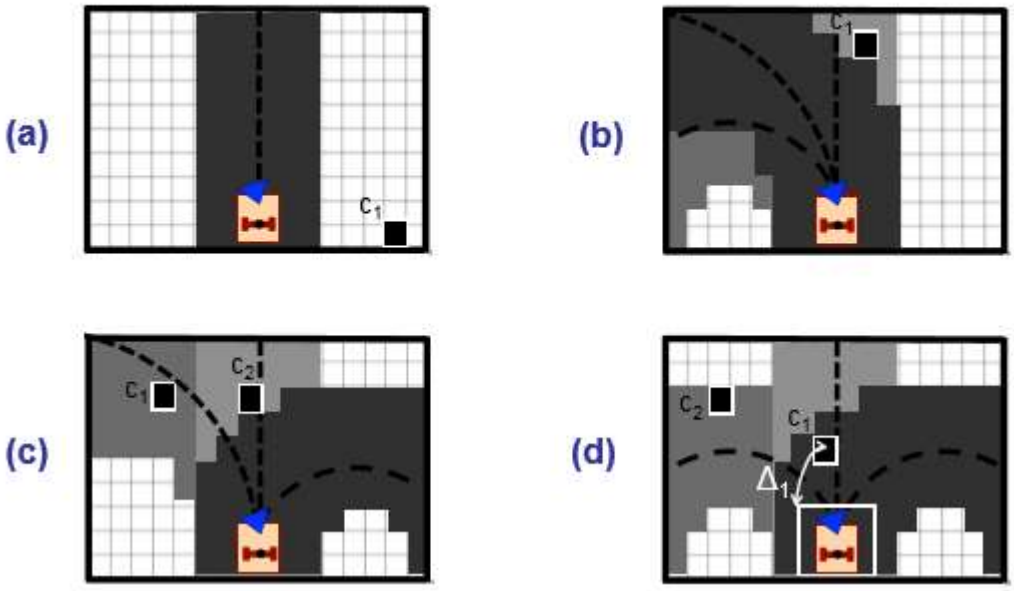


Fig.4.3 Stratégie de sélection du meilleur tentacule parmi 5 pour 4 scénarios différents

dans K n'est clair, celui dont le H_j calculé à l'aide de (17) est le minimum est choisi, et H est égal à ce H_j. Dans l'exemple de la Fig. 4.3 (d), le tentacule 1 est choisi et H est mis égale à H_1, puisque $\Delta_1 = \sup \{\Delta_1, ..., \Delta_5\}$, donc $H_1 = \inf \{H_1, ..., H_5\}$.

Les cellules associées au tentacule j, au meilleur tentacule précédent, et au meilleur tentacule sont présentés en nuances de gris (du plus foncé vers le plus clair) ; les tentacules correspondants sont tachetés, et les cellules c_1 et c_2 occupées sont représentées en noir. (a). Puisqu'il est libre, le tentacule j avec une courbure K_n est sélectionné : $K_b = K_n$. (b) Le tentacule libre avec un rayon de courbure dans $[K_n, K_{pb}]$ et qui est le plus proche de K_n est choisi. (c) Étant donné que tous les tentacules avec une courbure $[K_n, K_{pb}]$ sont occupés, le tentacule libre le plus proche du tentacule j est choisi. (d) Comme tous les tentacules sont occupés, le plus petit H_j est choisi, par conséquent, la plus grande distance de risque Δ_j (ici, Δ_1).

4.1.5 Le calcul des vitesses linéaires et angulaires :

L'objectif final du programme d'évitement est de calculer les vitesses linéaires et angulaires au robot, et ce, à partir d'une part de la loi de commande dont l'objectif est le suivi de la cible, et d'autre part, la prise en compte des obstacles obstruant le chemin.

Une fois la courbure idéale K_b trouvée, et sachant que la vitesse angulaire et linéaire sont liées par cette relation :

$K_b = \omega / V_{z,evit}$

Il suffit de trouver une des deux vitesses, et l'autre sera déduite systématiquement. Nous allons donc calculer $V_{z,evit}$, nous procédons comme suit :

Nous calculons $V_{z,evit,s}$ dans un contexte sûr, puis $V_{z,evit,u}$ dans un contexte dangereux (unsafe), et enfin, nous calculons $V_{z,evit}$ à l'aide des deux vitesses précédentes et de la fonction risque H.

Lorsque la vitesse angulaire est grande, l'asservissement visuel est moins efficace, et la vitesse de translation doit être réduite. Cela est généralement le cas lorsque le robot effectue de grands virages. Par conséquent, $V_{z,evit,s}$ est définie comme suit :

$$V_{z,evit,s}(\omega) = v_m + \frac{v_M - v_m}{2}\sigma \qquad (4.4)$$

Avec la fonction σ définie comme :

$$\sigma : IR * \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \rightarrow [0,2]$$

$$\omega \rightarrow 1 + \tanh(\pi - k_\omega |\omega|) \qquad (4.5)$$

$V_{z,evit,s}(\omega)$ décroît en fonction de ω de la valeur maximale v_M (quand $\omega = 0$) jusqu'à sa valeur minimale v_m (quand ω tend vers l'infini).

La décroissance de $V_{z,evit,s}$ peut être ajustée de façon empirique en utilisant le paramètre k_ω.

La vitesse de translation dangereuse $V_{z,evit,u}$ doit s'adapter au danger potentiel; elle peut être déduite à partir des obstacles sur la meilleure tentacule. En fait, $V_{z,evit,u}$ est déduite de la distance de collision δ_b, qui est une approximation prudente de la distance maximale que le robot peut parcourir le long de la meilleure tentacule sans entrer en collision. Puisque le rectangle le plus étroit contient le robot, si ce dernier suit le meilleur tentacule, les collisions ne peuvent se produire que dans les cellules occupées dans C_b. En effet, la collision avec la cellule c_i se produit à une distance notée $\delta_{ib} \geq 0$ que le robot doit couvrir le long du meilleur tentacule avant de toucher le centre de c_i

On définit alors δ_b comme étant le minimum des δ_{ib} :

$\delta_b = \inf(\delta_{ib}) / \ c_i \in (O \cap C_b)$

Si toutes les cellules dans C_b sont libres, $\delta_b = \infty$. Dans l'exemple de la figure 4.1, en supposant que le meilleur tentacule est celui de courbure nulle (b=2), on aurait $\delta_b = \delta_{12}$. Bien sûr, surestimer les dimensions du robot donne à des δ_b plus prudents.

La vitesse de translation dépendra donc de δ_b. Soient δ_d et δ_s deux paramètres seuils ajustables de sorte que $0 < \delta_d < \delta_s$. Si une possible collision est éloignée ($\delta_b \geq \delta_s$), la vitesse de translation peut être maintenue à celle calculée dans le contexte sûr. Par contre, si une dangereuse cellule occupée est proche ($\delta_b \leq \delta_d$), le robot doit s'arrêter. Pour satisfaire les conditions aux limites $V_{z,evit,u}$ (δ_d) =0 et $V_{z,evit,u}$ (δ_s) = $V_{z,evit,s}$, dans la situation intermédiaire, nous appliquons une décélération constante :

$$a = V_{z,evit,s}{}^2/2(\delta_d - \delta_s) < 0 \qquad (4.6)$$

Qu'on peut obtenir à partir de la distance de freinage requise :

$$\delta_b - \delta_d = - V_{z,evit,u}/2a \qquad (4.7)$$

$$V_{z,evit,u} = \begin{cases} V_{z,evit,s} & si\ \delta_s \leq \delta_b \\ V_{z,evit,s}\sqrt{\frac{\delta_b - \delta_d}{\delta_s - \delta_d}} & si\ \delta_d < \delta_b < \delta_s \\ 0 & sinon \end{cases} \qquad (4.8)$$

A présent, avec les valeurs de $V_{z,evit,u}$ et $V_{z,evit,s}$ nous pouvons calculer $V_{z,evit}$ de cette manière :

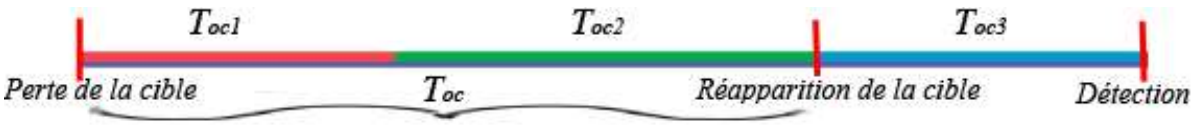
$$V_{z,evit} = (1 - H)V_{z,evit,s} + HV_{z,evit,u} \qquad (4.9)$$

4.2 Gestion de l'occultation - prédiction de la trajectoire

4.2.1 Généralités et hypothèses de travail

Un des objectifs de notre projet est de gérer les occultations, c'est-à-dire, la disparition de la cible du champ de vision. Nous nous plaçons ainsi dans un scénario où, durant la poursuite de la cible, cette dernière est cachée par un objet (ou une personne) durant un intervalle de temps T_{oc}. Durant T_{oc}, la Kinect donne la dernière position (avant occultation) de la cible durant le laps de temps T_{oc1}, puis n'envoie plus d'informations une fois ce temps dépassé, la cible apparaît dans le champ de vision après une durée T_{oc2}, enfin, le programme de perception repérera la cible après un intervalle de temps T_{oc3}, depuis sa réapparition. Le robot aura alors pour référence une mauvaise position de la cible qu'il poursuivra pendant T_{oc1}, puis s'arrêtera dès que la Kinect n'émet plus d'information pendant T_{oc2}+T_{oc3} pour finalement redémarrer et poursuivre la cible (s'il arrive à retrouver cette dernière).

$$T = T_{oc1} + T_{oc2} + T_{oc3}$$



Cette situation présente deux inconvénients majeurs :

- Le robot arrête de poursuivre la cible durant toute la période T, ce qui n'est pas permis par le cahier de charge.
- Si le robot s'arrête durant l'occultation, la cible réapparaîtra à un endroit éloigné de la position actuelle du robot et risque de ne pas être repérée.

Notre objectif est que le robot continue de poursuivre la cible même si cette dernière est occultée, et ce, jusqu'à ce que la Kinect détecte à nouveau la cible. Pour ce faire, nous effectuons une prédiction de la trajectoire de la cible réelle durant l'occultation. La précision de cette prédiction dépendra de deux paramètres essentiels :

- la durée de l'occultation. En effet, plus la durée d'occultation de la cible est importante, plus les erreurs de prédictions s'accumulent. Ceci conduit le robot à dévier de la trajectoire réelle de la cible et donc il aura moins de chances de la retrouver dans le voisinage de la position prédite quand celle-ci apparaîtra, ce qui a pour conséquence la perte définitive de la cible.

- la prédictibilité de la trajectoire. En outre, plus la trajectoire de la cible est aléatoire (accélération soudaine durant l'occultation, demi-tour…) plus faible sera la chance de retrouver la cible une fois celle-ci réapparue. Le cas idéal serait qu'elle garde la même direction et vitesse qu'avant l'occultation.

Hypothèses de travail :

Nous supposons les hypothèses suivantes :

- L'occultation est de courte durée (moins de 10 secondes) ;
- Durant le temps d'occultation, la cible ne change pas soudainement de direction ou de vitesse.

4.2.2 Prédiction de la trajectoire

Soit $(\boldsymbol{R_k})$ le repère lié à la Kinect à l'instant t = kΔt et soit $(\boldsymbol{R_a})$ le repère lié à la Kinect à t=0, appelé repère absolu.

Ces deux repères sont bidimensionnels, sur un plan parallèle au sol, il n'y a donc pas de composantes pour exprimer la hauteur bien que la Kinect puisse le faire.
Ce choix est justifié par le fait que le robot ne peut pas faire de mouvement vertical, et donc la composante "$\boldsymbol{y_k}$" (qui est la composante verticale pour la Kinect) est superflue.

Soit ${}^{k}\boldsymbol{P_{k,cible}}$ et ${}^{a}\boldsymbol{P_{k,cible}}$ les positions de la cible par rapport au repère $(\boldsymbol{R_k})$ exprimées respectivement dans les repères $(\boldsymbol{R_k})$ et $(\boldsymbol{R_a})$.

Soit ${}^{a}\boldsymbol{P_{a,cible}}$ et ${}^{a}\boldsymbol{P_{a,kinect}}$ les positions respectives de la cible et de la Kinect, par rapport à $(\boldsymbol{R_a})$ et exprimées dans $(\boldsymbol{R_a})$.

Nous noterons aussi ${}^{a}\boldsymbol{P_{a,kinect}}$ par $\boldsymbol{P_{kinect}} = (\boldsymbol{x_k}, \boldsymbol{z_k})$.
Soit $\boldsymbol{\theta}$ l'orientation de la Kinect (ou du robot) par rapport à $(\boldsymbol{R_a})$, qui représente aussi l'orientation de $(\boldsymbol{R_k})$ par rapport à $(\boldsymbol{R_a})$.

${}^{k}\boldsymbol{P_{k,cible}}$ est le vecteur position donné par la Kinect, la loi de commande utilise ce vecteur pour calculer les vitesses du robot. Ainsi, le rôle de la gestion de l'occultation est de fournir un

vecteur ${}^{k}\boldsymbol{P_{k,cible}}$ à la loi de commande quand la Kinect ne donne plus d'information.

Notre stratégie est de prédire à chaque itération la position de la cible en se basant sur la vitesse précédente calculée de celle-ci et en utilisant un algorithme qui gère les occultation.

Nous préférerons la vitesse de la cible dans le repère absolu ${}^{a}\boldsymbol{V_{a,cible}} = ({}^{a}\boldsymbol{V_{x\ cible}}, {}^{a}\boldsymbol{V_{y\ cible}})$ car celle exprimée dans le repère de la Kinect ${}^{k}\boldsymbol{V_{k,cible}}$ n'est pas la vitesse réelle de la cible, cette vitesse dépend aussi de celle du robot. Ainsi, si le robot se dirige vers une cible immobile, ${}^{k}\boldsymbol{V_{k,cible}}$ prédite serait non nulle, et nous prédirions une cible qui se dirige constamment vers le robot, ce qui le poussera à reculer constamment.

Nous utilisons donc pour la prédiction :

$$ {}^{a}\boldsymbol{V_{a,cible}}(\boldsymbol{k\Delta t}) = ({}^{a}\boldsymbol{P_{a,cible}}(\boldsymbol{k\Delta t}) - {}^{a}\boldsymbol{P_{a,cible}}((\boldsymbol{k-1})\boldsymbol{\Delta t}))/\boldsymbol{\Delta t} \quad \textbf{(4.18)} $$

En notant $(\boldsymbol{k\Delta t})$ par $[\boldsymbol{k}]$; on obtient

$$ {}^{a}\boldsymbol{V_{a,cible}}[\boldsymbol{k}] = ({}^{a}\boldsymbol{P_{a,cibl}}\ [\boldsymbol{k}] - {}^{a}\boldsymbol{P_{a,cible}}[\boldsymbol{k-1}])/\boldsymbol{\Delta t} \qquad (4.10) $$

On pose

$$ \boldsymbol{H} = \begin{pmatrix} {}^{a}\boldsymbol{x_{a,cible}} \\ {}^{a}\boldsymbol{y_{a,cible}} \\ {}^{a}\boldsymbol{V_{x\ cible}} \\ {}^{a}\boldsymbol{V_{y\ cible}} \end{pmatrix} \qquad (4.11) $$

$$ \boldsymbol{A} = \begin{pmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{\Delta t} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{\Delta t} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{pmatrix} \qquad (4.12) $$

L'étape de prédiction dans le repère absolu est alors :

$$ \boldsymbol{H_p}[\boldsymbol{k+1}] = \boldsymbol{A}.\boldsymbol{H}[\boldsymbol{k}] \qquad (4.13) $$

Où $\boldsymbol{H_p}[\boldsymbol{k+1}]$ représente le vecteur prédit à l'étape k+1 à partir du vecteur H donné par la Kinect à l'étape k.

$\boldsymbol{H[k]}$ se renouvelle tant que la Kinect donne encore des informations viables. Dès que la cible est perdue $\boldsymbol{H_p}$ est calculé à partir du vecteur prédit à l'étape précédente :

$$\boldsymbol{H_p[k+1] = A.H_p[k]} \tag{4.14}$$

Encore est-il que *H* est donné dans $(\boldsymbol{R_a})$, alors que la Kinect ne donne les positions que dans le repère $(\boldsymbol{R_k})$.

Il faut donc trouver les coordonnées de ${}^{\boldsymbol{a}}\boldsymbol{P_{a,cible}}$ à partir de ${}^{\boldsymbol{k}}\boldsymbol{P_{k,cible}}$ pour effectuer la prédiction, puis, passer de la position prédite ${}^{\boldsymbol{a}}\boldsymbol{P_{a,cible}}$ à ${}^{\boldsymbol{k}}\boldsymbol{P_{k,cible}}$ utilisée par la loi de commande.

$${}^{\boldsymbol{a}}\boldsymbol{P_{a,cible}} = {}^{\boldsymbol{a}}\boldsymbol{P_{a,kinect}} + {}^{\boldsymbol{a}}\boldsymbol{P_{k,cible}} \tag{4.15}$$

$${}^{\boldsymbol{a}}\boldsymbol{P_{a,cible}} = \boldsymbol{P_{kinect}} + \boldsymbol{R(\theta)}.{}^{\boldsymbol{k}}\boldsymbol{P_{k,cible}} \tag{4.16}$$

Où $\boldsymbol{R(\theta)}$ représente la matrice de rotation du repère $(\boldsymbol{R_k})$ par rapport au repère $(\boldsymbol{R_a})$:

$$\boldsymbol{R(\theta)} = \begin{pmatrix} \boldsymbol{\cos\theta} & \boldsymbol{\sin\theta} \\ \boldsymbol{-\sin\theta} & \boldsymbol{\cos\theta} \end{pmatrix} \tag{4.17}$$

Enfin, pour obtenir à nouveau ${}^{\boldsymbol{k}}\boldsymbol{P_{k,cible}}$, nous utilisons la relation :

$${}^{\boldsymbol{k}}\boldsymbol{P_{k,cible}} = \boldsymbol{R^{-1}(\theta)}.({}^{\boldsymbol{a}}\boldsymbol{P_{a,cible}} - \boldsymbol{P_{kinect}}) \tag{4.18}$$

$\boldsymbol{R}$ étant une matrice orthogonale, nous avons : $\boldsymbol{R^{-1}} = \boldsymbol{R^T}$ et donc

$$\boldsymbol{R^{-1}(\theta)} = \begin{pmatrix} \boldsymbol{\cos\theta} & \boldsymbol{-\sin\theta} \\ \boldsymbol{\sin\theta} & \boldsymbol{\cos\theta} \end{pmatrix} \tag{4.19}$$

➢ Conclusion

Nous avons détaillé dans ce chapitre les algorithmes qui permettent la gestion de l'occultation ainsi que l'évitement d'obstacles.

L'algorithme de gestion de l'occultation permet une prédiction de la trajectoire de la cible en adoptant comme hypothèse qu'elle se déplace avec une vitesse constante.

L'algorithme de l'évitement d'obstacles permet de prendre en compte les obstacles qui se trouvent dans l'environnement voisin du robot et de les dévier tout en gardant l'objectif principal qui est l'asservissement visuel.

Dans le chapitre suivant, nous allons présenter une étude des lois de commandes synthétisées avec leurs trois variantes et implémentées la plateforme.

Chapitre 5

Loi de commande et résultats expérimentaux.

➢ Introduction

Dans ce chapitre, nous passons en détail les différentes lois de commande déjà implémentées sur la plateforme robotique avant notre contribution, suivie d'une présentation de notre loi de commande avec ses variantes.

Enfin les résultats expérimentaux et leurs interprétations, englobant les tests des lois de commande ainsi que ceux de la gestion de l'occultation pour différents scénarios.

5.1 Loi de commande en asservissement visuel :

Dans le cadre de notre travail, il nous a été demandé de concevoir et d'implémenter une loi de commande qui va assurer le suivi de personnes et qui va remédier aux insuffisances de la loi déjà existante.

Un bref exposé des lois de commande existantes, suivi des lois proposées feront l'objet de cette partie.

5.1.1Asservissement visuel 2D :

Une fois que les coordonnées des points désirés sont récupérées, l'objectif est d'appliquer la commande référencée vision à notre robot mobile de manière à atteindre un but défini à priori dans l'image. Le principe de l'asservissement visuel 2D consiste à contrôler le mouvement de la caméra de manière à faire converger les indices visuels courants s(q,t) vers les indices visuels désirés s* . Pour cela, le formalisme des fonctions de tâche est utilisé :

$$\boldsymbol{e} = \boldsymbol{C}(\boldsymbol{s} - \boldsymbol{s}^{*}) \qquad (5.1)$$

Où C représente une matrice de combinaison permettant de prendre en compte un nombre d'indices visuels supérieurs au nombre de degré de liberté du robot. Dans notre cas, le nombre de degrés de liberté est limité à 3, par conséquent C est pris de dimension 4 relative à 4 jonctions : la tête, le torse, l'épaule gauche et l'épaule droite.

5.1.2 Asservissement 3D point :

Le formalisme du problème pour la première solution, nous conduit à imposer une fonction de tache, incorporant les informations 3D, de la forme suivante :

$$\boldsymbol{e(r,t) = C * (P - P^*)} \quad (5.2)$$

Où

r est le vecteur configuration du robot,

C est une matrice prenant en compte un nombre de fonctionnalité en plus du degré de liberté du robot ;

P est un point de la cible, c'est la valeur courante de l'information fournit par les capteurs ;

$\boldsymbol{P^*}$ Représente la pause de référence cible à atteindre dans le cadre du capteur.

Contrairement à la commande 2D qui agit au niveau de l'image, l'asservissement 3D considère la cible comme un modèle tridimensionnel situé dans le repère Kinect et donc impliquant sa position et son orientation.

Pour ces deux loi de commande, l'insuffisance peut être provoqué par la posture de la cible, puisque pour ces cas la convergence se traduit par la convergence d'un losange courant vers un losange désiré (c'est-à-dire des quatre points des joints : tête, épaule gauche, épaule droite et torse). Lorsque la personne se mettra de profil le losange devient distordu, ce qui provoque un disfonctionnement de la loi de commande

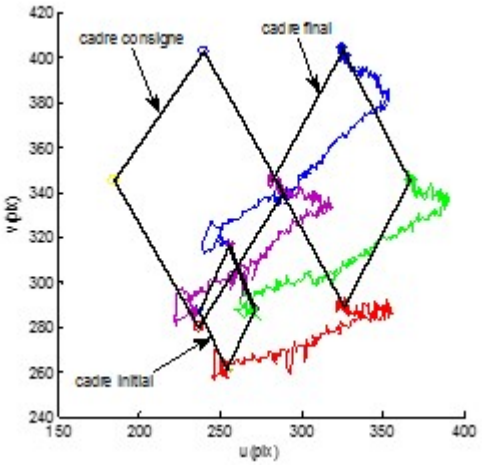


Fig.5.1 Fonction de tache et Indices visuels dans l'image pour le Cas1

5.1.3 Asservissement 3D par retour d'état dans l'espace capteur :

Cette fois-ci, la tache robotique à réaliser consiste à ramener le repère Kinect courant dans une situation désirée dite repère Kinect désiré. Cette tache peut être effectuée en deux étapes. La première étape consiste à définir le repère Kinect désiré à travers la pose désirée de la cible par rapport au robot (le mouvement de la cible étant libre on fait déplacer la Kinect pour atteindre cette pose). En deuxième étape, on calcule une commande qui fait converger le repère Kinect courant vers le repère Kinect de référence [Tam, 13].

Concernant cet asservissement, l'insuffisance se manifeste lors de la rotation de la cible, puisque la loi est synthétisée pour de faibles rotations. De plus, comme mentionnée, la loi consiste à faire converger un repère courant vers un repère consigne et puisque le robot se doit de garder la même vision de la cible, donc même si cette dernière ne fait que se tourner à gauche ou à droite, le robot va se déplacer sur un cercle pour se mettre dans la situation initiale.

Pour l'asservissement 3D en général, la loi de commande est synthétisée dans le but d'amener la position de la cible relative à la Kinect sur une position désirée. Pour ce faire, deux solutions ont été proposées. Pour la première solution, 4 points de la cible sont sélectionnés lesquels sont exprimés en 3D. Ainsi la mise en œuvre de la commande doit assurer la convergence des cordonnées courantes de ces 4 points vers les cordonnées désirées de ces mêmes points. Pour la deuxième solution, une loi de commande qui amène le repère lié à la Kinect sur le repère désiré de la Kinect correspondant à la situation désirée de la cible est synthétisée.

5.2 Conception d'une loi de commande visuelle pour le suivi de personnes :

L'objectif de cette partie est de concevoir une loi de commande permettant au robot de suivre une cible humaine, quel que soit le type de sa trajectoire et quel que soit son orientation.

5.2.1 Inconvénients des commandes implémentées et proposition de solutions :

Les lois de commande donnant les meilleurs résultats au début de notre travail sont l'asservissement 3D par retour d'état, et l'asservissement 3D point, ces deux méthodes présentent actuellement des inconvénients :

i. Elle ne fonctionnent que si la cible ne change pas d'orientation au-dessus d'un certain seuil ($\Delta\theta < 15°$). Ceci est dû à une approximation du sinus de l'angle par son DL.

ii. Même un fonctionnement parfait de cette loi de commande ne conviendrait pas à l'objectif de suivi du projet. En effet, cette loi a pour référence une position du robot qui fait constamment face à la cible, on peut comparer dans les schémas suivants le résultat désiré, et le résultat donné :

Le problème des lois de commandes déjà existantes réside dans le fait qu'elles prennent en compte l'orientation de la cible, or, celle-ci se trouve être nocive au vu des objectifs outre la complexité qu'elle ajoute au problème.

Faces à ces difficultés nous proposons les solutions suivantes :

Notre solution est donc que le robot suive la cible indépendamment de son orientation. L'orientation du robot est prise en compte mais pas celle de la cible. Ainsi, si nous prenons l'exemple de la commande 3D point, qui utilise 4 points de 3 coordonnées chacune du squelette de la cible afin de prendre en compte la position et l'orientation de cette dernière, ce qui nous fait une référence à 12 coordonnées. Nous simplifions le problème à 1 unique point du squelette qui suffira à donner la position du robot, qui plus est, étant donné que le robot ne peut faire de mouvement selon la verticale, nous ne prendrons les 2 coordonnées du plan horizontal (parallèle au sol) de ce point. Nous passons ainsi d'une référence de 12 coordonnées à une référence de 2 coordonnées, ce qui simplifie énormément le problème et permet une loi de commande plus performante.

Cette approche permet de résoudre les deux problèmes majeurs liés aux lois de commande précédentes :

i. le problème du changement d'orientation ne se pose plus vu que l'orientation de la cible n'est plus prise en compte, ainsi, la cible peut tourner brusquement, ou faire demi-tour sans que la loi de commande ne soit affectée.
ii. L'objectif de commande est atteint, le robot ne tente pas de se mettre automatiquement en face de la cible et suit des trajectoires plus simples.

5.2.2 Formulation générale des lois de commande visuelle proposées :

Notre loi de commande aura comme entrée la position (x,z) de la cible sur le plan horizontal, cette position est donnée par le programme de Gestion de l'occultation mentionné

précédemment.
La sortie de la loi de commande dépend de la structure mécanique du robot.

Le robot ne peut faire que deux mouvements :

- Avancer tout droit (selon l'axe z perpendiculaire à l'axe des roues).
- Tourner autour de l'axe vertical (axe y).

Nous devons donc fournir comme sorties la vitesse linéaire Vz et la vitesse angulaire ω du robot.

On pose C le vecteur de commande, q la configuration de la cible obtenue à partir de la position donnée par la Kinect, q_d la configuration désirée de la cible, ${}^k\dot{q}_{a,cible}$ la vitesse de la cible par rapport au repère absolu exprimée dans le repère de la Kinect :

$$C = \begin{pmatrix} V_z \\ \omega \end{pmatrix}$$

$$q = \begin{pmatrix} {}^kZ_{k,cible} \\ \theta \end{pmatrix}$$

Où θ désigne l'angle entre la cible et l'axe z par rapport au robot.

$$q_d = \begin{pmatrix} Z_d \\ \theta_d \end{pmatrix}$$

$$\dot{q} = \begin{pmatrix} {}^kV_{z,cible} \\ \dot{\theta} \end{pmatrix}$$

$$\dot{q} = {}^k\dot{q}_{a,cible} - C \tag{5.3}$$

$$C = {}^k\dot{q}_{a,cible} - \dot{q} \tag{5.4}$$

Nous avons élaboré et testé les trois lois de commandes suivantes :

5.2.2.1 Commande linéaire saturée

Nous avons en premier temps opté pour une commande linéaire de la forme :

$$C = K(q - q_d) \tag{5.5}$$

$$K = \begin{pmatrix} K_1 & 0 \\ 0 & K_2 \end{pmatrix}$$

$$\begin{cases} V_z = K_1({}^kZ_{k,cible} - Z_d) \\ \omega = K_2(\theta - \theta_d)\ ;\ \theta_d = 0 \end{cases} \tag{5.6}$$

De plus, les vitesses linéaires et angulaires du robot sont limitées par les contraintes mécaniques du robot, ainsi, nous avons saturé la commande pour prendre en compte ces limitations :

$$-0{,}6 < V_z < 0{,}6 \ ; \left(\frac{m}{s}\right)$$

$$-70 < \omega < 70 \ ; \ (\frac{\text{deg}}{\text{s}})$$

En remplaçant dans (5.3), on obtient l'équation de la vitesse de la cible :

$$\dot{q} = {}^{k}\dot{q}_{a,cible} - K(q - q_d) \qquad (5.7)$$

La vitesse de la cible ${}^{k}\dot{q}_{a,cible}$ agit alors comme une perturbation, ainsi, on peut schématiser les équations précédentes par un schéma bloc :

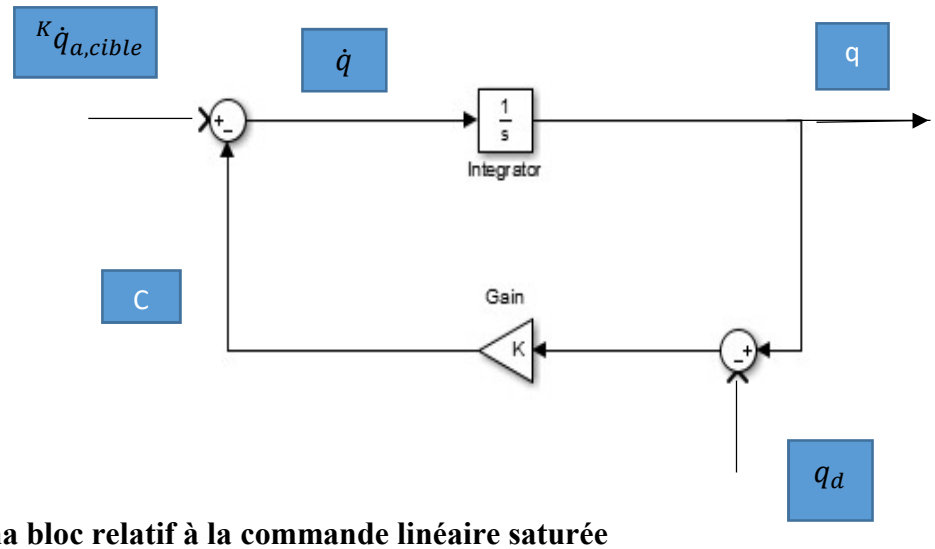


Fig 5.2 schéma bloc relatif à la commande linéaire saturée

a) **Cas de la cible à l'arrêt**

${}^{k}\dot{q}_{a,cible} = 0$, et nous obtenons les équations du mouvement suivantes :

$$\dot{q} = -K(q - q_d) \qquad (5.8)$$

$$\begin{cases} {}^{k}V_{z,cible} = \frac{d({}^{k}Z_{k,cible})}{dt} = -K_1({}^{k}Z_{k,cible} - Z_d) \\ \dot{\theta} = -K_2\theta \end{cases} \qquad (5.9)$$

Ce qui permet une convergence exponentielle vers la position et l'angle désirés :

$$\begin{cases} {}^{k}Z_{k,cible} = A_1 e^{-K_1 t} + Z_d \\ \theta = A_2 e^{-K_2 t} \end{cases} \quad (5.10)$$

b) Cas de la cible ayant des vitesses initiales (V_0 et ω_0) non nulles :

$$\begin{cases} {}^{k}\dot{Z}_{k,cible} = V_0 - K_1({}^{k}Z_{k,cible} - Z_d) \\ \dot{\theta} = \omega_0 - K_2\theta \end{cases} \quad (5.11)$$

Le robot dans ce cas aura un décalage proportionnel à la vitesse de la cible et inversement proportionnel aux gains K_i :

$$\begin{cases} {}^{k}Z_{k,cible} = A_1 e^{-K_1 t} + Z_d + \frac{V_0}{K_1} \\ \theta = A_2 e^{-K_2 t} + \frac{\omega_0}{K_2} \end{cases} \quad (5.12)$$

Ainsi, plus la vitesse de la cible est faible ou plus K_i est élevé meilleure est la poursuite. Mais les K_i imposent aussi la vitesse de convergence de la position et de l'angle vers la configuration désirée, plus on augmente K_i plus les mouvements du robot deviennent brusques, ce qui a des effets néfastes sur la mécanique du robot.

Conclusion : Cette loi de commande est très efficace quand la cible est à l'arrêt ou marche à faible vitesse, mais n'atteint pas une poursuite pour des cibles bougeant rapidement.

5.2.2.2 Commande dynamique saturée :

Nous pouvons imposer une convergence exponentielle de q vers q_d, quelle que soit la vitesse de la cible en imposant la dynamique suivante :

$$\dot{q} = -K(q - q_d) \quad (5.13)$$

En remplaçant dans l'équation (5.4), nous obtenons :

$$C = {}^{k}\dot{q}_{a,cible} + K(q - q_d) \quad (5.14)$$

Cette loi de commande a pour paramètre la vitesse de la cible qui, à priori, est inconnue.

Nous pouvons résumer ces équations par le schéma bloc suivant :

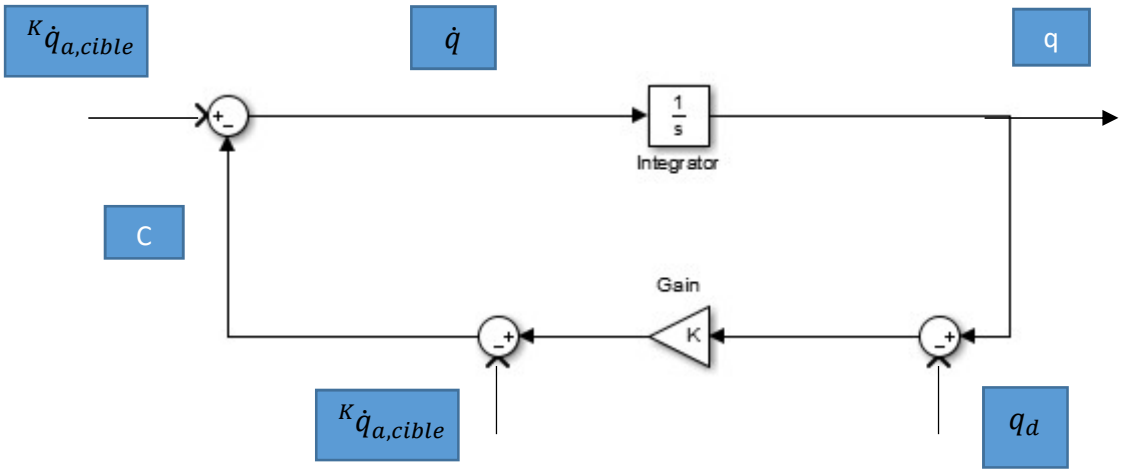


Fig 5.3 schéma bloc relatif à la commande dynamique saturée

En analysant l'information à instant précis, il n'est pas possible de déterminer la vitesse angulaire ω_{cible} et linéaire $V_{z,cible}$ de la cible. Mais il est possible d'extraire cette information sur une succession de données en ayant les informations suivantes :

- La position de la cible par rapport à la Kinect à l'instant [k] et [k+1].
- Les vitesses linéaires et angulaires du robot à l'instant [k].

En utilisant les mêmes notations que précédemment :

Soit $^kZ_{a,cible} = {}^k\overrightarrow{OM}.\vec{k}$, où O est l'origine du repère absolue, M la position de la cible dans le repère absolu, et $\vec{k}$ le vecteur unitaire parallèle à l'axe (OZ) du repère absolu.
Soit $^kZ_{a,rob} = {}^k\overrightarrow{OR}.\vec{k}$, où R est la position du robot par rapport au repère absolu.

Soit Δt la période d'échantillonnage.

Et nous rappelons que Vz et ω sont, respectivement, les vitesses linéaire et angulaire du robot.
On pose :

$$\Delta Z_{rob} = {}^kZ_{a,rob}[\mathrm{k}+1] - {}^kZ_{a,rob}[\mathrm{k}] \qquad (5.15)$$

$$\Delta Z_{cible} = {}^kZ_{a,cible}[\mathrm{k}+1] - {}^kZ_{a,cible}[\mathrm{k}] \qquad (5.16)$$

Nous avons alors les équations suivantes :

$$\Delta Z_{rob} = V_z \cos(\omega \Delta t)\, \Delta t \quad (5.17)$$

$$\Delta Z_{cible} = \mathrm{V}_{z,cible} \Delta t \quad (5.18)$$

$$\Delta^k Z_{k,cible} = \Delta Z_{cible} - \Delta Z_{rob} = {}^k Z_{k,cible}[k+1] - {}^k Z_{k,cible}[k] \quad (5.19)$$

En combinant les équations (5.17), (5.18) et (5.19), nous obtenons :

$$\mathrm{V}_{z,cible} \Delta t - V_z \cos(\omega \Delta t) \Delta t = {}^k Z_{k,cible}[k+1] - {}^k Z_{k,cible}[k]$$

$$\mathrm{V}_{z,cible}[k] = \frac{{}^k Z_{k,cible}[k+1] - {}^k Z_{k,cible}[k]}{\Delta t} + V_z \cos(\omega \Delta t) \quad (5.20)$$

Il est à noter que $\mathrm{V}_{z,cible}$ à l'instant [k] dépend de données à l'instant [k+1], ceci est dû au fait que la position à l'instant [k+1] dépend de la vitesse à l'instant [k], donc en ayant pour donnée la position à l'instant [k+1] nous pouvons déduire qu'elle a été la vitesse à l'instant [k].

Nous avons donc obtenu la vitesse linéaire de la cible $\mathrm{V}_{z,cible}$, nous utilisons la même méthode pour trouver ω_{cible} :

$$\omega = \omega_{cible} + K\, \theta[k+1]$$

$$\Delta\theta_{rob} = \omega[\mathrm{k}] \Delta t$$

$$\Delta\theta_{cible} = \omega_{cible}[\mathrm{k}] \Delta t$$

$$\Delta\theta_{cible} - \Delta\theta_{rob} = \Delta\theta = \theta[k+1] - \theta[k]$$

$$\omega_{cible}[\mathrm{k}] \Delta t - \omega[\mathrm{k}] \Delta t = \theta[k+1] - \theta[k]$$

$$\omega_{cible}[\mathrm{k}] = \frac{\theta[k+1] - \theta[k]}{\Delta t} + \omega[\mathrm{k}] \quad (5.21)$$

Ainsi nous avons obtenu la vitesse angulaire de la cible, nous pouvons remplacer les deux vitesses obtenues dans l'équation (5.14) de la commande :

$$C[k+1] = \begin{pmatrix} \mathrm{V}_{z,cible}[k] \\ \omega_{cible}[k] \end{pmatrix} + K(q[k+1] - q_d[k+1]) \quad (5.22)$$

$$\begin{pmatrix} V_z[k+1] \\ \omega[\mathrm{k}+1] \end{pmatrix} = \begin{pmatrix} \frac{{}^k Z_{k,cible}[k+1] - {}^k Z_{k,cible}[k]}{\Delta t} + \mathrm{Vz}.\cos(\omega \Delta t) \\ \frac{\theta[k+1] - \theta[k]}{\Delta t} + \omega[\mathrm{k}] \end{pmatrix} + K(q[k+1] - q_d[k+1])$$

(5.23)

5.2.2.3 Commande dynamique non linéaire :

Nous avons enfin conçu une loi de commande non linéaire se basant sur le principe suivant :

Nous voulons que le robot soit à une distance désirée l_d et à un angle désiré $\theta_d = 0$ de la cible. Nous avons donc proposé la loi de commande suivante :

$$\begin{cases} V_z = \mathrm{V}_{z,cible} + K_1({}^k l_{k,cible} - l_d) \\ \omega = \omega_{cible} + K_2(\theta - \theta_d)\ ;\ \theta_d = 0 \end{cases} \quad (5.24)$$

${}^k l_{k,cible}$ est la distance de la cible par rapport à la Kinect dans le repère de la Kinect

${}^k l_{k,cible}$ est calculée de la manière suivante (voir Fig 5.7) :

$${}^k Z_{k,cible} = {}^k l_{k,cible}.\cos(\theta) \Rightarrow {}^k l_{k,cible} = \frac{{}^k Z_{k,cible}}{\cos(\theta)}$$

En remplaçant dans le système d'équations (5.24) nous obtenons :

$$\begin{cases} V_z = \mathrm{V}_{z,cible} + K_1(\frac{{}^k Z_{k,cible}}{\cos(\theta)} - l_d) \\ \omega = \omega_{cible} + K_2\theta \end{cases} \quad (5.25)$$

La dynamique obtenue est :

$$\begin{cases} {}^k\dot{Z}_{k,cible_z} = -K_1(\frac{{}^k Z_{k,cible}}{\cos(\theta)} - l_d) \\ \dot{\theta} = -K_2\theta \end{cases} \quad (5.26)$$

La résolution du système est ardue mais il est possible de faire une analyse qualitative de la solution :

Nous remarquons d'abord que le seul point critique du système d'équations est :

$$\left({}^k Z_{k,cible}, \theta\right) = (l_d, 0)$$

Qui est effectivement l'objectif de commande désiré.

En utilisant la toolbox pplane, et la commande pplane8 sur matlab, nous avons effectué un échantillonnage en prenant différentes conditions initiales dans le plan $\left({}^k Z_{k,cible}, \theta\right)$ pour le système d'équations différentielles non linéaires :

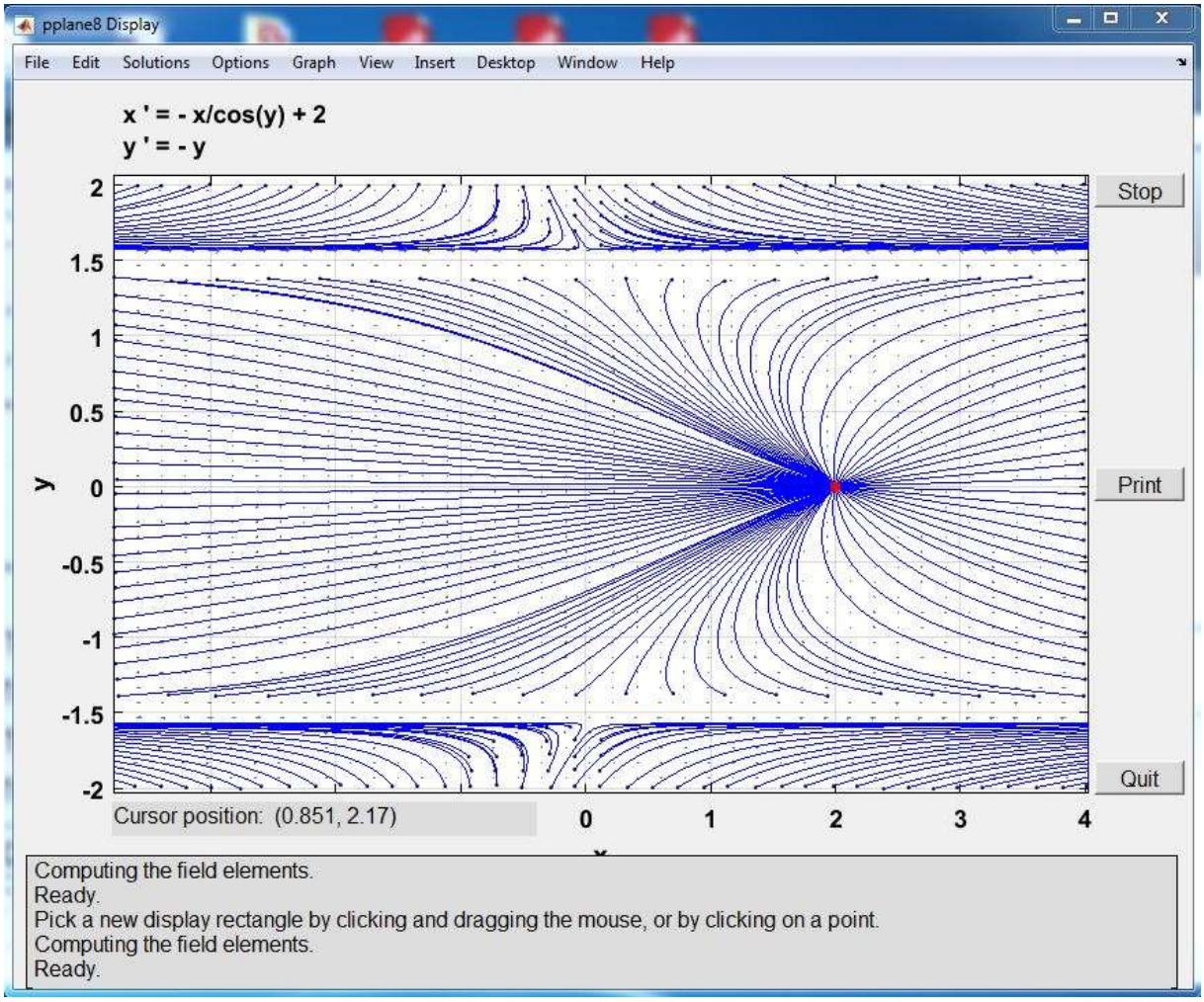


Fig.5.4 Diagramme représentatif des solutions du système (5.26) en prenant en compte les paramètres utilisés avec le robot B21r.

On remarque que dans l'intervalle $-\pi/2 < \theta < \pi/2$, et quel que soit ${}^{k}Z_{k,cible}$ dans $\mathbb{R}$, pour toute condition initiale dans ce domaine la solution du système converge vers l'unique point d'équilibre $(2,0) = (l_d, 0)$ (nous avons pris comme distance désirée celle utilisée durant les tests sur le robot).

Toute condition initiale en dehors du domaine précédemment mentionné donne lieu à une solution divergente.

En effectuant la transformation $\left({}^{k}Z_{k,cible}, \theta\right) \mapsto \left({}^{k}X_{k,cible}, {}^{k}Z_{k,cible}\right)$, le domaine :

$D_1 = \left\{\left({}^{k}Z_{k,cible}, \theta\right) \in \mathbb{R}^2 / \ {}^{k}Z_{k,cible} \in \mathbb{R} \text{ et } -\pi/2 < \theta < \pi/2 \right\}$ se transforme en

$D_2 = \left\{\left({}^{k}X_{k,cible}, {}^{k}Z_{k,cible}\right) \in \mathbb{R}^2 / \ {}^{k}Z_{k,cible} \neq 0 \right\}$

On remarque que cette loi de commande ne peut diverger que si la position de la cible a pour coordonnée précisément la valeur ${}^{k}Z_{k,cible} = 0$.

En pratique, cette contrainte ne pose pas de problème sur la commande, et ce, pour plusieurs raisons :

- Une position qui a pour coordonnée ${}^{k}Z_{k,cible} = 0$ est forcément en dehors du champ de vision de la Kinect.
- Même si la cible sort du champ de vision de la Kinect, la gestion de l'occultation permet d'estimer sa position, si la position prédite à un instant [k] vaut exactement ${}^{k}Z_{k,cible} = 0$ (ce qui est peu probable en soit, étant donné que la précision des résultats donne 9 chiffres après la virgule), la cible quitterait cette position dès l'itération suivante.
- Les commandes sont saturées pour éviter des dommages sur le robot.

En somme, la loi de commande dynamique non linéaire impose une dynamique stable pour toute condition initiale dans le domaine D_2, ce dernier étant égal à $\mathbb{R}^2$ *sauf la droite Z=0.*

- ✓ **Etude de la stabilité du système par la 2ème méthode Lyapunov :**

Nous entamons à présent une étude théorique de la stabilité du système décrit par les équations différentielles (5.26).
Nous utilisons la 2ème méthode de Lyapunov qui consiste à trouver une fonction de Lyapunov définie positive adaptée au système d'équations et étudier le caractère négatif ou semi-négatif de sa dérivée temporelle.

On rappelle donc le système dont on veut faire l'étude de Lyapunov :

$$\begin{cases} {}^{k}\dot{Z}_{k,cible_z} = -K_1\left(\frac{{}^{k}Z_{k,cible}}{\cos(\theta)} - l_d\right) \\ \dot{\theta} = -K_2\theta \end{cases} \quad (5.26)$$

Pour pouvoir appliquer la méthode de Lyapunov, nous devons avoir pour point d'équilibre (0,0) (ainsi, quand la fonction est définie positive, elle ne s'annulera que pour le point d'équilibre).
Nous faisons donc le changement de variable suivant :

$$Z = {}^{k}Z_{k,cible} - l_d$$

Le système d'équations devient alors :

$$\begin{cases} \dot{Z} = -K_1\left(\frac{Z+l_d}{\cos(\theta)} - l_d\right) \\ \dot{\theta} = -K_2\theta \end{cases} \quad (5.27)$$

On choisit V la fonction de Lyapunov suivante :

$$V(Z,\theta) = \frac{Z^2}{2} + \frac{\theta^2}{2}$$

V est une fonction définie positive vérifiant le premier critère de Lyapunov

Calculons à présent sa dérivée temporelle :

$$\dot{V}(Z,\theta) = -K_1\left(\frac{Z+l_d}{\cos(\theta)} - l_d\right)Z - K_2\theta^2 \quad (5.28)$$

$$\dot{V}(Z,\theta) = -\frac{K_1}{\cos(\theta)}\left(Z^2 + l_d Z(1-\cos(\theta))\right) - K_2\theta^2 \quad (5.29)$$

$$\dot{V}(Z,\theta) = -\frac{K_1}{\cos(\theta)}\left(Z^2 + 2\frac{l_d}{2}Z(1-\cos(\theta)) + \frac{l_d^2}{4}(1-\cos(\theta))^2\right) + \frac{K_1 l_d^2(1-\cos(\theta))^2}{4\cos(\theta)} - K_2\theta^2$$

$$\dot{V}(Z,\theta) = -\frac{K_1}{\cos(\theta)}\left(Z + \frac{l_d}{2}(1-\cos(\theta))\right)^2 + \frac{K_1 l_d^2(1-cos(\theta))^2}{4cos(\theta)} - K_2\theta^2$$

$$\dot{V}(Z,\theta) = -\frac{K_1}{\cos(\theta)}\left(Z + \frac{l_d}{2}(1-\cos(\theta))\right)^2 - \frac{K_1 l_d^2}{4}\left(\frac{4K_2}{K_1 l_d^2}\theta^2 - \frac{(1-cos(\theta))^2}{cos(\theta)}\right) \quad (5.30)$$

Nous remarquons alors que $\dot{V} < 0$ quand :

$$\frac{4K_2}{K_1 l_d^2}\theta^2 - \frac{(1-cos(\theta))^2}{cos(\theta)} > 0 \quad (5.31)$$

Cette inéquation dépend uniquement du facteur positif $\frac{4K_2}{K_1 l_d^2}$ et de θ, c'est une inéquation non linéaire que nous pouvons résoudre numériquement et illustrer graphiquement dans la figure suivante :

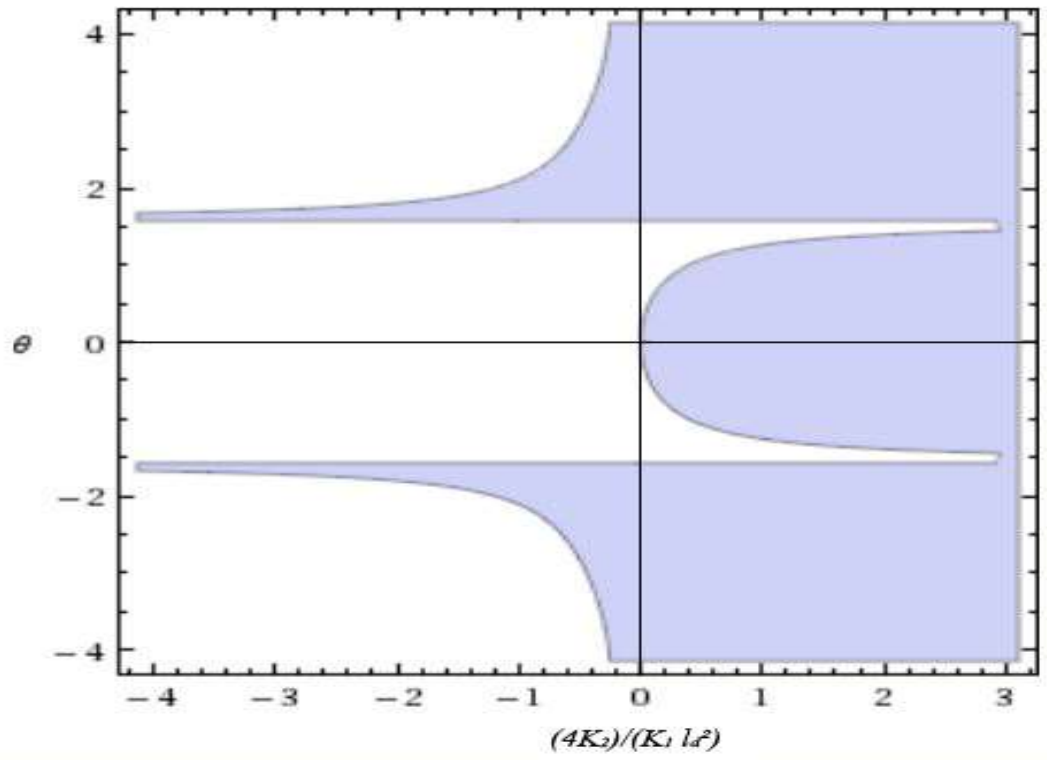


Fig5.5 : Représentation graphique des solutions de l'inéquation (5.31) en bleu.

Le domaine pour lequel $\dot{V} < 0$ est en bleu sur la figure, sachant que $\frac{4K_2}{K_1 l_d^2} > 0$ et que $-\pi/2 < \theta < \pi/2$.

Etant donné que V n'est pas tout le temps négative, il est intéressant d'étudier l'inéquation pour les valeurs de K_1, K_2 et l_d utilisées pour notre commande :

$K_1 = K_2 = 0{,}6$, $l_d = 2$, *donc* $\frac{4K_2}{K_1 l_d^2} = 1$;

Nous pouvons alors calculer numériquement et montrer graphiquement pour quelles valeurs de θ nous obtenons $\dot{V} < 0$:

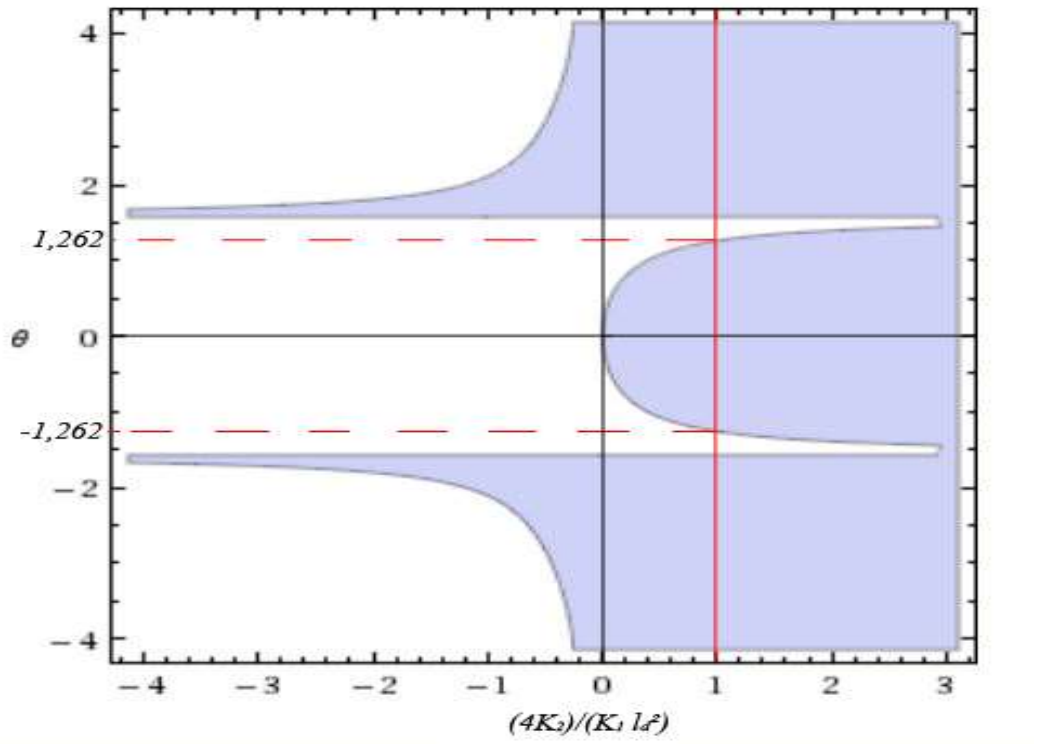


Fig5.6 Représentation graphique des solutions de l'inéquation (5.31) pour les paramètres utilisés dans le robot B21r.

Le domaine pour lequel nous obtenons une convergence asymptotique de la solution est :

$D_3 = \{(Z, \theta) \in \mathbb{R}^2 / \, Z \in \mathbb{R} \text{ et } -1{,}262 < \theta < 1{,}262\}$

$-1{,}262 < \theta < 1{,}262 \;\leftrightarrow\; -72{,}29° < \theta < 72{,}29°$

Bien sûr, le fait d'avoir prouvé la convergence avec la méthode de Lyapunov sur ce domaine ne veut pas dire que le système ne converge pas en dehors de ce domaine. D'ailleurs, en résolvant le système numériquement on a observé qu'il convergeait pour tout le domaine D_2.

Nous n'avons pas trouvé de fonction de Lyapunov donnant de meilleurs résultats.

5.3 Résultats expérimentaux :

Dans cette partie nous présentons les résultats recueillis pour les différentes lois de commande et méthode d'occultation citées dans ce rapport en explicitant le protocole expérimental et les outils utilisés pour recueillir les données de l'expérience.

Cette partie est divisée en deux sections, la première sera consacrée aux différentes lois de commandes utilisées durant notre projet, quant à la seconde, elle contiendra les résultats des tests concernant la gestion de l'occultation.

5.3.1 Résultats expérimentaux des lois de commande :

5.3.1.1 Commande linéaire saturée :

- ✓ **Protocol expérimental :**

Durant cette expérience nous testons la loi de commande utilisée pour le suivi. Une cible se tient à une distance quelconque du robot, dans notre cas $\left({}^{k}X_{k,cible}(0), {}^{k}Z_{k,cible}(0)\right) = (0{,}36;\ 4{,}25)$.

La loi de commande est de la forme suivante :

$$C = K(q - q_d)$$

Où nous rappelons que $q_d = \begin{pmatrix} Z_d \\ \theta_d \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$

Ainsi nous lançons le programme de commande à l'instant t=0, et nous recueillons les données $\left({}^{k}X_{k,cible}(t)\ , {}^{k}Z_{k,cible}(t)\ \right)$ durant une période de 10 secondes (suffisante pour la convergence de la réponse) :

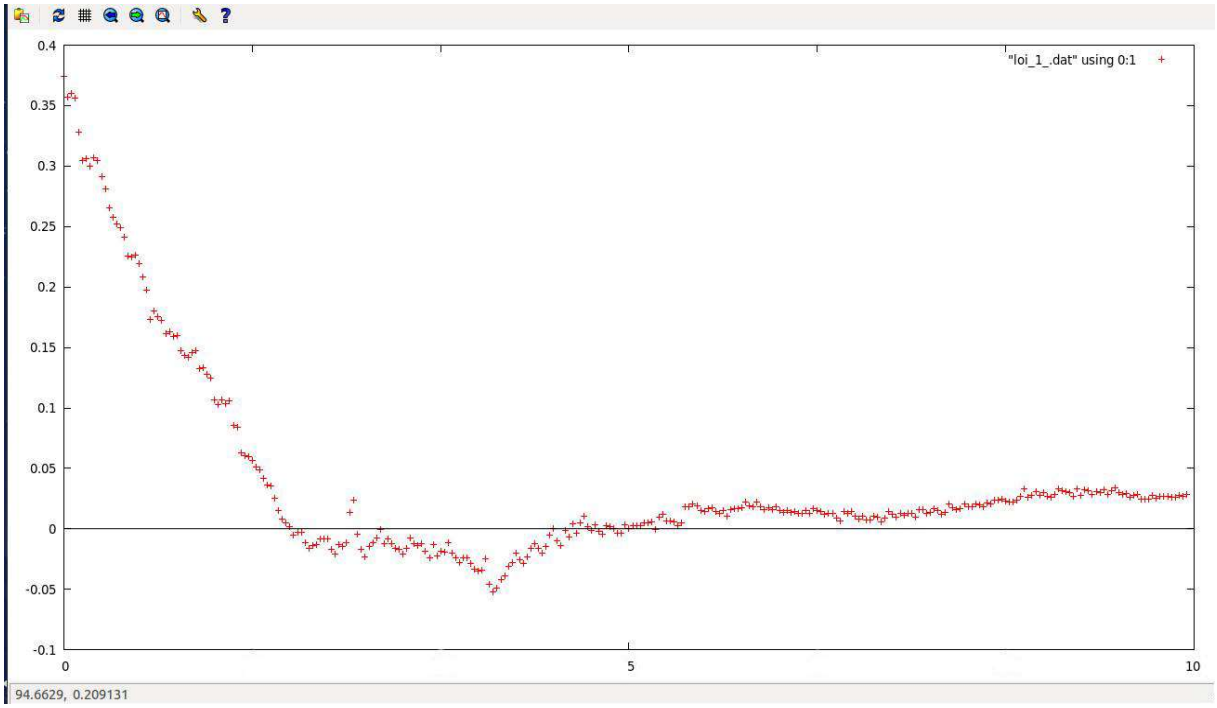


Fig 5.7 Sortie ${}^{k}X_{k,cible}(t)$ relative à la commande linéaire saturée

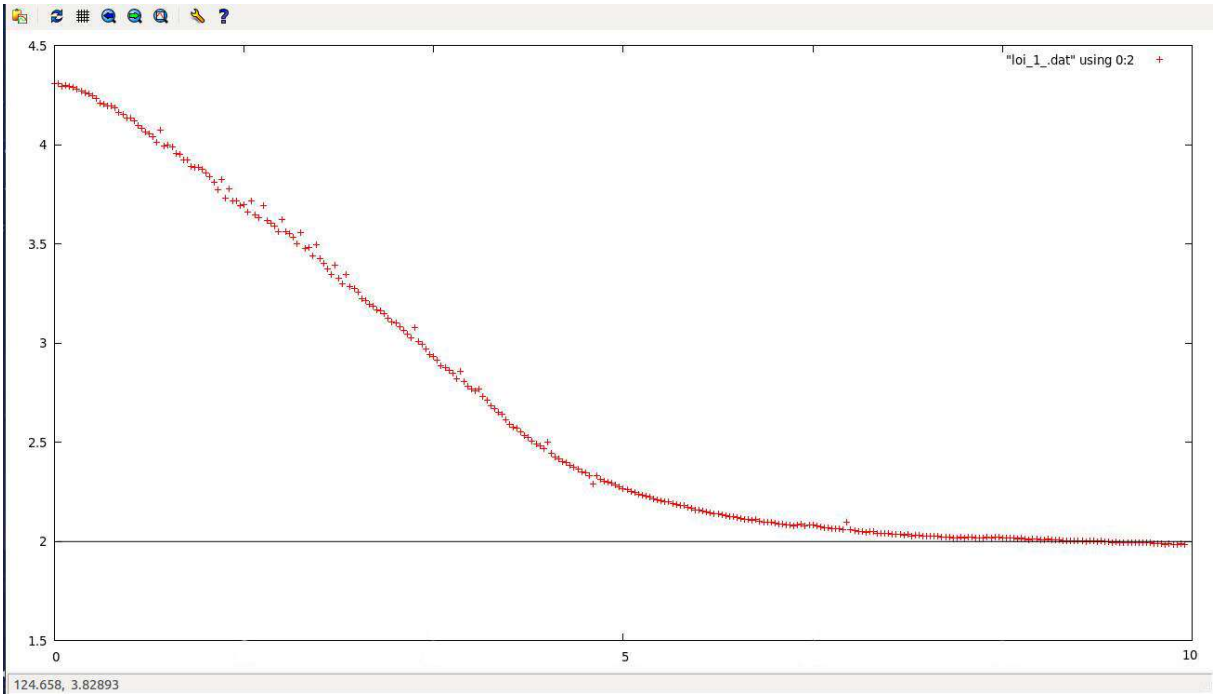


Fig 5.8 Sortie ${}^{k}Z_{k,cible}(t)$ relative à la commande linéaire saturée

Nous remarquons que ${}^{k}X_{k,cible}$ converge vers $\theta_d = 0$ avec une erreur statique de l'ordre de 0,025 (2,5 cm) et un temps de réponse à 10% de la valeur initiale valant 4 secondes.
Nous remarquons aussi que ${}^{k}Z_{k,cible}$ converge effectivement vers la valeur désirée $Z_d = 2$ avec une erreur statique indiscernable et un temps de réponse à 10% de la valeur finale égal à 6 secondes.
Il est à noter que la vitesse de convergence dépend fortement de la valeur de saturation de la vitesse linéaire V_z que nous avons limité, pour ce test, à 0,5m/s.

Conclusion : La loi de commande linéaire saturée fonctionne en pratique, sa performance en précision est satisfaisante au vu de l'application. Quant à sa vitesse, elle dépend principalement des contraintes imposées par la mécanique du robot.

5.3.1.2 Commande non linéaire saturée :

- ✓ **Protocol expérimental :**

Dans ce test, nous réalisons la même expérience que précédemment en utilisant la loi de commande non linéaire :

$$\begin{cases} V_z = K_1\left(\dfrac{{}^{k}Z_{k,cible}}{\cos(\theta)} - l_d\right) \\ \omega = K_2(\theta - \theta_d) \end{cases}$$

Nous recueillons les données expérimentales $\left({}^{k}X_{k,cible}(t)\ ,\ {}^{k}Z_{k,cible}(t)\right)$ pendant une période de 20 secondes :

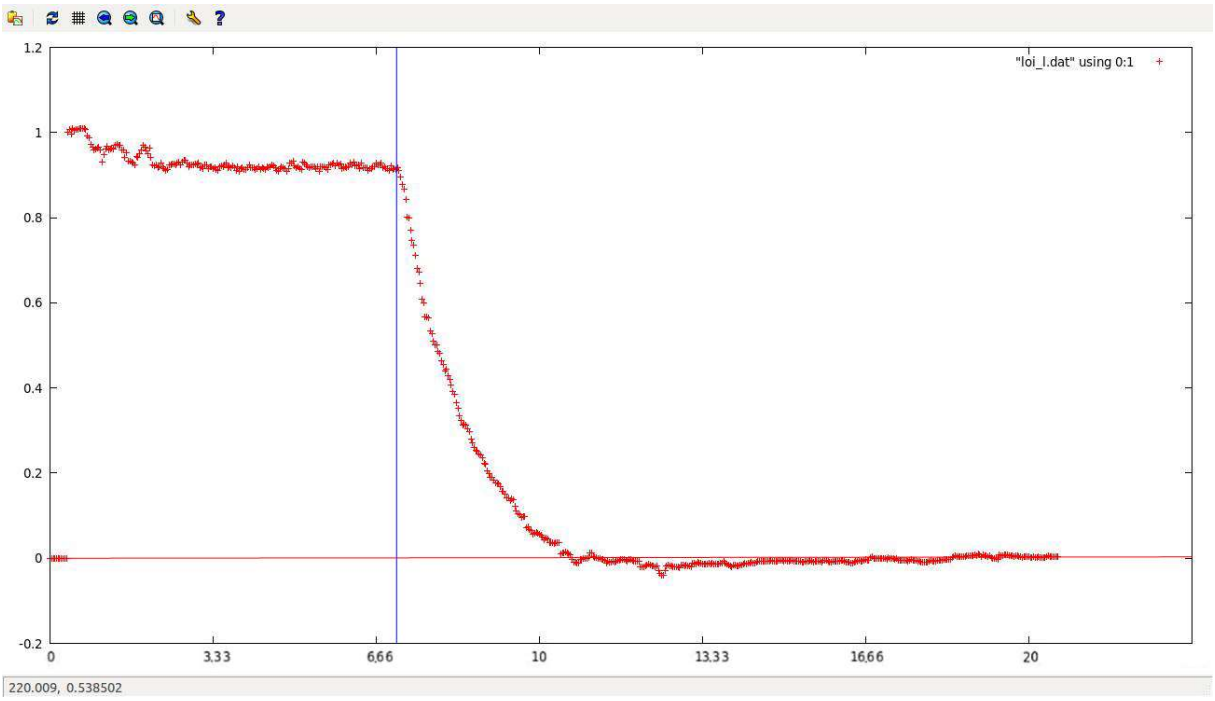


Fig 5.9 Sortie ${}^{k}X_{k,cible}(t)$ relative à la commande non linéaire saturée

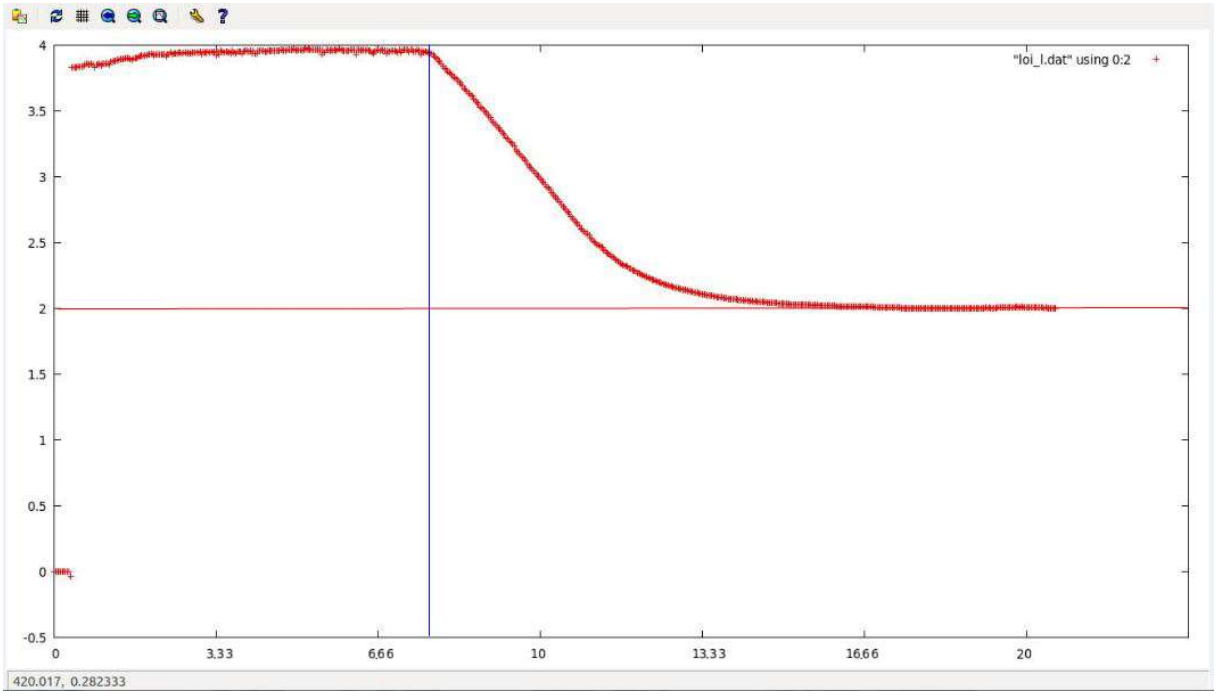


Fig 5.10 Sortie ${}^{k}Z_{k,cible}(t)$ relative à la commande non linéaire saturée

Les temps de convergence à 10% sont similaires à la première loi étudiée.

Il semble que nous obtenons une convergence plus précise pour ${}^{k}X_{k,cible}$ par rapport à la première loi de commande malgré une condition initiale plus éloignée de la valeur désirée.

Conclusion : La loi de commande non linéaire saturée fonctionne en pratique. Elle donne des résultats similaires à la première loi avec une meilleure précision pour la variable ${}^{k}X_{k,cible}$.

5.3.2 Résultats expérimentaux de la gestion de l'occultation :

Nous avons réalisé deux tests différents pour valider la gestion de l'occultation, dans le premier test, la cible sort complètement de la scène, quant au deuxième, la cible est occultée (cachée) temporairement par une personne qui passe.

5.3.2.1 Cible sortie de la scène :

- ✓ **Protocol expérimental :**

Durant cette expérience, une cible est bien visible dans le champ de vision de la Kinect, puis sort du cadre de la scène sans y revenir, le programme de gestion de l'occultation prédit alors la trajectoire qu'aura cette cible selon les informations recueillies aux instants qui précèdent l'occultation.

Nous recueillons les positions $\left({}^{k}X_{k,cible}(t)\ ,\ {}^{k}Z_{k,cible}(t)\right)$ données par la Kinect et ceux données par le programme de prédiction :

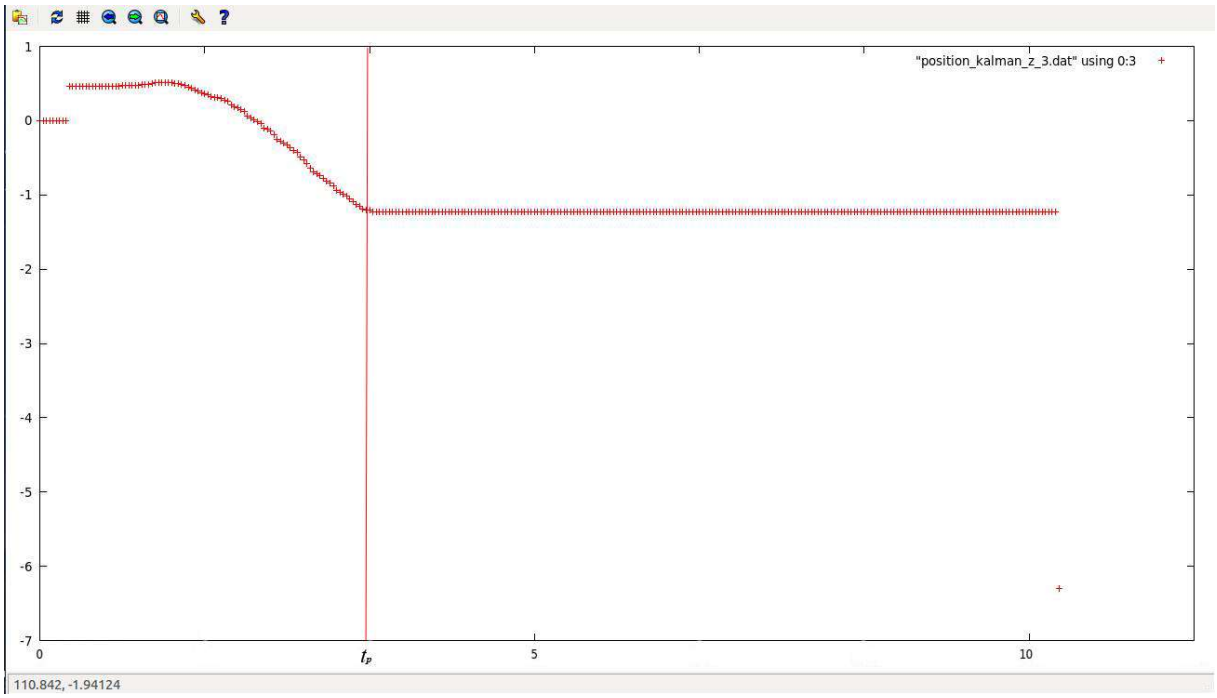


Fig 5.11 Sortie ${}^{k}X_{k,cible}(t)$ donnée par la Kinect

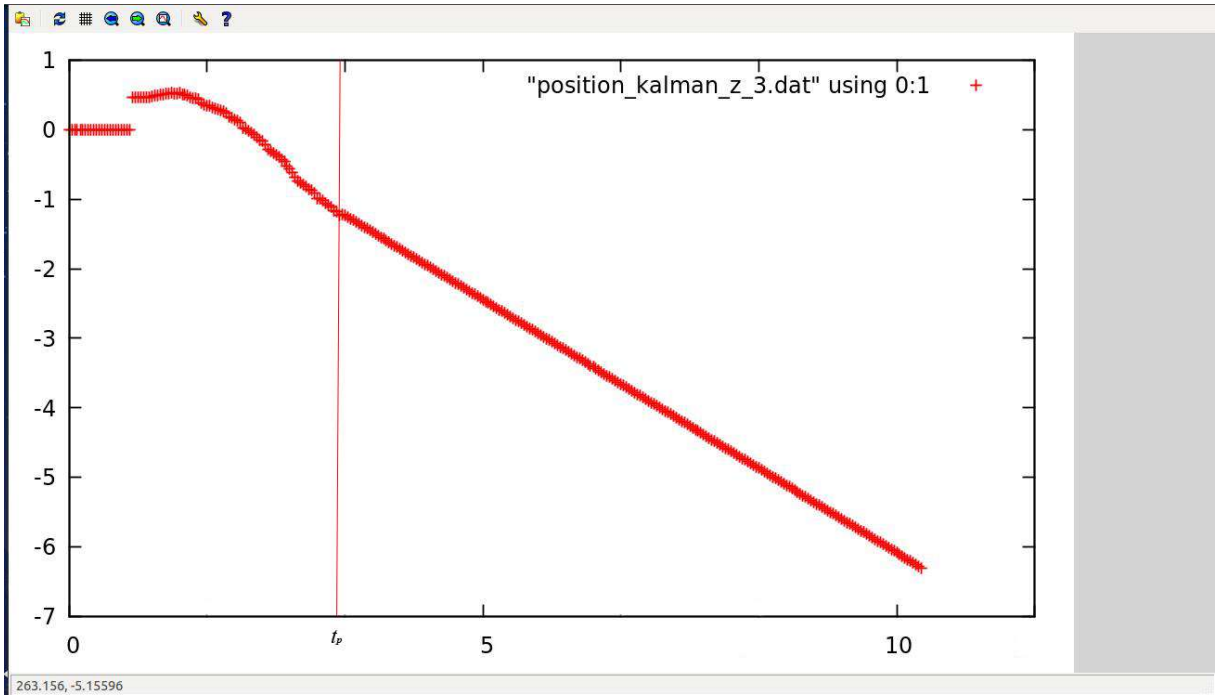


Fig 5.12 Sortie ${}^{k}X_{k,cible}(t)$ donnée par le programme de la gestion de l'occultation

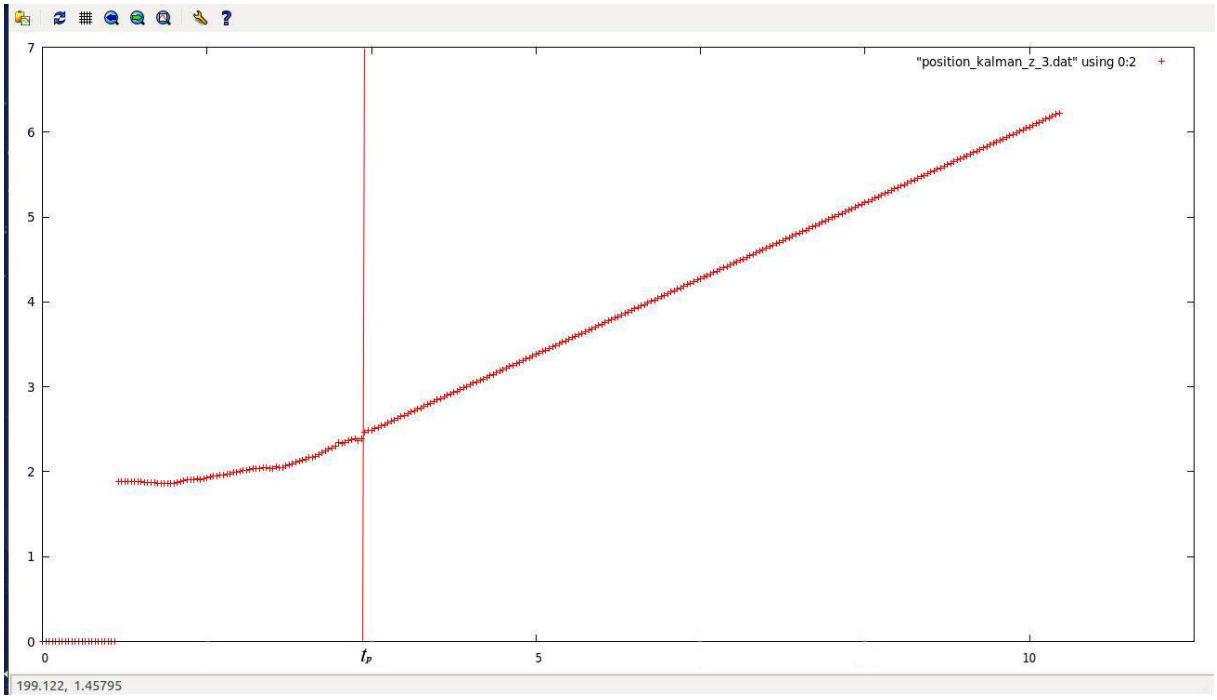


Fig 5.13 Sortie ${}^{k}Z_{k,cible}(t)$ donnée par le programme de la gestion de l'occultation

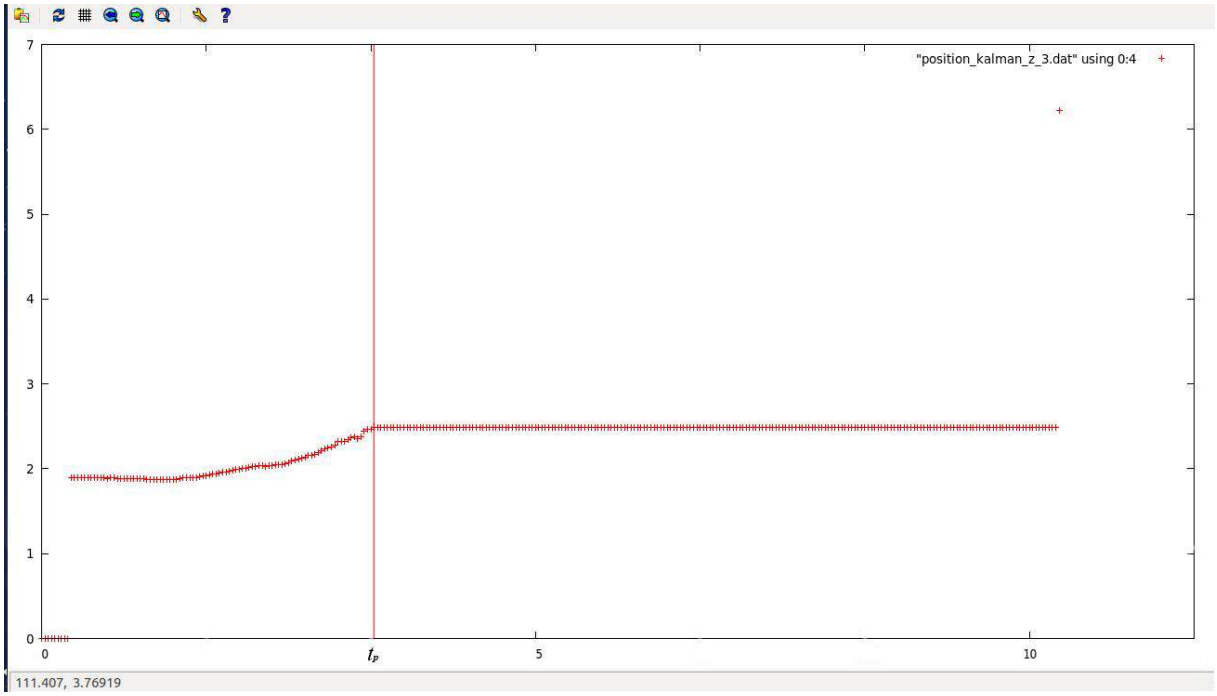


Fig 5.14 Sortie $^{k}Z_{k,cible}(t)$ donnée par la Kinect

Nous remarquons que pour les positions données par la Kinect, dès que la cible sort du champ de vision les positions deviennent constantes. Ce problème est dû à un retard de la Kinect à annoncer la perte de la cible, généralement, le programme de suivi (OpenNi Tracker) qui donne les positions de la cible prend plus de 10 secondes pour afficher le message « Lost user » qui correspond à la perte de la cible, entretemps, les positions données sont constantes et égales à la dernière position observée.

Nous exploitons cette particularité d'OpenNi Tracker dans notre programme de sorte que dès que les observations commencent à être exactement les mêmes (ce qui est impossible même quand la cible est immobile à cause des bruits) nous déduisons que la cible a été perdue et nous entamons la prédiction de la trajectoire.

Nous remarquons que le programme de gestion de l'occultation prédit de façon cohérente les trajectoires des cibles en gardant la même pente que celle avant la perte de la cible, et ce, malgré les bruits.

5.3.2.2 Occultation temporaire de la cible :

- ✓ **Protocol expérimental :**

Durant cette expérience, la cible se déplace dans le champ de vision de la Kinect, et une personne passe entre la Kinect et la cible, cachant temporairement la cible.
Nous recueillons les positions données par la Kinect, et celles données par le programme d'occultation pour une durée de 10 secondes :

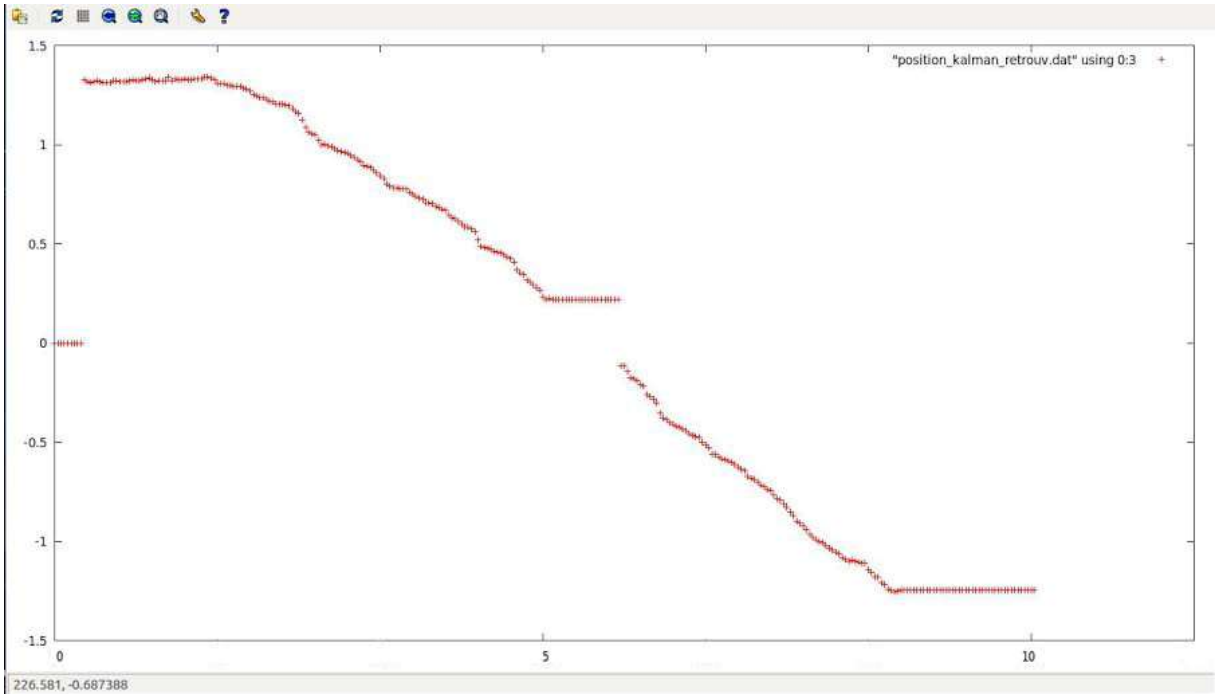


Fig 5.15 Sortie ${}^{k}X_{k,cible}(t)$ donnée par la Kinect

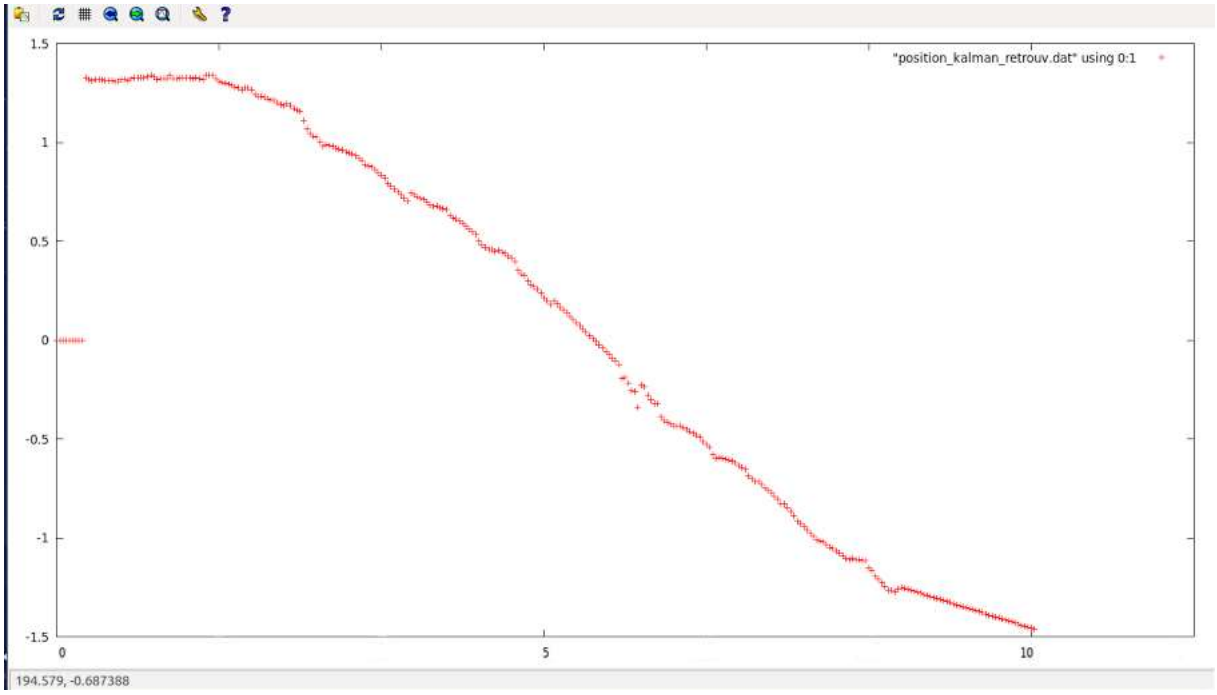


Fig 5.16 Sortie ${}^{k}X_{k,cible}(t)$ donnée par le programme de gestion de l'occultation

On observe le même phénomène décrit auparavant, les positions données par la Kinect deviennent constantes dès que la cible est perdue.
En comparant les images dans l'intervalle de temps où la cible est perdue, on remarque que malgré les perturbations dans les données de la Kinect, la trajectoire prédite est cohérente, et la cible est retrouvée au voisinage de la position prédite à la fin de l'occultation.

➢ Conclusion :

Au vu des tests réalisés dans différents contextes, la prédiction de la trajectoire à réussit à réaliser une gestion de l'occultation en répondant aux problèmes citées au Chapitre 4 :
- Le robot continue de suivre la position prédite de la cible au lieu de suivre les positions constantes données par la Kinect.
- Le robot ne s'arrête pas quand la Kinect n'émet plus de données.
- Le robot retrouve la cible au voisinage de la position prédite et peut continuer la poursuite de celle-ci une fois retrouvée.

Conclusion générale

Conclusion générale

Dans le cadre de ce mémoire, nous avons traité le problème de la navigation d'un robot guide basée sur un capteur visuel. Pour ce faire, nous avons tout d'abord présenté le système robotique destiné à notre application : le robot B21R équipé d'un capteur de vision (Kinect) ainsi qu'un télémètre laser.

Nous nous sommes penchés sur trois problématiques essentielles qui sont :

- L'amélioration de l'asservissement du robot à travers la conception et l'implémentation de lois de commande qui ont assuré un asservissement visuel plus efficace et plus souple comparé aux lois déjà implémentées sur la plateforme.
- Un des objectifs de notre projet est de gérer les occultations, c'est-à-dire, la disparition de la cible du champ de vision, la prédiction de la trajectoire à réussit à réaliser une gestion de l'occultation en répondant aux problèmes cité dans le chapitre 4 :

  - Le robot continue de suivre la position prédite de la cible au lieu de suivre les positions constantes données par la Kinect.
  - Le robot ne s'arrête pas quand la Kinect n'émet plus de données.
  - Le robot retrouve la cible au voisinage de la position prédite et peut continuer la poursuite de celle-ci une fois retrouvée.

- Dans le cadre de notre projet, l'objectif de l'évitement d'obstacles est que le robot puisse suivre la cible tout en évitant les différents obstacles qui peuvent gêner son mouvement ou l'endommager.

Le programme global implémenté a permis d'atteindre les objectifs suivants :

1- Résoudre le problème d'occultation en implémentant un programme de gestion de l'occultation pour prédire la trajectoire de la personne suivie (**H1**) et ainsi de la retrouver si celle-ci est occultée par un autre objet.
2- Résoudre les problèmes concernant la poursuite dus à un changement brutal d'angle, et ceci, en changeant complètement de stratégie de commande
3- Adapter l'évitement d'obstacles déjà implémenté au contexte de poursuite de la cible (**H1**).

Conclusion générale

D’après cette courte expérience, et à travers les résultats des tests effectués, il nous semble intéressant de continuer ce travail en traitant par la suite les thèmes suivants :

- Palier au problème de reconnaissance de la cible, car dans le cas présent les programmes implémentés ne permettent pas de reconnaitre la personne suivie.
- Equipé la PTU pour que le robot puisse bénéficier d’un degré de liberté additionnel et qui va permettre d’améliorer et de mieux répondre au problème de gestion de l’occultation.

Bibliographie

Bibliographie

[Eje 12] : L EJEUNE, S. PIÉRARD, M. V ANd ROOGENBROECK et J. V ERLY. Utilisation de la Kinect. Linux Magazine France, N°151, pages 16-29, Juillet-Août, 2012.

[Fre 10] : B. Freedman, A. Shpunt, M. Machline, and Y. Arieli. Depth mapping using projected patterns, 2010. US Patent Application 20100118123.

[Gar 05] : N. Garcia-Aracil, E. Malis, R. Aracil-Santonja, and C. Perez-Vidal, "Continuous visual servoing despite the changes of visibility in image features,"Robotics,IEEE Transaction son,vol.21,no.6,pp.1214–1220, 2005.

[Ben 03] : S. Benhimane and E. Malis, "Vision-based control with respect to planar and non-planar objects using a zooming camera," in IEEE International Conference on Advanced Robotics, 2003.

[Che 02] : G. Chesi, K. Hashimoto, D. Prattichizzo, and A. Vicino, "Keeping features in the camera's field of view: a visual servoing strategy," in 15th Int. Symp. on Mathematical Theory of Networks and Systems, August 2002.

[Mez 03] : Y. Mezouar and F. Chaumette, "Optimal camera trajectory with imagebased control," I. J. Robotic Res., vol. 22, no. 10-11, pp. 781–804, 2003.

[Che 08] : A. Cherubini, F. Chaumette, and G. Oriolo, "A position-based visual servoing scheme for following paths with nonholonomic mobile robots," in IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS 2008, Nice, France, September 2008.

[Fol 05] : D. Folio and V. Cadenat, "A controller to avoid both occlusions and obstacles during a vision-based navigation task in a cluttered environment," in Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on, Dec.

"A sensor-based controller able to treat total image loss and to guarantee non-collision during a vision-based navigation task," in Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference.

http://www.optique-ingenieur.org/fr/cours/OPI_fr_M04_C01/co/Grain_OPI_fr_M04_C01_1.html

http://www.cdta.dz/plateforme-robotique/

http://www.robotshop.com/blog/fr/nouveaux-capteurs-distance-laser-hokuyo-robotique-3825

http://www.generationrobots.com/fr/content/52-se-localiser-avec-un-telemetre-laser-hokuyo

[Dur 12] : Adrien Durand-Petite ville. Navigation référencée multi-capteurs d'un robot mobile en environnement encombré. Automatique / Robotique. Université Paul Sabatier - Toulouse III, 2012. Français.

[Ben 09] : Yannick Benezeth. Détection de la présence humaine par vision. Autre. Université d'Orléans, 2009. Français.

https://fr.wikipedia.org/wiki/Vision_par_ordinateur

https://fr.wikipedia.org/wiki/D%C3%A9tection_de_personnes

[Har 00] : I. Haritaoglu, D. Harwood, and L.S. Davis. W 4 : real-time surveillance of people and their activities. Pattern Analysis and Machine Intelligence, 22 :809 830, 2000

[Sta 99] : C. Stauer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. international conference on Computer Vision and Pattern Recognition, 2, 1999

[Sie 02] : N.T. Siebel and S. Maybank. Fusion of multiple tracking algorithms for robust people tracking. European Conference on Computer Vision, pages 373387, 2002.

[Son 04] : X. Song and R. Nevatia. Combined face-body tracking in indoor environment. International Conference on Pattern Recognition, 4 :159162, 2004.

[Jav 02] : O. Javed and M. Shah. Tracking and object classication for automated surveillance. European Conference on Computer Vision, pages 343357, 2002.

[Int 95] : S.S. Intille and A.F. Bobick. Closed-world tracking. International Conference on Computer Vision, pages 672678, 1995

[Tan 00] : C.Y. Tang, Z. Chen, and Y.P. Hung. Automatic detection and tracking of human heads using an active stereo vision system. International Journal of Pattern Recognition and Articial Intelligence, 14(2) :137166, 2000.

[Yas 05] : M. Yasuno, S. Ryousuke, N. Yasuda, and M. Aoki. Pedestrian detection and tracking in far infrared images. Computer Vision and Pattern Recognition Workshop, pages 182187, 2005.

[Dav 93] : J. Davis and M. Shah. Gesture recognition. Technical report, University of Central Florida, 1993.

[Meu 07] : M. Meuter, D. Muller, S. Muller-Schneiders, U. Iurgel, S. Park, and A. Kummert. Pedestrian tracking from a moving host using corner points. Advances in Visual Computing, pages 367376, 2007.

[Tsu 09] : Y. Tsuduki and H. Fujiyoshi. A method for visualizing pedestrian tra-c ow using sift feature point tracking. Pacic Rim Symposium on Advances in Image and Video Technology, 5414 :2536, 2009.

[Kim 06] : H.-J. Kim and KK.-M. Lee. Silhouette-based human motion estimation for movement education of young children. International Conference on Hybrid Information Technology, 2 :673678, 2006.

[Xia 08] : S. Xiang, F. Nie, Y. Song, and C. Zhang. Contour graph based human tracking and action sequence recognition. Pattern Recognition, 41(12) :36533664, 2008

[Zha 06] : Q. Zhao, J. Kang, H. Tao, and W. Hua. Part-based human tracking in a multiple cues fusion framework. International Conference on Pattern Recognition, 1 :450455, 2006.

[Har 00] : T. Harada, T. Sato, and T. Mori. Human motion tracking system based on skeleton and surface integration model using pressure sensors distribution bed. Workshop on Human Motion, pages 99106, 2000.

[Per 02] : P. Perez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. European Conference on Computer Vision, pages 661675, 2002.

[Ziv 04] : Z. Zivkovic and B.Krose. An em-like algorithm for color-histogram-based object tracking. international conference on Computer Vision and Pattern Recognition, pages 798803, 2004.

[Hun 95] : E. Hunter, J. Schlenzig, and R. Jain. Posture estimation in reduced-model gesture input systems. International Workshop on Automatic Face and Gesture Recognition, pages 290295, 1995.

[Den 07] : S. Denman, V. Chandrana, and S. Sridharan. An adaptive optical ow technique for person tracking systems. Pattern Recognition Letter, 28(10) :1232 1239, 2007.

[Ran 91] : K. Rangarajan and M. Shah. Establishing motion correspondence. international conference on Computer Vision and Pattern Recognition, pages 103108, 1991.

[Vee 01] : C. J. Veenman, M. J. T. Reinders, and E. Backer. Resolving motion correspondence for densely moving points. Pattern Analysis and Machine Intelligence, 23 :5472, 2001

[Sha 05] : K. Shafique and M. Shah. A non-iterative greedy algorithm for multi-frame point correspondence. Pattern Analysis and Machine Intelligence, pages 51 65, 2005.

[Liu 07] : G. Liu, X. Tang, J. Huang, J. Liu, and D. Sun. Hierarchical model-based human motion tracking via unscented kalman lter. International Conference on Computer Vision, pages 18, 2007

[Nic 01] : K. Nickels and S. Hutchinson. Model-based tracking of complex articulated objects. Robotics and Automation, 17 :2836, 2001.

[Bru 09] : D. Brulin and E. Courtial G. Allibert. Visual receding horizon estimation for human presence detection. People Detection and Tracking Workshop of the International Conference on Robotics and Automation Workshop, 2009.

[Jun 08] : G. Junxia, D. Xiaoqing, W. Shengjin, and W. Youshou. Full body trackingbased human action recognition. International Conference on Pattern Recognition, pages 14, 2008.

[Lin 07] : Z. Lin, L.S. Davis, D. Doermann, and D. DeMenthon. Hierarchical parttemplate matching for human detection and segmentation. International Conference on Computer Vision, pages 18, 2007

[Aya 00] : V. Ayala-Ramirez, C. Parra, and M. Devy. Active tracking based on hausdor matching. International Conference on Pattern Recognition, 4 :706709, 2000.

[Shi 94] : J. Shi and C. Tomasi. Good features to track. international conference on Computer Vision and Pattern Recognition, pages 593600, 1994

http://www.generationrobots.com/fr/content/55-ros-robot-operating-system

http://perso.ensi-bourges.fr/dfolio/index.php/ResearchTopics/VisualServo?userlang=fr

[Fol 07] : David Folio. Stratégies de commande référencées multi-capteurs et gestion de la perte du signal visuel pour la navigation d'un robot mobile. Automatique / Robotique. Université Paul Sabatier - Toulouse III, 2007.

https://fr.wikipedia.org/wiki/Kinect

http://www.gameblog.fr/article-lecteur_612_comment-fonctionne-la-technologie-kinect

https://fr.wikipedia.org/wiki/Calibration_de_cam%C3%A9ra

Home.http://openni.org/

Annexes

Annexe A

Modélisation du robot B21r

A.1 Présentation du Robot mobile B21r :

Le robot mobile *B21r* est une plate-forme expérimentale construite par la société *iRobot* pouvant se déplacer sur un terrain non accidenté ayant comme type de traction, la traction synchrone. Il dispose de quatre roues décentrées orientables tournant selon deux axes : une rotation selon l'axe y pour engendrer la translation, et une rotation selon l'axe verticale au sol pour engendrer une rotation sur lui-même. Ce robot mobile est muni de deux ceintures de capteurs à ultrasons, une ceinture de capteurs infrarouges, un laser, des capteurs tactiles placés le long de ses parois, et une caméra CCD N/B.

Fig. A.1 : Robot mobile B21r

A.2 Modélisation du système robotique

La détermination de la commande d'un système donné, nécessite le plus souvent la connaissance d'un modèle du système. Cette modélisation doit être menée avec rigueur, en vue d'obtenir le modèle le plus proche de la réalité afin de garantir par la suite de meilleures performances en termes de stabilité et de précision.

En effet, nous allons présenter une modélisation explicite et détaillée de la plateforme robotique utilisée. Pour ce faire nous définissons tout d'abord les repères suivants :

- le repère lié à la scène $R_W(W,X_W,Y_W, Z_W)$;
- le repère lié à la base mobile $R_R(R,X_R, Y_R, Z_R)$;

- et le repère lié à la Kinect $R_C(C,X_C, Y_C , Z_C)$.

De plus, nous notons par (t_x,t_y) les coordonnées de l'origine du repère lié à la Kinect dans le repère mobile.

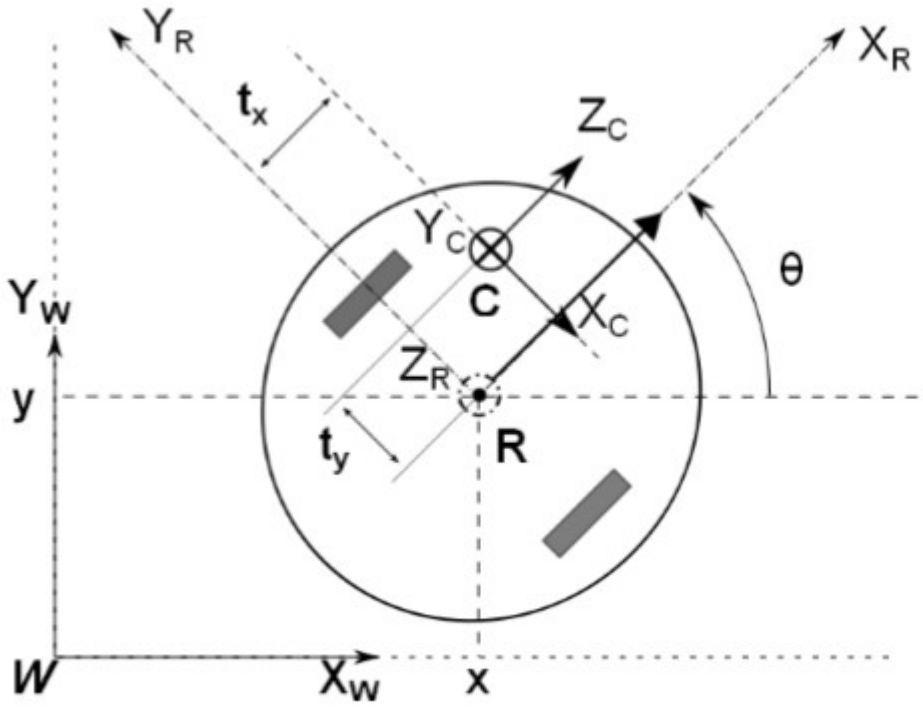


Fig. A.2: Repères du système Robot-Kinect

La position du robot mobile est définie par les coordonnées (x, y) du point R dans le repère R_W, par contre son orientation est repérée par l'angle ϑ entre l'axe X_W et l'axeX_R. En outr la position relative de la camera dans la base mobile est définie par le vecteur :

$$\overrightarrow{RC} = \begin{pmatrix} t_x \\ t_y \\ 0 \end{pmatrix} \qquad (A.1)$$

Le vecteur configuration du robot est défini comme suit :

$$q = \begin{pmatrix} l \\ \vartheta \end{pmatrix} \qquad (A.2)$$

Où l représente l'abscisse curviligne du robot dans le repère de la scène ainsi la dérivée par rapport au temps du vecteur configuration représente le vecteur de commande.

Dans la mesure où notre système robotique est commandable en vitesse, il y a lieu de définir la relation entre le torseur cinématique du robot noté Y_C et le vecteur commande

notéq̇.Par définition, ce torseur cinématique est constitué des composantes des vitesses de translation et de rotation ; il est alors comme suit :

$$\mathrm{Y_C} = [\mathrm{V_C}\ \omega_\mathrm{C}]^{T} \tag{A.3}$$

Avec :

$$\mathrm{V_C} = (\ V_{X_C} V_{Y_C} V_{Z_C}\ )^{T} \tag{A.4}$$

$$\omega_C = (\omega_{X_C} \omega_{Y_C} \omega_{Z_C})^{T} \tag{A.5}$$

La vitesse ${}^{C}V_{C,W}$ est déduite en exploitant la loi de composition des vitesses :

$${}^{C}V_{W,C} = {}^{C}V_{R,C} + {}^{C}V_{W,R} + {}^{C}\omega_R \wedge {}^{C}\overrightarrow{RC} \tag{A.6}$$

La Kinect est repérée par le vecteur ${}^{C}\overrightarrow{RC} = (-t_y \quad 0 \quad t_x)^{T}$ et comme elle est fixée sur le robot par conséquent sa vitesse, par rapport au repère lié au robot, est nulle et donc :

$${}^{C}V_{R,C} = \frac{d}{dt}\ {}^{C}\overrightarrow{RC} = 0 \tag{A.7}$$

En outre, le robot est repéré dans la scène par le vecteur $\overrightarrow{\mathrm{WR}}$ et sa vitesse par rapport à la scène est alors donnée par :

$${}^{C}V_{W,R} = \frac{d\ {}^{C}\overrightarrow{WR}}{dt} \tag{A.8}$$

Donc :

$${}^{C}V_{W,R} = \begin{pmatrix} 0 \\ 0 \\ v \end{pmatrix} \tag{A.9}$$

Sachant que ${}^{C}\overrightarrow{RC} = (0 \quad 0 \quad l)^{T}$ et $v = \frac{dl}{dt}$ le calcul du terme ${}^{C}\omega_R$ conduit à

$${}^{C}\omega_R = \begin{pmatrix} 0 \\ \dot{\vartheta} \\ 0 \end{pmatrix} \tag{A.10}$$

En remplaçant les termes dans l'équation on obtient le vecteur suivant :

$$^{C}V_{W,R} = \begin{pmatrix} \dot{\vartheta} tx \\ 0 \\ v + \dot{\vartheta} ty \end{pmatrix} \quad \text{(A.11)}$$

Par conséquent, le torseur cinématique de la Kinect, par rapport à Roet projeté dans le repère Kinect, est exprimé par:

$$\begin{pmatrix} V_{X_C} \\ V_{Y_C} \\ V_{Z_C} \\ \omega_{X_C} \\ \omega_{Y_C} \\ \omega_{Z_C} \end{pmatrix} = \begin{pmatrix} 0 & tx \\ 0 & 0 \\ 1 & ty \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} v \\ \dot{\vartheta} \end{pmatrix} \quad \text{(A.12)}$$

Donc la matrice Jacobienne est telle que :

$$J = \begin{pmatrix} 0 & tx \\ 0 & 0 \\ 1 & ty \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{(A.13)}$$

La forme réduite du Jacobien est obtenue en ne considérant que les mouvements réalisables par la Kinect donc ne sont prises en compte que la translation le long de X_k et celle le long de l'axe Z_k en plus de la rotation autour de l'axe Y_k d'où on obtient :

$$J_r = \begin{pmatrix} 0 & tx \\ 1 & ty \\ 0 & 1 \end{pmatrix} \quad \text{(A.14)}$$

Annexe B

Modèle de la caméra

B.1 : Modèle de la Caméra

Le modèle sténopé modélise une caméra par une projection perspective. Ce modèle transforme un point 3D de l'espace M en un point-image m et peut se décomposer en trois transformations élémentaires successives (cf. figure 2) : la transformation entre le repère du monde R_W et celui de la caméra R_C T_1, la transformation entre le repère caméra et le repère R_r (plan rétinien) T_2 et la transformation entre le repère capteur et le repère image.

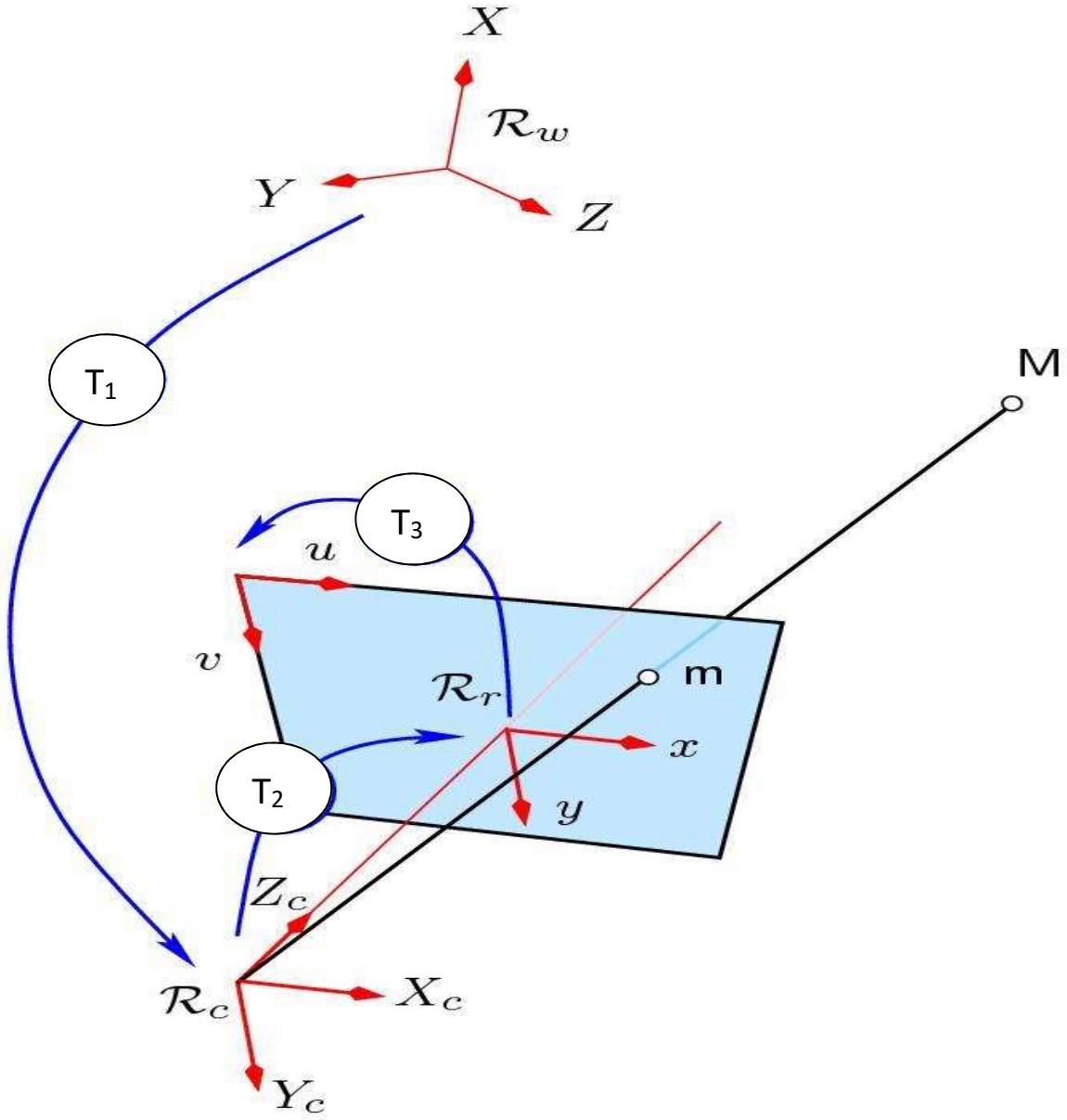


Fig. B.1 : Les transformations élémentaires du modèle sténopé et leurs repères

B.2 : Coordonnées homogènes :

En vision par ordinateur, on utilise souvent les coordonnées homogènes :

En 3D :

$$M = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \longrightarrow M' = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad \text{(B.1)}$$

Coordonnées euclidiennes Coordonnées homogènes

Transformation entre le repère monde et le repère caméra

Comme indiqué sur la **Figure B.1**, T_1représente une transformation entre le repère du monde R_w (choisi arbitrairement) et le repère caméra R_C (dont l'origine est située au centre optique de la caméra). Cette transformation rigide peut se décomposer en une rotation R et une translation. Les paramètres de cette transformation sont appelés paramètres extrinsèques de la caméra.

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R & p \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = T_1 \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad \text{(B.2)}$$

Avec :

$$p = \begin{bmatrix} p_X \\ p_y \\ p_z \end{bmatrix} \quad ; \quad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \text{(B.3)}$$

T1 est une matrice 4 × 4.

Transformation entre le repère caméra et le repère capteur (plan rétinien)

La deuxième transformation, notée T_2 sur la figure 2 relie le repère caméra R_C au repère capteur R_r (plan rétinien). C'est une projection perspective (matrice 3×4) qui transforme un point 3D (X_C, Y_C, Z_C)en un point-image (x, y) (en unité métrique).

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 00 \\ 0 & f & 00 \\ 0 & 0 & 10 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = T_2 \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} \quad \text{(B.4)}$$

Oùf désigne la focale de l'objectif utilisé.

Transformation entre le repère capteur et le repère image

La troisième et dernière transformation, notée T_3 sur la **Figure B.1**, décrit l'opération de conversion des coordonnées images (x, y)(en unité métrique) en coordonnées images discrètes(u, v)(pixels).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & k_x cot\theta & c_x + c_y cot\theta \\ 0 & k_y/sin\theta & c_y/sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T_3 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{(B.5)}$$

Où :

- c_x et c_y (en pixels) désignent les coordonnées de l'intersection de l'axe optique avec le plan image (théoriquement au centre de l'image)
- k_x et k_y désignent le nombre de pixels par unité de longueur suivant les directions $x$et $y$du capteur respectivement ($k_x = k_y$ dans le cas de pixels carrés)
- Θ traduit la non orthogonalité éventuelle des lignes et colonnes de l'image. En pratique, θ est très proche de π/2. Ce paramètre est désigné par « skew factor » en anglais.

On considère souvent que le « skew factor » est négligeable θ = π/2 et l'équation (3) se simplifie alors de la façon suivante :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T_{3_{simplifiée}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{(B.6)}$$

B.3 : Modèle sténopé complet :

La composition des trois transformations T_1, T_2 et T_3 peut être résumée par le schéma de la figure 3.

$$(X, Y, Z) \overset{T1}{\rightarrow} (X_c, Y_c, Z_c) \overset{T2}{\rightarrow} (x, y) \overset{T3}{\rightarrow} (u, v)$$

Cela conduit à l'équation du modèle sténopé :

$$m' = T_1 T_2 T_3 \, M' \qquad \text{(B.7)}$$

Avec :

$$K = T_3 T_2 = \begin{bmatrix} k_x & k_x \, cot\theta & c_x + c_y cot\theta \\ 0 & k_y / sin\theta & c_y / sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} f_x & f_x \, cot\theta & c_x + c_y cot\theta & 0 \\ 0 & f_y / sin\theta & c_y / sin\theta & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \text{(B.8)}$$

Où $f_x = f k_x$ et $f_y = f k_y$ désignent la focale de la caméra en nombre de pixels suivant les directions x et y respectivement.

Les 5 paramètres $(c_x c_y f_x f_y \, \theta)$ de la matrice K sont appelés paramètres intrinsèques de la caméra.

Dans le cas où le « skew factor » est négligé, le modèle sténopé, qui relie les coordonnées 3D (X, Y, Z) d'un point exprimé dans le repère du monde aux coordonnées 2D (u, v)de sa projection dans le plan image (point-image = pixel), est souvent écrit de la façon suivante :

$$u = f_x \frac{r_{11} X + r_{12} Y + r_{13} Z + p_x}{r_{31} X + r_{32} Y + r_{33} Z + p_z} + c_x \qquad \text{(B.9)}$$

$$v = f_y \frac{r_{21} X + r_{22} Y + r_{23} Z + p_y}{r_{31} X + r_{32} Y + r_{33} Z + p_z} + c_y \qquad \text{(B.10)}$$

Ces relations sont parfois désignées par le terme relations de colinéarité.

Annexe C

Présentation de la Kinect de Microsoft

C.1 Description matérielle de la Kinect de Microsoft :

C.1.1 Généralités :

Kinect est un accessoire doté de deux caméras et de micros. La première caméra permet de **capturer l'image en couleur** tandis que la seconde permet grâce à un capteur infrarouge de **capter la profondeur.**

La combinaison de tout ça permet à Kinect de **visualiser les personnes présentes devant l'écran** ainsi que leur position physique, même en profondeur. Un logiciel s'occupe ensuite d'interpréter ces données pour les transposer à l'écran, en arrivant ainsi à **reproduire le squelette d'un joueur** via une vingtaine de points (articulations et extrémités du corps).

Description de la Kinect (Hard)

C.1.2 Le Microphone :

Kinect embarque 2 microphones ainsi que 4 détecteurs digitaux externes de sources audio. La combinaison de l'ensemble de ce système audio permet ainsi à Kinect de détecter **la localisation spatiale d'une source sonore** mais aussi **d'éliminer les bruits de fond parasites** grâce à un traitement de données.

C.1.2 La Camera RGB (Red Green Blue)

La première des deux caméras embarquées dans la technologie Kinect est une **caméra couleur RGB** avec un capteur photographique **de type CMOS**. Elle se situe au centre de la barre horizontale (Fig. C.1).

La lumière est composée de 3 couleurs primaires qui sont le Rouge, le Vert et le Bleu comme illustrées sur la Figure 3. Chaque couleur est donc définie selon le système RGB c'est-à-dire en fonction de la proportion respective en rouge, vert et bleu. Le mélange de ces trois couleurs compose aussi chacun des pixels de l'image.

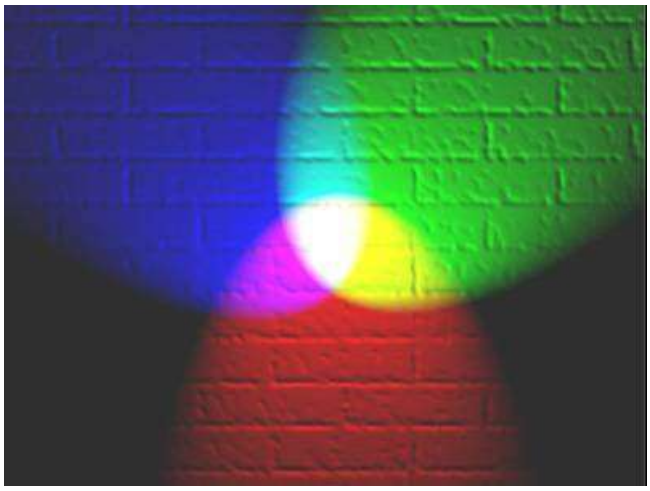


Fig. C.1 : Les couleurs primaires (rouge – bleu – vert)

Un **capteur photographique** est un composant électronique photosensible servant à convertir un rayonnement électromagnétique (UV, visible ou IR) en un signal électrique analogique. Ce signal est ensuite amplifié, puis numérisé par un convertisseur analogique-numérique et enfin traité pour obtenir une image numérique.

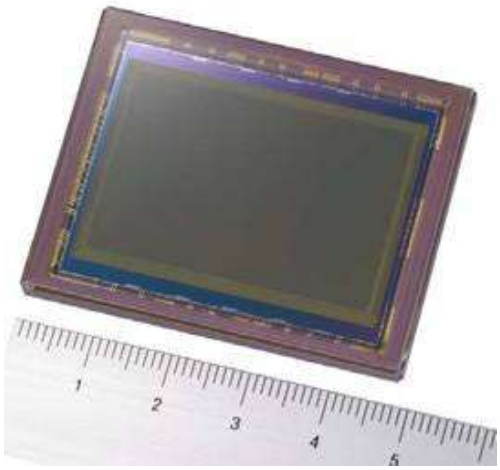


Fig. C.2 Exemple d'un capteur photographique de type CMOS développé par Sony.

La lumière (l'image) arrive en face du capteur, celle-ci est d'abord purifiée par un filtre Infra Rouge (bloquant les ondes infrarouges et laissant passer la couleur), puis traverse un « mini filtre » de couleur rouge, vert ou bleu placé en face du capteur lui-même. Ainsi les millions de

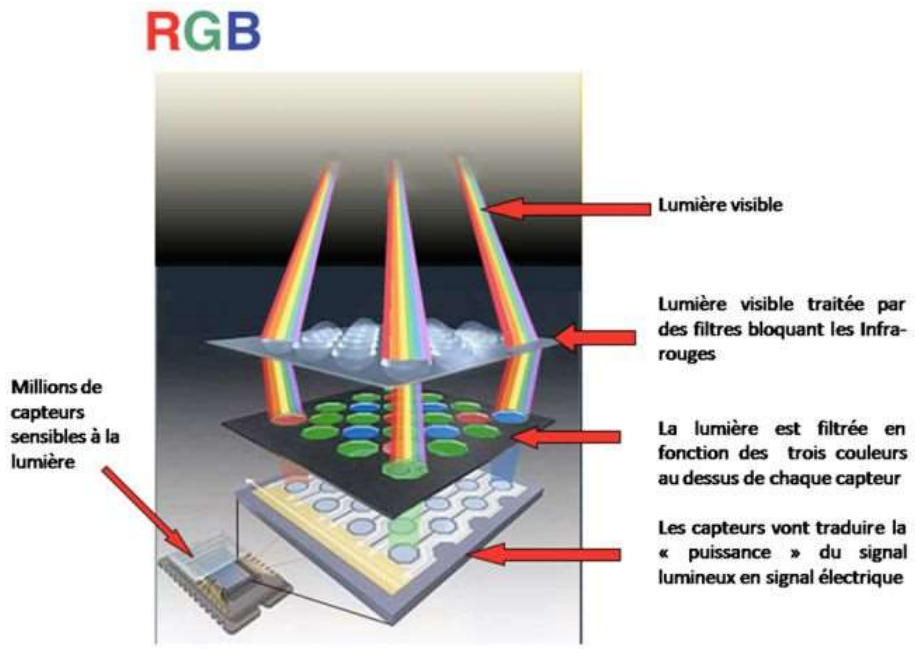


Fig. C.3 : Fonctionnement d'un capteur photographique de type CMOS pour les appareils photos numériques et les cameras couleur.

capteurs présents à la surface du CMOS vont émettre un signal électrique relatif à une des 3 couleurs, qui sera ensuite converti numériquement.

C.1.3 Le capteur de profondeur :

Une image de profondeur se présente comme une image monochromatique (c'est-à-dire à niveaux de gris) qui, pour un ensemble d'éléments d'une image (appelés pixels) arrangés dans un tableau à deux dimensions, associe une valeur représentative de la distance physique entre le point de la scène et la caméra.

La Kinect est une caméra 3D active de type stéréoscopique Comme le montre la figure1, elle comprend deux parties :

– une source de lumière infrarouge « structurée » ;

– une caméra infrarouge.

En regardant l'image des profondeurs, on l'interprète comme une image en noir et blanc distordue. Elle semble étrange parce que la couleur des différentes parties de l'image n'indiquent pas à quel point les objets sont lumineux, mais à quelle distance ils sont. Les parties les plus claires indiquent les objets proches, les plus sombres les plus lointains. Ce type d'image facilite l'interprétation d'une scène à un ordinateur.

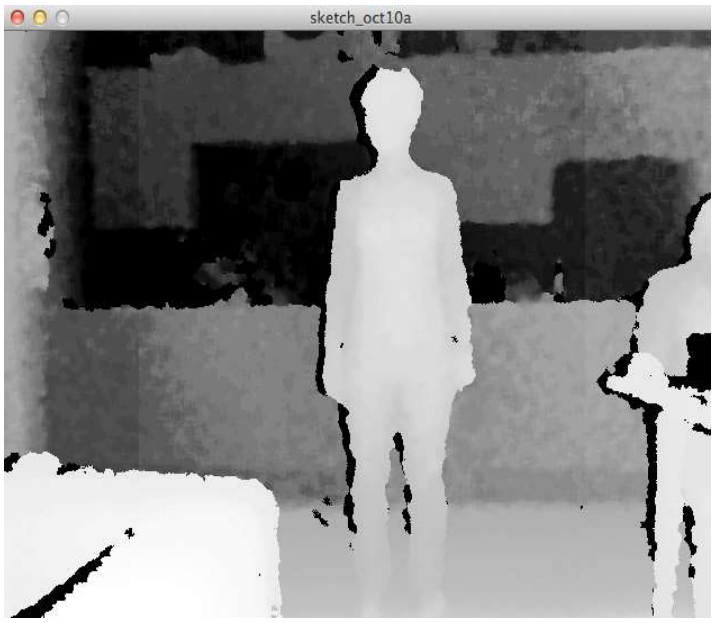


Fig. C.3 Image de profondeur perçue par la Kinect

L'œil qui est plus excentré est le projecteur infrarouge. La lumière infrarouge a une longueur d'onde qui ne la rend pas perceptible pour l'œil humain. Le projecteur infrarouge de la Kinect projette une grille de points lumineux sur toute la scène en face de lui. Même si ces points nous sont invisibles, ils peuvent être détectés grâce à une caméra infrarouge, ce dont dispose également la Kinect.

L'avantage de l'utilisation de la lampe IR est de pouvoir jouer dans toutes les conditions de luminosité. Même si certaines fonctionnalités devraient être affectées comme la reconnaissance faciale ou le scan des objets (utilisant la caméra RGB, dépendante du spectre de lumière visible).

C.2 Description logicielle de la Kinect (Soft) :

La Kinect est munie d'un port USB classique qui permet de récupérer les images en couleur et les images de profondeur par logiciel. Mis à part la librairie développée par Microsoft, toutes les librairies mentionnées sont multiplateformes et fonctionnent sous GNU/Linux, Mac OSX et Microsoft Windows.

PrimeSense, l'entreprise ayant réalisé le capteur de profondeur utilisé par la Kinect, a publié Open NI qui est un framework open source permettant de développer des applications utilisant des interactions naturelles (voix, mouvements du corps, etc.). Ce framework est accompagné d'un driver open source pour la Kinect et d'un middleware propriétaire appelé NITE qui s'occupe à proprement parler du traitement des images de profondeur. Le middleware NITE

permet de segmenter les différentes personnes en face de la caméra, d'obtenir leur pose ou encore de reconnaître certains gestes. Le framework Open NI est conçu pour être indépendant des capteurs utilisés et des algorithmes de traitement. En pratique cependant, seules les caméras de profondeur basées sur la technologie de PrimeSense sont supportées et il n'existe toujours que le middleware propriétaire NITE pour utiliser les fonctions avancées de traitement.
Trois caméras de profondeur basées sur la même technologie que la Kinect sont disponibles sur le marché :
– la Kinect pour Xbox, première caméra du genre,
– la caméra ASUS XtionPRO, et
– la Kinect for Windows, une version adaptée de la Kinect pour Xbox, permettant notamment d'obtenir la profondeur de scènes « proches ».

C.2.1 La librairie OpenNI

Le framework OpenNI est une couche abstraite capable de s'interfacer avec différents types de matériel et fournissant des fonctions pour le développement d'applications utilisant des interactions naturelles. Pour un fonctionnement correct, il faut installer des modules implémentant certaines couches ou fonctionnalités du framework.
En principe, OpenNI supporte quatre types de matériel :
– des capteurs 3D tels que la Kinect ;
– des caméras RGB comme celle intégrée dans la Kinect ;
– des caméras infrarouges ;
– des appareils audio (microphone ou réseau de microphones).

Par exemple, le module SensorKinect permet d'utiliser la Kinect avec OpenNI.
Les composants logiciels « middleware » implémentent des fonctions de traitements des données acquises grâce aux différents capteurs. OpenNI propose des fonctions correspondant à quatre types de traitements :
– l'analyse du corps : récupération de la pose du corps sous forme d'une information sur la position et l'orientation des différentes articulations du corps humain ;
– l'analyse du pointage avec la main : permet de suivre la position d'une main pointant vers la caméra ;
– la détection de certains gestes particuliers comme, par exemple, deux mains qui applaudissent;
– l'analyse de la scène : extraction de l'avant-plan, segmentation du sol, etc.

Il n'existe pour le moment que l'implémentation fournie par le middleware propriétaire NITE.

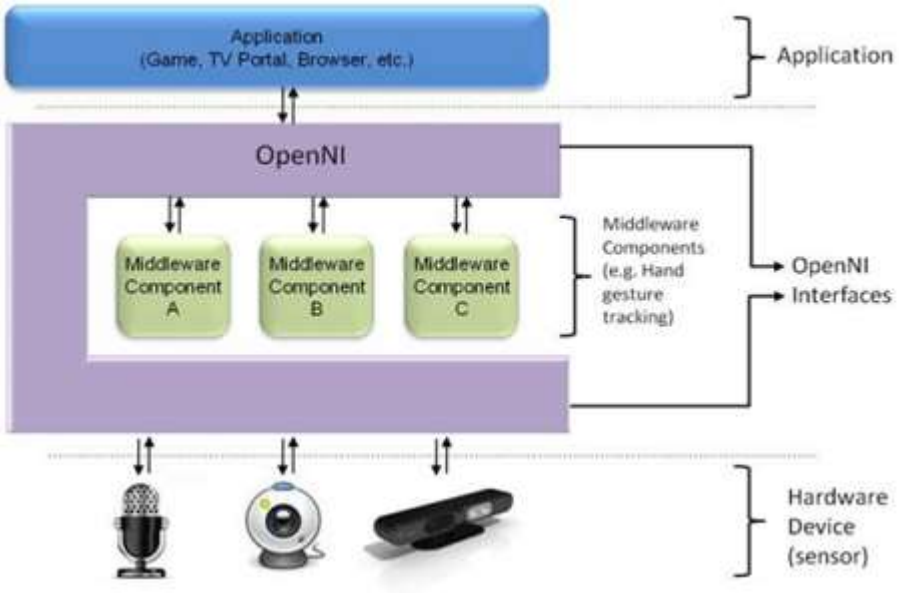


Fig. C.4 l'API d'Open NI qui est organisée autour de deux concepts essentiels

C.3 Télémètre laser embarqué sur la Kinect :

C.3.1 Caractéristiques du télémètre :

Les télémètres lasers Hokuyo permettent de mesurer et de cartographier l'environnement.

Spécifications :

- Modèle : UST-05LN
- Tension d'alimentation DC 12V / DC 24V (plage de fonctionnement de 10 à 30V, ondulation à moins de 10%) Fournir 150mA courant continu (DC 24V) ou moins (lors du démarrage à propos de 400mA est nécessaire.)
- Source de lumière laser semi-conducteur (905nm),
- Laser de classe 1 (IEC60825-1: 2007, numéro d'accession: 1420210-000)
- Plage de détection et de l'objet 60mm à 5000mm (feuille de Kent blanc) 60mm à 2000mm (réflectance diffuse 10%)
- Taille minimale détectable (change en fonction de la distance) 130mm
- Précision 60mm à 5000mm ± 40mm
- L'écart type σ <20mm * 2

- Angle de balayage 270 °
- Vitesse de numérisation (moteur de 2400rpm vitesse) 25ms
- Résolution angulaire 0,5 °
- Temps de démarrage en 10 secondes (temps de démarrage diffère si un dysfonctionnement est détecté pendant le démarrage)
- Les sorties

Photo-coupleur, sortie à collecteur ouvert Max DC 30V 50mA
Sortie 1 : sortie 1 OFF lors de la détection d'objet
Sortie 2 : Sortie 2 OFF lors de la détection d'objet
Sortie 3 : Sortie 3 OFF lors de la détection d'objet
Sortie de dysfonctionnement : ON pendant le fonctionnement normal, OFF en cas de panne
Note : Sortie 1 à 3 sont mis hors tension pendant l'état de dysfonctionnement

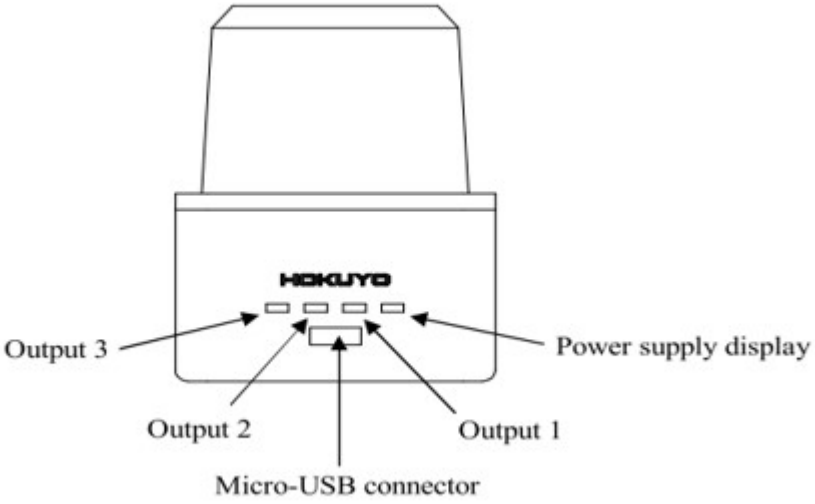


Fig. C.5 Affichage des LED

Figure et le tableau indiquent la relation entre la position de l'objet détecté et les états de sortie.

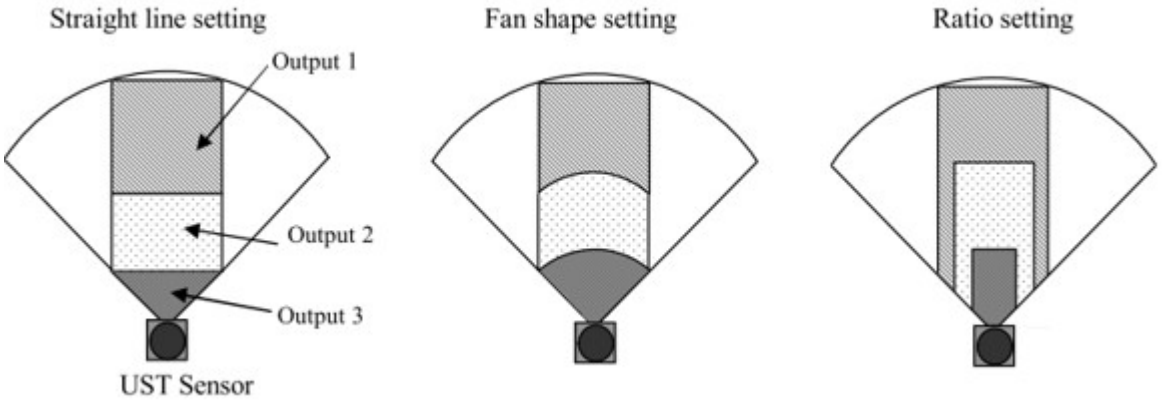


Fig. C.6 : les zones de configuration

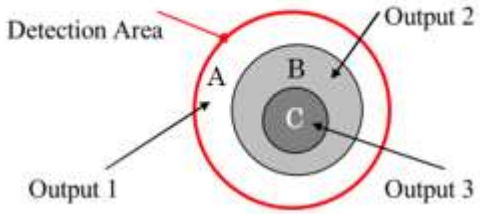


Fig. C.7 les zones de sortie

Position de l'objet	Sortie 1	Sortie 2	Sortie 3
A	OFF	ON	ON
B	OFF	OFF	ON
C	OFF	OFF	OFF

Tableau la relation entre la position de l'objet et l'état de la sortie

C.3 généralités sur ROS (Robot Operating System) :

C.3.1 Historique et généralités :

ROS a été initialement développé en 2007 sous le nom « switchyard » par le Stanford Artificial Intelligence Laboratory dans le cadre du projet Stanford AI Robot STAIR (Stanford AI Robot).

De 2008 à 2013, le développement a été effectué principalement par Willow Garage, un institut / incubateur de recherche en robotique. En février 2013, le développement de ROS est poursuivi par l'Open Source Foundation Robotics. En aout 2013, un blog annonce que Willow Garage sera absorbée par une autre société créée par son fondateur, Suitable Technologies. Le support du robot PR2 créé par Willow Garage a été par la suite repris par Clear path Robotics.

Robot Operating System (ROS) est une plateforme de développement logicielle pour robot. Il s'agit d'un méta-système d'exploitation (entre un OS et un middleware) qui peut fonctionner sur un ou plusieurs ordinateurs et qui fournit plusieurs fonctionnalités : abstraction du matériel, contrôle des périphériques de bas niveau, mise en œuvre de fonctionnalités couramment utilisées, transmission de messages entre les processus et gestions des packages installés.

Le logiciel ROS peut être séparé en trois groupes :

1. les outils utilisés pour créer, lancer et distribuer des logiciels basés sur ROS (roscore, roslaunch, catkin)
2. les clients ROS pour des langages : roscpp (C++) et rospy (python)
3. les packages contenant des programmes pour ROS utilisant un ou plusieurs clients ROS

Les outils et les principaux clients ROS (roscpp et rospy) sont publiés sous les termes de la licence BSD*. De nombreux packages sont publiés pour ROS avec diverses licences open source (BSD, MIT*). Ces packages permettent de lancer des applications, des algorithmes ou encore des programmes pour interfacer ROS avec des robots.

Il existe des clients ROS non officiels. Parmi eux on peut notamment citer :

- rosjava (Java) qui a permis de faire fonctionner ROS sur le système d'exploitation Android.
- roslibjs (JavaScript) qui permet d'interagir avec un système ROS depuis un navigateur. Ce client est développé dans le cadre du Robot Web Tools effort.

C.3.2 Les concepts de ROS

La philosophie derrière le système d'exploitation ROS se résume dans les cinq grands principes suivants :

- Peer to Peer ou Egale à Egale : un robot suffisamment complexe est composé de plusieurs ordinateurs ou cartes embarquées reliées par Ethernet ainsi que parfois des ordinateurs externes au robot pour des tâches de calcul intensif. Une architecture peer to peer couplée à un système de tampon (buffering) et un système de lookup (un name service appelé master dans ROS), permet à chacun des acteurs de dialoguer en direct avec un autre acteur, de manière synchrone ou asynchrone en fonction des besoins.

- Multi langage : ROS est neutre d'un point de vue langage et peut être programmé en différents langages. La spécification de ROS intervient au niveau message. Les connexions peer to peer sont négociées en XML-RPC qui existe dans un grand nombre de langages. Pour supporter un nouveau langage, soit on redéfinit les classes C++ (ce qui a été fait pour le client Octave par exemple), soit on écrit les classes permettant de générer les messages. Ces messages sont décrits en IDL (Interface Definition Language).
- Basé sur des outils : plutôt qu'une exécution (runtime) monolithique, ROS a adopté un design microkernel qui utilise un grand nombre de petits outils pour compiler (build) et exécuter (run) les différents programmes. Dans Ros chaque commande qui est en vérité un exécutable, permet de manipuler les nœuds et les messages. . L'avantage de cette solution est qu'un problème sur un exécutable n'affecte pas les autres, rendant le système plus robuste et plus évolutif qu'un système basé sur un runtime centralisé.

- Léger : afin de lutter contre le développement d'algorithmes plus ou moins liés avec l'OS robotique et donc difficilement réutilisables ensuite, les développeurs de ROS souhaitent que les pilotes et autres algorithmes soient contenus dans des exécutables indépendants. Cela assure la réutilisabilité maximale et surtout le maintien d'une taille réduite. Ce mécanisme rend ROS facile d'usage, la complexité se trouvant dans les bibliothèques. Cette organisation facilite en plus le test unitaire. Enfin, ROS utilise des codes qui sont des pilotes et algorithmes issus d'autres projets open source (licence libre) :

- simulateur Player/stage ;
- bibliothèques de traitement d'image et de vision artificielle : OpenCV ;
- algorithme de planification : OpenRave ;

- Gratuit et open source (licence libre) : nous avons déjà expliqué les raisons de ce choix avant. Notons toutefois que l'architecture choisie est cohérente avec ce choix : ROS passe des données grâce à de la communication interprocessus. De ce fait, les modules n'ont pas besoin d'être liés dans un unique processus, facilitant ainsi l'usage des différentes licences.

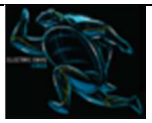
C.3.3 Le système de fichiers de ROS :

Les ressources de ROS sont organisées dans une structure hiérarchique sur disque. Deux concepts importants se détachent :

- Le package : C'est l'unité principale d'organisation logicielle de ROS. Un package est un répertoire qui contient les nœuds, les librairies externes, des données, des fichiers de configuration et un fichier de configuration xml nommé package.xml.
- La pile (méta-package) : Une pile (méta-package) est une collection de packages. Elle propose une agrégation de fonctionnalités telles que la navigation, la localisation… Une pile (méta-package) est un répertoire qui contient les répertoires des packages ainsi qu'un fichier de configuration nommé package.xml.

En plus de ces deux notions très importantes, on relève également la notion de distribution. Une Distribution, comme dans Linux, est un ensemble de versions, qui sont classées par ordre chronologique comme suit :

Les versions de ROS peuvent être incompatibles entre-elles et sont souvent désignées par leur nom de code plutôt que leur numéro de version.

Distribution	Date de publication	Image	Date de fin de vie
Jade	23 mai 2015		30 mai 2017
Indigo	22 juillet 2014		30 avril 2019
Hydro	4 septembre 2013		31 mai 2014
GroovyGalapagos	31 décembre 2012		31 juillet 2014
FuerteTurtle	23 avril 2012		
Electric Emys	30 août 2011		
Diamondback	2 mars 2011		
C Turtle	3 août 2010		
Box Turtle	2 mars 2010		
ROS 1.0	22 janvier 2010		

C.3.4 Les notions de base de ROS :

ROS regroupe plusieurs fonctionnalités qui facilitent le développement d'applications souple et modulable pour la robotique.

Architecture de communication

ROS offre une architecture souple de communication interprocessus et inter-machine. Les processus ROS sont appelés des nœuds et chaque nœud peut communiquer avec d'autres via des topics. La connexion entre les nœuds est gérée par un master et suit le processus suivant :

1. Un premier nœud averti le master qu'il a une donnée à partager
2. Un deuxième nœud averti le master qu'il souhaite avoir accès à une donnée
3. Une connexion entre les deux nœuds est créée
4. Le premier nœud peut envoyer des données au second

Un nœud qui publie des données est appelé un Publisher et un nœud qui souscrit a des données est appelé un subscriber. Un nœud peut être à la fois Publisher et subscriber. Les messages envoyés sur les topics sont pour la plupart standardisés ce qui rend le système extrêmement flexibles.

ROS permet une communication inter-machine, des nœuds s'exécutant sur des machines distinctes, mais ayant connaissance du même master peuvent communiquer de manière transparente pour l'utilisateur.

Le principe de base d'un OS robotique est de faire fonctionner en parallèle un grand nombre d'exécutable qui doivent pouvoir échanger de l'information de manière synchrone ou asynchrone. Par exemple, un OS robotique doit interroger à une fréquence définie les capteurs du robot (capteur de distance à ultrasons ou infrarouge, capteur de pression, capteur de température, gyroscope, accéléromètre, caméras, microphones…), récupérer ces informations, les traiter (faire ce que l'on appelle la fusion de données), les passer à des algorithmes de traitement (traitement de la parole, vision artificielle, localisation et cartographie simultanée,…) et enfin contrôler les moteurs en retour. Tout ce processus s'effectue en continue et en parallèle. D'autre part, l'OS robotique doit assurer la gestion de la concurrence afin d'assurer l'accès efficace aux ressources du robot.

Nous décrivons ci-dessous les concepts regroupés dans ROS sous le nom de « ROS Computation Graph » et qui permettent d'atteindre ces objectifs. Il s'agit des concepts utilisés par le système en cours de fonctionnement tandis que le « le système de fichiers de ROS » décrit dans le paragraphe précédent correspond aux concepts statiques.

i. Les nœuds

ROS répond à tout cette problématique grâce à des notions de base simples. La première notion est la notion de **nœud**.

Dans ROS, un nœud est une instance d'un exécutable. Un nœud peut correspondre à un capteur, un moteur, un algorithme de traitement, de surveillance… Chaque nœud qui se lance se déclare au **Master**. On retrouve ici l'architecture micro kernel où chaque ressource est un nœud indépendant.

ii. Le Master

Le **Master** est un service de déclaration et d'enregistrement des nœuds qui permet ainsi à des nœuds de se connaitre et d'échanger de l'information. Le Master est implémenté via XMLRPC.

Le Master comprend une sous-partie très utilisée qui est le **Parameter Server**. Celui-ci, également implémenté sous forme de XMLRPC, comme son nom l'indique est une sorte de base de données centralisée dans laquelle les nœuds peuvent stocker de l'information et ainsi partager des paramètres globaux.

iii. Les topics

Un **topic** est un système de transport de l'information basé sur le système de l'abonnement / publication (subscribe / publish). Un ou plusieurs nœuds pourront publier de l'information sur un **topic** et un ou plusieurs nœuds pourront lire l'information sur ce **topic**. Le **topic** est en quelque sorte un bus d'information asynchrone un peu comme un flux RSS. Cette notion de bus many-to-many asynchrone est essentielle dans le cas d'un système distribué.

Le **topic** est typé, c'est-à-dire que le type d'information qui est publiée (le **message**) est toujours structuré de la même manière. Les nœuds envoient ou reçoivent des messages sur des topics.

iv. Les messages

Un **message** est une structure de donnée composite. Un message est composé d'une combinaison de types primitifs (chaines de caractères, booléens, entiers, flottants…) et de message (le message est une structure récursive). Par exemple un nœud représentant un servomoteur du robot, publiera certainement son état sur un topic (selon ce que vous aurez programmé) avec un message contenant par exemple un entier représentant la position du moteur, un flottant représentant sa température, un autre flottant représentant sa vitesse…

La description des messages est stockée dans *nom_package/msg/monMessageType.msg*. Ce fichier décrit la structure des messages.

v. Les services

Le topic est un mode de communication asynchrone permettant une communication many-to-many. Le **service** en revanche répond à une autre nécessité, celle d'une communication synchrone entre deux nœuds.

vi. Les bags

Les « bags » sont des formats pour stocker et rejouer les messages échangés. Ce système permet de collecter par exemple les données mesurées par les capteurs et les rejouer ensuite autant de fois qu'on le souhaite afin de faire de la simulation sur des données réelles. Ce système est également très utile pour déboguer un système a posteriori.

L'outil rxbag permet de visualiser graphiquement les données enregistrées dans les fichiers bag.