REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SURERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE

Département d'Automatique

Mémoire de fin d'études

En vue de l'obtention du diplôme d'ingénieur d'état en Automatique

Thème :

Implémentation d'une technique SLAM VISUEL 3D dans un environnement d'intérieur : Cas du B21r

Présenté par:

BENYOUCEF Rayane **MERADI Dounia**

Soutenu le 20 / 06 /2016 devant le jury composé de :

Président :	***Mr ILLOUL Rachid***	***Docteur à l'ENP***
Examinateur :	***Mr ABDAOUEL Lahcen***	***Docteur à l'ENP***
Encadreurs :	***Mr CHAKIR Messaoud***	***Docteur à l'ENP***
	Mme CHAIB Khaoula	***Assistante de recherche au CDTA***
Invité :	***Mr OUADAH Nourdine***	***Sous-directeur de recherche au CDTA***

ENP 2016

الملخص:

قيام الروبوت المتحرك باستكشاف بيئة ديناميكية غير معروفة، تحديد الموقع وبناء الخريطة في نفس الوقت يعد كمشكلة الدجاجة أو البيضة، والمعروف باسم التزامن في تحديد الموقع ورسم الخرائط فقد كان حقلا نشطا جدا لأكثر من عشرين عاما. لدينا الآن الحلول لمعالجة مشكلة رسم خرائط ثنائية البعد التحدي الجديد هو رسم خرائط ثلاثية البعد وهذا هو الهدف من هذه الدراسة

في هذا المشروع، سنقوم بتقديم لمحة عامة عن مشكلة التزامن في تحديد الموقع ورسم الخرائط البصرية ثم نعالج مشكلة بناء الخرائط ثلاثية البعد من البيانات المسجلة بواسطة جهاز استشعار كينكت

الكلمات المفتاحية: التزامن البصري في تحديد الموقع ورسم الخرائط، جهاز الاستشعار RGB–D ،

الرسم البياني المثالي، كاميرا العمق ، كينيكت، روبوت متحرك ، النقاط المهمة، ICP، ORB

Abstract

For a mobile robot exploring an unknown dynamic environment, localizing itself and building a map at the same time is a chicken-or-egg problem, known as Simultaneous Localization And Mapping (SLAM). It has been a very active field of research for more than twenty years. Now we have solutions to address the 2D SLAM problem. The new challenge is to tackle the 3D SLAM problem. This is the goal of this study.

In this project we provide an overview of the visual SLAM problem then we address the problem of building 3D maps from the data recorded by Kinect sensor. This problem is challenging because it requires the sensor data to be interpreted correctly into the map, minimizing the error while aligning observations to achieve consistency.

Key–Words: Visual SLAM, RGB-D sensor, Graph Optimization, *Depth camera, Kinect, Mobile Robotic, Keypoints, ORB, ICP.*

Résumé

Pour un robot mobile, explorer un environnement dynamique inconnu, se localise et construire une carte en même temps est un problème de poule-œuf connu sous le nom Simultaneous Localization et de cartographie, C’est un domaine de recherche très actif depuis plus de vingt ans et il est maintenant possible de faire du SLAM en 2D. Le nouveau challenge est de faire le SLAM en 3D. C’est le but de cette étude.

Dans ce projet, nous présentons un aperçu du problème de SLAM visuel puis nous abordons le problème de la construction des cartes en 3D à partir des données acquises par le capteur Kinect. Ce problème est difficile, car il nécessite des données de capteur interprétées correctement pour la construction de la carte, ce qui minimise l'erreur, tout en alignant les observations pour obtenir plus de cohérence.

Mots-clés: SLAM visuel, capteur RGB-D, Optimisation du Graph, caméra de profondeur, Kinect, Robot Mobile, Points d’intérêt, ORB, ICP

À l'homme le plus cher au monde mon père,

À mon trésor ma mère,

À mon frère et mes Sœurs,

À mes cousins Imad et Moetaz,

A mes chers amies Nour El-Hoda Gabour et Aicha Laidoudi

À toute la Famille MERADI et BAHRI

À tous mes enseignants,

À mes amis et camarades.

Dounia MERADI

Dédicace

Après avoir rendu grâce à DIEU LE TOUT PUISSANT,

Je dédie ce modeste travail à :

Mes chers parents ;

Mes frères;

Ainsi qu'à tous mes chers amis.

BENYOUCEF Rayane

REMERCIEMENTS

Louange à notre seigneur 'ALLAH' qui nous a doté de la merveilleuse faculté de raisonnement. Louange à notre créateur qui nous a incité à acquérir le savoir. C'est à lui que nous adressons toute notre gratitude en premier lieu.

Nous tenons tout d'abord à exprimer ma reconnaissance envers les membres du jury, dont la renommée et la qualité scientifique honore grandement ce travail :

– Monsieur ILLOUL Rachid professeur de l'école national polytechnique Alger pour m'avoir fait l'honneur de présider le jury de notre thèse.

–Monsieur Abdelouel Lahcen de nous avoir fait l'honneur d'être examinateur de ce travail.

Nous tenons à exprimer toute notre gratitude aux rapporteurs Madame CHAIB Khaoula et Monsieur CHAKIR Messaoud qui ont assuré la direction et l'encadrement du travail présenté dans ce mémoire.

Table des matières

Chapitre 1

Chapitre 2

Chapitre 3

Liste des Figures

Chapitre 1

Chapitre 2

Chapitre 3

Chapitre 4

Liste des tableaux

Liste des abréviations

BRIEF	**B**inary **R**obust **I**ndependent **E**lementary Feature.
BRISK	**B**inary **R**obust **I**nvariant **S**calable **K**eypoints
CCD	**C**harge **C**oupled **D**evice
CMOS	**C**omplementary **M**etal **O**xide **S**emiconductor.
DoG	**D**ifference **o**f **G**aussian.
EKF	**E**xtended **K**alman **F**ilter.
FAST	**F**reatures from **A**ccelerated **S**egment **T**est.
FPGA	**F**ield **P**rogrammable **G**ate **A**rray.
G2O	**G**eneral Framework for **G**raph **O**ptimization
HoG	**H**istogram **o**f local oriented **G**radients.
ICP	**I**terative **C**losest **P**oint.
NARF	**N**ormal **A**ligned **R**adial **F**eature.
ORB	**O**riented FAST and **R**otated **B**RIEF.
PC	**P**ersonnel **C**omputer.
RANSAC	**RAN**dom **SA**mple **C**onsensus.
RGB	**R**ed, **G**reen, **B**lue.
RGB-D	**R**ed, **G**reen, **B**lue–**D**epth.
SGD	**S**tochastique **G**radient **D**escente.
SIFT	**S**calar **I**nvariant **F**eature **T**ransform.
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping.
SURF	**S**peeded **U**p **R**obust **F**eature.
TORO	**T**ree-based netw**OR**k **O**ptimizer
ROS	**R**obot **O**perating **S**ystem.
V-SLAM	**V**isual-**S**imultaneous **L**ocalization **A**nd **M**apping.
2D	**2D**imensions.
3D	**3D**imensions.

INTRODUCTION GENERALE

La thématique de la navigation autonome constitue l'un des principaux axes de recherche dans le domaine de la robotique mobile.

Les robots doivent donc d'une part être suffisamment autonomes pour remplir leur mission sans adaptation à l'environnement et sans la supervision de l'Homme et, d'autre part être capable de fournir à l'utilisateur des informations sur la topologie du lieu, le plus souvent sous la forme d'une carte.

En effet, sans informations suffisantes sur la position du robot (localisation) et sur la nature de son environnement (cartographie), les autres algorithmes (génération de trajectoire, évitement d'obstacles ...) ne peuvent pas fonctionner correctement.

Dans ces cas, les problèmes de cartographie et de localisation sont interdépendants et il faut les résoudre ensemble, ce qui a donné naissance à un domaine de recherche très actif depuis les années 1990 : celui de la cartographie et localisation simultanées, désigné par son acronyme anglais SLAM (*Simultaneous Localization and Mapping*).

Comme il sera montré, il est désormais possible de résoudre le problème du SLAM en 3D grâce à des capteurs innovants et à des algorithmes performants. Aussi, pour extraire les informations utiles à l'accomplissement de sa tâche, il est nécessaire que le robot dispose de nombreux capteurs mesurant l'environnement dans lequel il évolue. Le choix des capteurs dépend bien évidemment de l'application envisagée.

Parmi les nombreux organes sensoriels développés par la nature, la vision est celui qui apporte l'information la plus riche et la plus complète sur le milieu environnant. La vue permet en effet de percevoir simultanément l'environnement proche et lointain avec une quantité de détails inaccessible à tout autre sens. A cet effet, le calcul de l'information de la profondeur des objets dans la scène est d'une grande utilité pour qu'un robot puisse se déplacer et interagir avec son environnement, Le capteur qui a été retenu pour la résolution du problème du SLAM est : la Kinect de Microsoft. Il s'agit d'une camera RGB-D, autrement dit d'un système d'acquisition fournissant à la fois des images et des données métriques. Cette approche du SLAM basée sur la vision est apparue il y a une dizaine d'années est nommée RGB-D SLAM ou V-SLAM.

Bien évidemment, l'aspect soft apparait comme un point essentiel à traiter car Les techniques de reconstruction des scènes 3D à partir des images 2D nécessitent des algorithmes

robustes, Il faut choisir l'outil le plus adéquat et le mieux satisfaisant à nos besoins. ROS est un système d'exploitation pour un robot (open-source) Il fournit un très grand nombre d'outils et de bibliothèques facilitant les développements.

Le sujet proposé concerne la localisation du robot de service « *B21r* » et la cartographie de son environnement en simultané. Il s'agit donc de développer une application V-SLAM en milieu intérieur, sous l'environnement de programmation ROS (Robot Operating System).

Le chapitre 1 : débutera par une présentation du contexte et des motivations de cette recherche et justifiera le choix de restreindre cette étude à la vision en faisant une description du capteur considéré (Kinect) et son principe de fonctionnement, nous décrirons ensuite la plateforme mobile expérimentale sur laquelle s'est effectué notre travail « b21r » et enfin la représentation des différents types de capteurs.

Le chapitre 2 : dans ce chapitre on s'intéressera plus aux algorithmes développés au cours de ce travail de fin d'études, tout d'abord nous détaillerons les techniques d'extraction et de description des points d'intérêt. Par la suite, on passe à la description du principe de l'algorithme RANSAC qui est une méthode d'estimation adoptée pour le calcul de la transformation de mouvement et l'élimination des points aberrants. Pour affiner cette transformation on décrira par la suite le principe de l'algorithme ICP. Nous terminons ce chapitre par donner le principe de la probabilité bayésienne pour estimer la détection de fermeture de boucle et on passe enfin en détails les différentes méthodes d'optimisation des graphes.

Le chapitre 3 : Pour se focaliser sur le problème du SLAM, nous allons nous restreindre dans ce chapitre aux comparaisons entre le SLAM en 2D et le SLAM en 3D. On débutera dans ce chapitre par présenter les différentes techniques du SLAM en 2D puis on détaillera les étapes de notre approche du SLAM visuel 3D pour pouvoir établir enfin une comparaison entre les deux types.

Le chapitre 4 : Ce chapitre est consacré à la présentation et l'analyse des différents résultats obtenus durant nos expérimentations. Nous allons d'abord présenter, analyser et comparer quelques résultats de test sur les différents types de détection/extraction des points d'intérêt.

Ensuite, nous allons présenter les résultats liés aux améliorations majeures apportées par l'algorithme d'ICP pour une estimation fiable de la transformation entre les images, nous allons

également faire une comparaison entre deux approches d'optimisation du graphe et à la fin nous allons démontrer les bases théoriques évoquées dans le chapitre précédent du SLAM Visuel.

Nous clôturant ce mémoire par une conclusion générale dans laquelle on récapitule ce qui a été fait, et on mentionne des perspectives pour notre travail.

1

Généralités sur la robotique

1.1 INTRODUCTION

Les méthodes SLAM (*Simultaneous Localization And Mapping*) permettent d'estimer à la fois la trajectoire d'un Robot mobile et la structure de l'environnement à partir de données acquises depuis un capteur embarqué, il est intéressant de constater que les chercheurs en robotique ont rapidement essayé de faire bénéficier les robots de capacités de vision en supposant qu'un robot doté de vision serait plus à même d'effectuer des tâches essentielles telles que la navigation dans un environnement inconnu par exemple.

Dans ce chapitre, nous présentons le problème général du SLAM puis nous abordons des généralités sur la robotique mobile, nous parlons aussi des différentes approches du SLAM, nous exposons par la suite les objets qui constitueront les éléments nécessaires à notre travail notamment la plateforme mobile expérimentale sur laquelle s'est effectué notre travail le robot « B21r », et enfin une description générale sur le capteur Kinect.

1.2 Généralités sur la robotique mobile

Au début de 21ème siècle, la place occupée par les robots dans notre société n'a jamais été aussi importante, et elle tend à s'agrandir de plus en plus rapidement dans le futur.

Cependant, dans de nombreuses applications, il peut être utile de construire un robot qui peut fonctionner avec une plus grande mobilité. Contrairement à la plupart des robots fixes, où l'espace environnant est adapté en fonction des tâches de robot, les robots mobiles doivent adapter leur comportement à leur environnement. Au lieu d'effectuer une séquence fixe d'actions, les robots mobiles ont besoin de développer une certaine conscience de leur environnement grâce à l'interaction avec tous les types de capteurs ; ils utilisent pour déterminer la meilleure action à prendre.

Les applications des robots peuvent se trouver dans de nombreuses activités pénibles ou dangereuses, mais également pour des applications ludiques ou de service, comme l'assistance aux personnes âgées ou handicapées.

Parmi les domaines d'applications possibles de la robotique, citons :

– La robotique de service (hôpital, bureaux, maison) ;

– La robotique de loisir (jouets, robot 'compagnon') ;

– La robotique industrielle ou agricole (entrepôts logistiques, récolte de productions agricoles, mines) ;

– La robotique en environnement dangereux (spatial, industriel, militaire, catastrophes naturelles).

Figure 1.1 : quelques applications de la robotique mobile

1.2.1 Robot mobile

Un robot est une machine équipée de capacités de perception, de décision et d'action qui lui permettent d'agir de manière autonome dans son environnement en fonction des données acquises par les capteurs.

1.2.1.1 Mobilité

Les robots mobiles sont généralement les robots qui peuvent se mouvoir ou se déplacer afin d'effectuer une tache qui lui a été assignée. La mobilité donne au robot une grande flexibilité pour accomplir des tâches intéressantes et complexes. Les Robots à mobilité peuvent naviguer dans un environnement n'a pas été conçu spécialement pour eux. Ces robots peuvent travailler dans un espace humain centré et de coopérer avec les hommes en partageant un espace de travail ensemble [4].

1.2.1.2 Autonome

Un robot autonome est un système capable de traiter ses informations intelligemment en vue de prendre des décisions et savoir comment interagir avec tels ou tels objets présents dans son environnement sans la supervision de l'homme.

1.2.1.3 Holonomes et non holonome

En robotique mobile les termes omnidirectionnels, holonomes et non holonomes sont souvent utilisés. Les termes holonomes et omnidirectionnelles sont parfois confondus. Tout mouvement effectué dans un plan peut être décomposé en un maximum de deux mouvements de translation purs effectués respectivement par rapport aux axes X et Y plus un mouvement de rotation pur autour de l'axe perpendiculaire au plan (Figure 1.2).

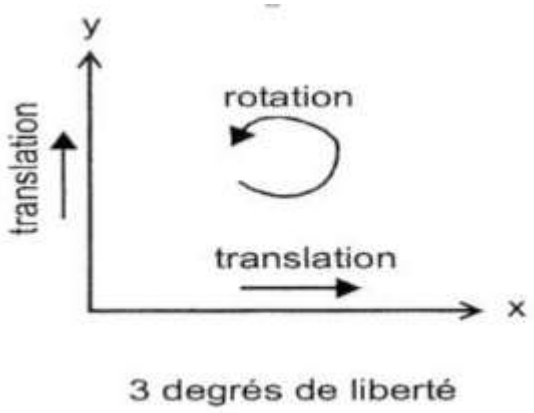


Figure 1.2: degrés de liberté

Les robots holonomes offrent une mobilité totale puisque le nombre de degrés de libertés contrôlables est égal au nombre total de degrés de liberté. Cela rend la planification de

trajectoire plus facile parce qu'il n'y a pas des contraintes qui doivent être intégrées. La mise en œuvre des comportements réactifs est facile car il n'y a pas de contraintes qui limitent les directions dans lesquelles le robot peut accélérer.

Par contre les plates-formes omnidirectionnelles permettent de découpler de manière plus nette le contrôle de la rotation et de la translation d'un robot et sont donc quasiment holonomes

Dans ce contexte, un robot mobile holonome a les propriétés suivantes :

- La configuration du robot est décrite par trois coordonnées. La géométrie interne ne figure pas dans les équations cinématiques du robot mobile, il peut donc être ignoré.
- Le robot a trois degré de liberté sans singularités.
- Le robot peut instantanément se déplacer et accélérer dans une combinaison arbitraire de directions x, y, θ.

Les robots non holonomes sont les plus répandus en raison de leur conception simple et la facilité de contrôle. Par leur nature, les robots mobiles non holonomes ont moins de degrés de liberté que les robots mobiles holonomes. Ces quelques degrés de liberté actionnés dans les robots mobiles non holonomes sont souvent contrôlable indépendamment ou découplé mécaniquement, ce qui simplifie encore le contrôle de bas niveau du robot. Comme ils ont moins de degrés de liberté, il y a certains mouvements qu'ils ne peuvent pas effectuer. Cela crée des problèmes difficiles pour la planification et la mise en œuvre des comportements réactifs.

Dans ce contexte, un robot mobile non holonome a les propriétés suivantes :

- La configuration du robot est décrite par plus de trois coordonnées. Trois valeurs sont nécessaires pour décrire la position et l'orientation du robot, tandis que d'autres sont nécessaires pour décrire la géométrie interne.
- Le robot a deux degré de liberté, ou trois degré de liberté avec singularités. (Un degré de liberté est cinématiquement possible, mais est-il un robot ?).

1.2.4 La vision en robotique mobile

L'objectif principal est la conception d'un système permettant d'adapter les capacités du robot en fonction du contexte visuel. La plupart des applications de vision pour la robotique nécessitent tout d'abord d'extraire certaines informations pertinentes de l'image, puis utilisent ces informations pour une tâche donnée (contrôle, cartographie, etc.).

Localisation et cartographie basée sur la vision a pour but d'effectuer simultanément la construction d'une carte de l'environnement dans lequel se déplace la caméra et la localisation de la caméra dans celle-ci. Elle s'inspire d'un côté des travaux de localisation et cartographie simultanée (Simultaneous Localization And Mapping ou SLAM) qui se basent généralement sur des capteurs de distance D'un autre côté, cette approche reprend un certain nombre de concepts développés dans les travaux de reconstruction 3D à partir du mouvement en leur ajoutant la notion de traitement en temps réel. Cette approche de SLAM basée sur la vision est apparue il y a une dizaine d'années avec les travaux d'Andrew Davison principalement [5]. Elle est très utilisée aujourd'hui car elle représente un moyen pratique et efficace d'effectuer de la cartographie et de la localisation en utilisant uniquement la vision. [6]

1.2.5 Présentation du Robot mobile B21r

Le robot mobile B21r est une plate-forme expérimentale construite par la société iRobot (Fig.1.3) et mise à notre disposition par le CDTA (Centre de Développement et Technologies Avancées). Ce robot est destiné à se déplacer dans un milieu d'intérieur, il a une forme cylindrique pouvant se déplacer grâce à quatre moteurs indépendants (trois pour l'entrainement et l'autre pour l'orientation). Celui destiné à la rotation peut changer l'orientation des quatre roues du robot avec une vitesse entre 1deg/sec et 90deg/sec et le moteur de translation peut faire tourner les roues avec une vitesse variant entre 1cm/s et 90cm/s. 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 son paroi.

Le B21r est dit non holonome car il dispose de 2 degrés de liberté sur un plan et puisque les translations latérales sont impossibles à réaliser et sachant que les vitesses peuvent être négatives ; les mouvements possibles sont : la translation (avance ou recule) et la rotation (tourne vers la droite ou vers la gauche).

Figure 1.3 : le robot mobile B21r

1.2.5.1 Caractéristique du « B21r »

Le robot sur lequel a été faite l'implémentation de notre projet est le B21r (Figure 1.3) qui a les principales caractéristiques techniques suivantes :

- Diamètre : 52cm (forme cylindrique),
- Hauteur : 106cm,
- Poids total : 122.5 kg,
- Charge utile : 90 kg,
- Vitesse de déplacement : 90 cm/s,
- Alimentation : 28.5 Volts,
- Batteries : 4 batteries de 31 AMP/h de 12Volts,
- Motorisation : 4 moteurs (1 pour la direction des roues) 24V,
- Vitesse de rotation : 167deg/s.

1.2.5.2 Partie matérielle du « B21r »

La plateforme est composée principalement de trois sections (Figure 1.4) : la base, l'enceinte et la console, cette dernière est physiquement attachée à l'enceinte qui elle, peut faire des rotations libres indépendamment de la base. La base du robot comporte la partie de bas niveau allant des roues jusqu'aux moteurs et leurs électroniques de commande en passant par les

batteries et les systèmes d'entrainement mécanique, la partie externe de la base comporte des capteurs à ultrasons [7]. L'enceinte contient l'ordinateur principal et au centre un scan laser de type « Hokuyo », à l'extérieur sont disposées deux ceintures de capteur à ultrasons et infrarouges, il dispose aussi d'un système d'odométrie fournissant la position du robot en coordonnées cartésiennes. La console supporte une caméra de type RGB-D «Kinect», et une interface IHM « RFLEX ».

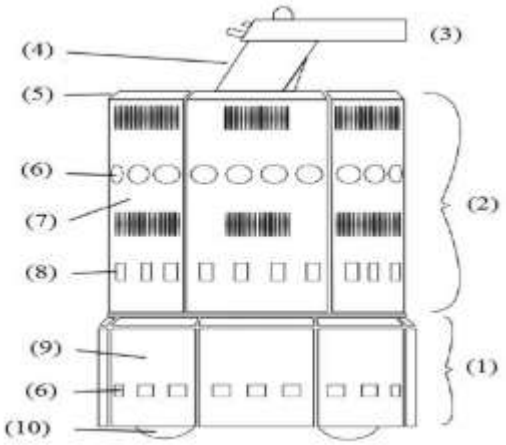


Figure 1.4 : composants du B21r

(1) : la base

(2) : l'enceinte

(3) : la console

(4) : entretoise de la console

(5) : arc de la porte

(6) : capteurs à ultrasons

(7) : porte de l'enceinte

(8) : capteurs à infrarouges

(9) : porte de la base

(10) : roues

Le B21r dispose de quatre roues motrices qui restent toujours parallèles (Figure 1.5) et quatre moteurs, trois moteurs pour l'entraînement et un moteur pour la direction des roues, ce dispositif de séparation entre l'entraînement et la direction permet une plus grande précision de l'orientation [7].

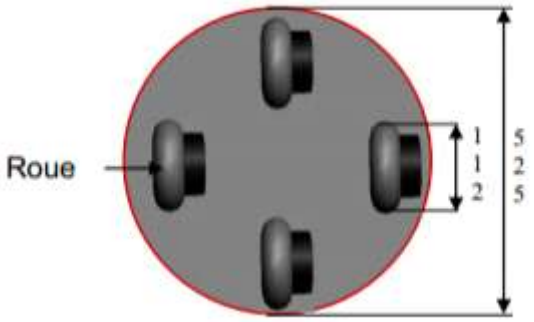


Figure 1.5 : disposition des roues du B21r

1.2.5.3 Partie logicielle du B21r

On communique avec le robot à travers un PC embarqué et la carte Rflex pour accéder au bas niveau.

L'architecture hardware de notre système robotique comporte essentiellement 4 dispositifs à savoir : les cartes de commandes des moteurs, la carte rFlex, le PC externe et la Kinect.

Le système rFlex

Le système rFlex est un système de contrôle conçu par la société iROBOT, il est constitué d'une carte de commande et d'un écran. La carte de commande rFlex est considérée comme l'élément central du robot car elle est connectée aux différents capteurs et moteurs. Cette carte est dotée de programmes ayant pour charge la récupération des données capteurs et la transmission des consignes aux moteurs. Quant à l'écran, il présente une interface offrant à l'utilisateur la possibilité de diriger le robot au moyen d'un menu facile à manipuler. Cet écran affiche également l'état du robot (batteries, luminosité,..).

Ce système est accessible via le PC externe car il est équipé d'un port de connexion série RS232.

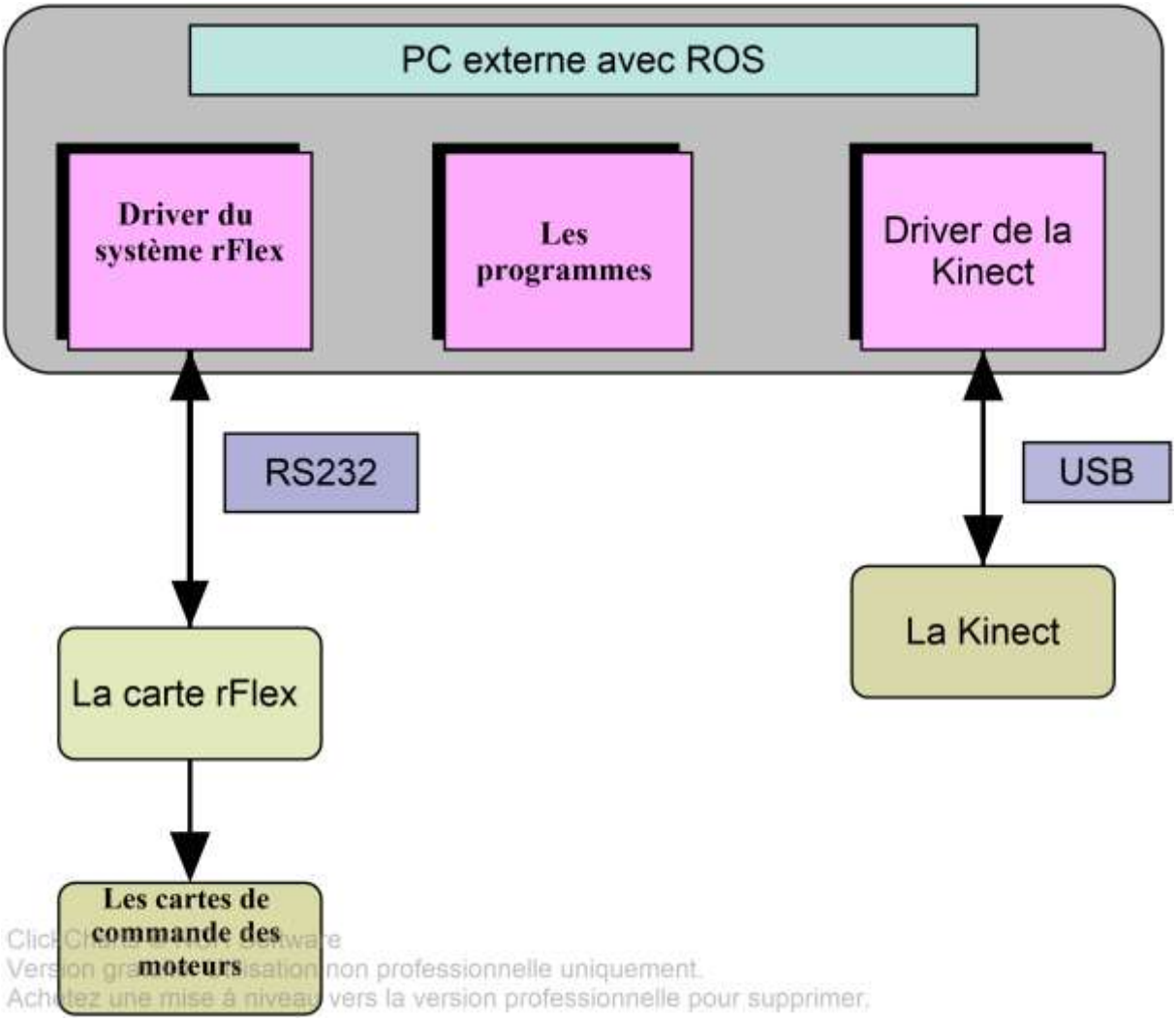


Figure 1.6 : Organigramme de l'architecture globale du robot

1.3 SLAM

L'un des principaux objectifs de la robotique est de développer des robots qui disposent d'une autonomie presque totale à long terme dans des environnements non structurés et inconnus. Une telle autonomie ne sera atteinte grâce à des algorithmes qui permettent aux robots de percevoir, d'interpréter et d'interagir avec le monde qui l'entoure.

Le problème de localisation et cartographie simultanées (SLAM) traite deux questions importantes dans la robotique mobile. La première question est : « Où suis-je ? ». La réponse à cette question définit la localisation du robot. La deuxième question concerne les caractéristiques de l'environnement du robot : « À quoi ressemble l'environnement où je me trouve ? »[1].

1.3.1 Formulation du problème de SLAM

Le SLAM est composé d'un ensemble de méthodes permettant à un robot de construire une carte d'un environnement et en même temps de se localiser en utilisant cette carte. La trajectoire du robot et la position des amers[1] dans la carte sont estimées au fur et à mesure, sans avoir besoin de connaissances a priori.

Considérons un robot se déplaçant dans un environnement inconnu, en observant un certain nombre d'amers grâce à un capteur embarqué sur le robot. La figure 1.7 montre une illustration du problème.

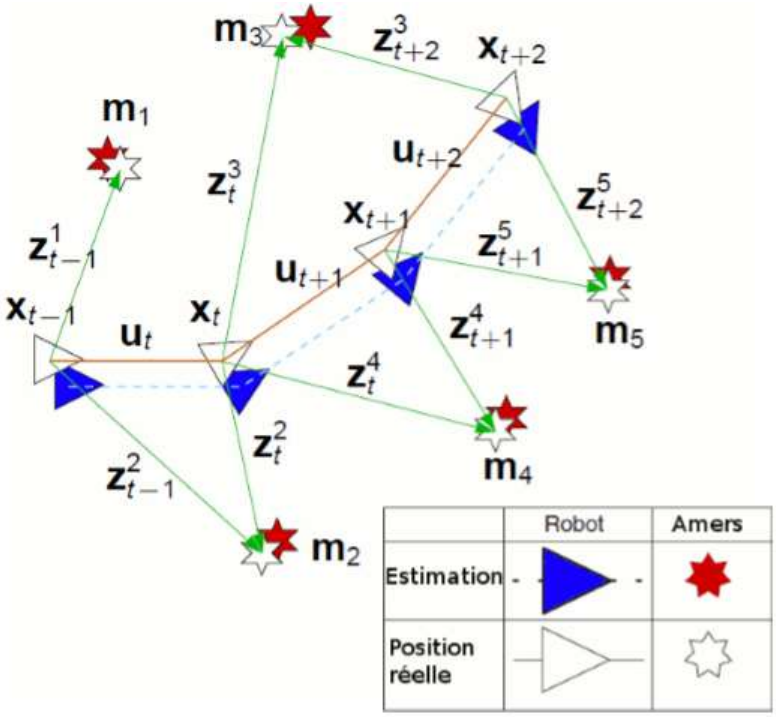


Figure 1.7 : l'idée de base du SLAM

A l'instant k on définit les quantités suivantes :

- x_k : le vecteur d'état. Il contient la position du robot.
- u_k: le vecteur de contrôle. L'application de u_k à l'instant $k-1$ mène le robot de l'état x_{k-1} à l'état x_k.
- m_i: vecteur contenant la position de l'amer i.
- z_k : l'observation à l'instant k.

1 : qui sont les points d'intérêt.

On définit aussi les ensembles suivants :

- $x_{0:k} = \{x_0, x_1, ..., x_k\} = \{x_{0:k-1}, x_k\}$: l'ensemble des vecteurs d'état jusqu'à l'instant k.
- $u_{0:k} = \{u_0, u_1, ..., u_k\} = \{u_{0:k-1}, u_k\}$: l'ensemble des vecteurs de commande jusqu'à l'instant k.
- $z_{0:k} = \{z_0, z_1, ..., z_k\} = \{z_{0:k-1}, z_k\}$: l'ensemble des observations jusqu'à l'instant k.
- $m = \{m_1, m_2, ..., m_n\}$: la carte de l'environnement contenant une liste d'objets statiques.

1.3.1.1 La localisation

Le problème de localisation du robot consiste à estimer sa position dans un environnement donné, en utilisant l'historique de ses observations, l'historique des commandes et la connaissance de l'environnement. La figure 1.8 schématise ce principe.

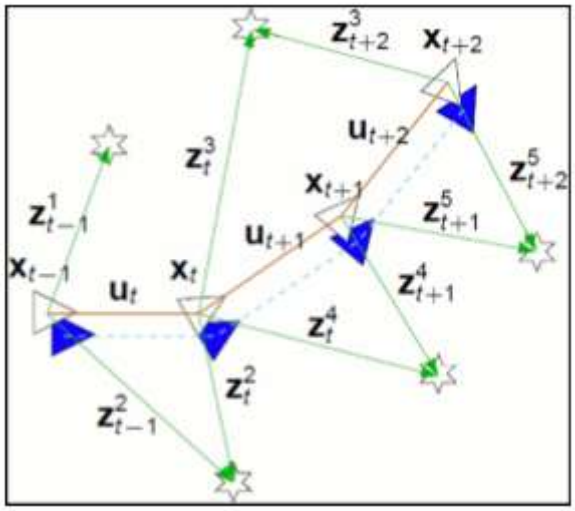


Figure 1.8 : La localisation : le système cherche à estimer sa position en utilisant les informations sur l'environnement dont il dispose

On peut analytiquement représenter cette opération par l'estimation de la probabilité de distribution :

$$P(x_k | Z_{0:k}, U_{0:k}, m) \qquad (1.1)$$

L'estimation d'une telle quantité définit la localisation globale, dans la mesure où on utilise toutes les données de l'historique des observations et des commandes pour estimer la position. On obtient ainsi une estimation robuste de la position a postériori, mais on augmente largement la complexité des calculs.

Afin de simplifier l'algorithme, on peut définir une localisation locale, où on utilise uniquement les données de l'instant (k−1) pour estimer la position à l'instant k. On représente analytiquement cette opération par l'estimation de la distribution de probabilité :

$$P(x_k|z_{k-1}, u_{k-1}, x_{k-1}, m) \qquad (1.2)$$

En utilisant cette méthode, on simplifie largement la complexité de l'algorithme, mais on risque de dévier de la position correcte du robot, sans pouvoir corriger cela.

1.3.1.2 La cartographie

Le problème de cartographie consiste à déterminer la carte d'un environnement, en utilisant les données des capteurs et l'historique des positions réelles du robot. Sur le schéma de la figure 1.9, le système connait sa position exacte et estime la carte de l'environnement en utilisant les données de ses capteurs.

On peut exprimer cela analytiquement ainsi :

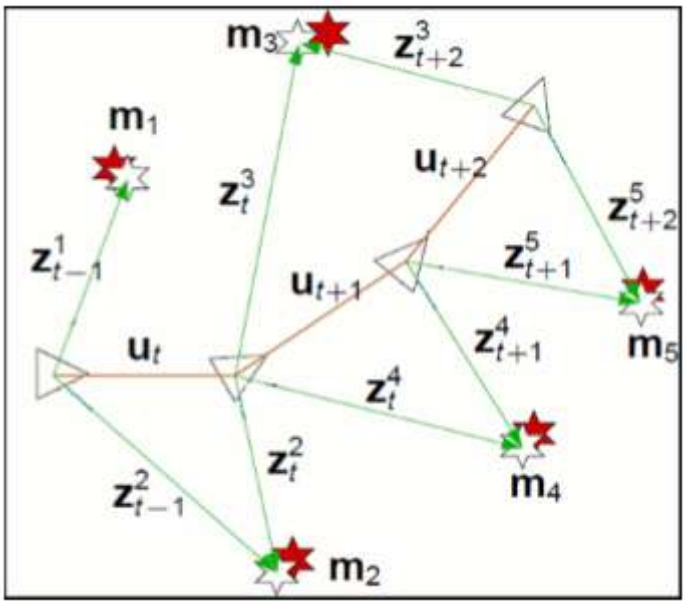


Figure 1.9 : La cartographie : le système crée la carte de l'environnement en se basant sur sa position connue et les informations de ses capteurs.

1.3.2 Les différentes approches de SLAM

Nous allons présenter dans cette partie trois méthodes largement utilisées pour la résolution du problème de SLAM. Chaque méthode a des avantages et des inconvénients.

1.3.2.1 EKF-SLAM

Depuis les années 1980, plusieurs méthodes de traiter le problème du SLAM ont vu le jour. Sans doute la plus utilisée, reste le SLAM par filtre de Kalman étendu (*EKF : Extended Kalman Filter*) [2].

Elle utilise un vecteur d'état pour représenter la position du robot et des amers dans la scène, auquel est associe une matrice d'erreur représentant les incertitudes sur les positions, les observations et les corrélations entre les différentes variables du vecteur d'état. Cette matrice est mise à jour régulièrement à chaque fois que le robot change de position [1] par conséquence sa taille grandit quadratiquement en O (n2) (où n est le nombre d'amers de la carte)

Les inconvénients majeurs de cet algorithme c'est qu'elle est complexe, se base uniquement sur la probabilité gaussienne, En plus il atteindra un point où il ne pourra pas mettre à jour sa carte en temps réel et il n'est pas toujours facile d'extraire des amers correctes et intéressants dans un environnement non-structuré ou externe.

1.3.2.2 SLAM basé sur les filtres particulaires

Un filtre particulaire est un filtre récursif qui permet d'estimer l'état a postériori en utilisant un ensemble de particules. Le principe est de suivre un grand nombre d'hypothèses en parallèle qui sont autant de trajectoires possibles. Ces différentes hypothèses correspondent à un échantillonnage de la distribution de probabilité des trajectoires.

L'utilisation du filtrage particulaire souffre de plusieurs problèmes à cause de sa complexité grandissante qui augmente exponentiellement avec le nombre d'amers de l'environnement et des difficultés rencontrées lors de la définition du nombre de particules.

1.3.2.3 Graph-based SLAM

Le *Graph-based SLAM* reste une solution intéressante grâce à sa simplicité et son efficacité en temps de calcul, L'idée principale est de construire un graph dans lequel les poses du robot sont modélisées par des nœuds qui sont reliés par des contraintes. Après la construction le graph doit être optimisé, pour se faire il existe des approches diverses comme HOG-Man, G2O, TORO **…..**[3].

Le gros avantage du Graph-SLAM est qu'il permet de gérer les cartes composées d'un très grand nombre de nœuds ce qui est impossible avec les autres techniques. Cependant l'optimisation du graphe peut être très lourde.

1.4 Les capteurs utilisés pour résoudre le problème du SLAM

Pour résoudre le problème du SLAM, le robot a besoin de différents types de capteurs capables de le renseigner sur l'état du milieu où il opère, mais aussi sur son état, le choix de ces capteurs dépend de la tâche réalisée et le type d'environnement. En robotique mobile, on classe traditionnellement les capteurs en deux catégories selon qu'ils mesurent l'état du robot lui-même ou l'état de son environnement. Dans le premier cas, à l'image de la perception chez les êtres vivants, on parle de proprioception et donc de capteurs proprioceptifs. On trouve par exemple dans cette catégorie les capteurs de position ou de vitesse des roues et les capteurs de charge de la batterie. Les capteurs renseignant sur l'état de l'environnement, donc de ce qui est extérieur au robot lui-même, sont eux appelés capteurs extéroceptifs. Il s'agit de capteurs donnant la distance du robot à l'environnement, la température, signalant la mise en contact du robot avec l'environnement, etc.

Dans cette section nous représentons les capteurs dont le robot B21r est équipé.

1.4.1 Odomètre

L'odométrie permet d'estimer le déplacement de la plateforme à partir de la mesure de rotation des roues. La mesure de rotation est en général effectuée par un codeur optique disposé sur l'axe de la roue, ou sur le système de transmission. Le problème majeur de cette mesure est que l'estimation du déplacement fournie dépend très fortement de la qualité du contact entre la roue et le sol. Pour limiter ce problème, il peut être intéressant de positionner le codeur optique sur une roue non motrice qui glissera moins.
L'idée pour que l'encodeur fonctionne est de placer un disque alternant des zones transparentes et opaques devant un capteur de lumière et de rentre le disque solidaire de l'axe de rotation de la roue. La fréquence d'apparition des zones blanches et noires (ou de tout autre principe offrant un contraste suffisant) devant le capteur de lumière va indiquer la vitesse de rotation. Le schéma suivant présente le principe de fonctionnement basique de l'encodeur :

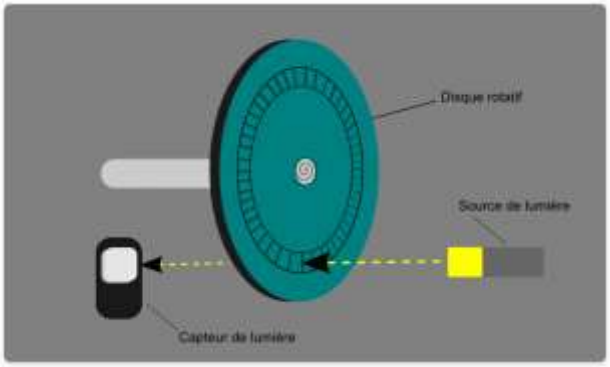


Figure 1.10 : principe de fonctionnement d'encodeur

Lorsque le disque tourne, les segments opaques bloquent la lumière alors que les segments transparents la laissent passer. Ceci génère des impulsions d'onde carrée qui peuvent ensuite être interprétées comme position ou mouvement.

1.4.2 Kinect

Kinect est apparu le 4 Novembre 2010, comme accessoire pour la console Xbox 360. Il est un dispositif mis au point par la société PrimeSense en collaboration avec Microsoft. En Janvier 2012, plus de 18 millions d'unités ont été vendues. En Février 2012, une version pour Windows a été libérée.

Le capteur Kinect est une barre horizontale reliée à une petite base avec un pivot motorisé, conçu pour être placé au-dessus ou en dessous de l'affichage vidéo (téléviseur, écran d'un ordinateur). Le dispositif comporte une caméra RGB, un capteur de profondeur constitué par un projecteur laser et une caméra infrarouge (IR) et un microphone (voir l'annexe A) ; Comme le montre la figure suivante :

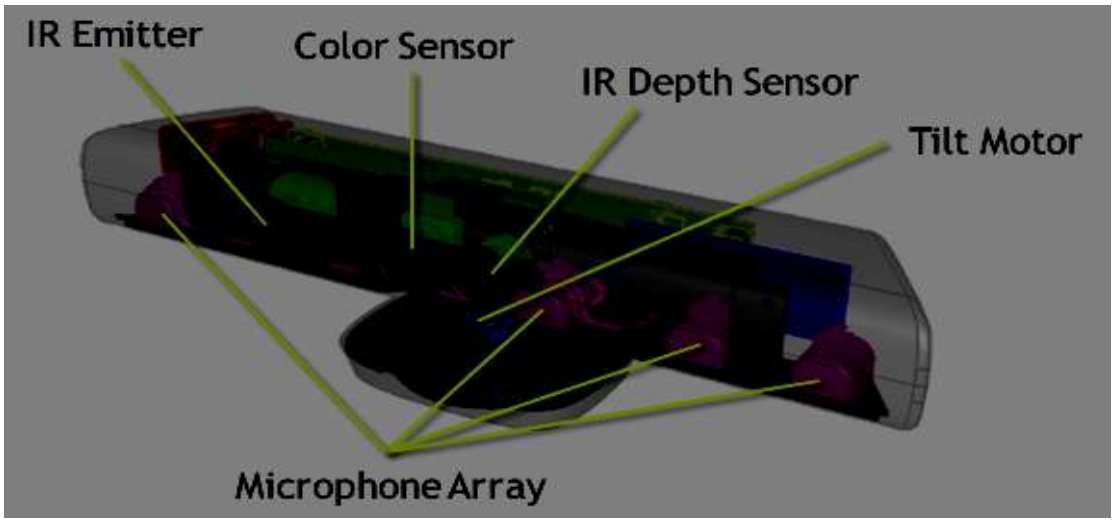


Figure 1.11 : Face avant de la Kinect : caméra émettant une mire de points dans l'infrarouge, caméra infrarouge, caméra couleur et réseau de microphones

1.4.2.1 Principe de fonctionnement de la Kinect

La camera RGB fournit des images qui seront traitées avec les outils usuels en vision par ordinateur. Le couple camera IR/Laser sert à l'estimation de profondeur. Le laser projette dans la scène des patterns qui ne sont visibles que par la camera IR. L'image IR permet alors, connaissant la distance de la base entre le laser et la camera, d'estimer la distance d'un point de l'espace par triangulation. La résolution de la camera IR permet d'obtenir un nuage de point de taille 640x480, qui correspond à la résolution de l'image. Il est alors possible d'attribuer une couleur précise à chaque point en faisant correspondre les points du nuage issu des mesures 3D avec l'image RGB.

1.4.3 Laser Hokuyo

Le capteur Laser Smart-URG UST-20LX permet d'atteindre une portée maximale de 20 mètres dans un arc de 270° avec une résolution angulaire de 0.25°. Le nombre de points de balayage (un scan laser) peut atteindre 1081 points (voir figure 1.12).

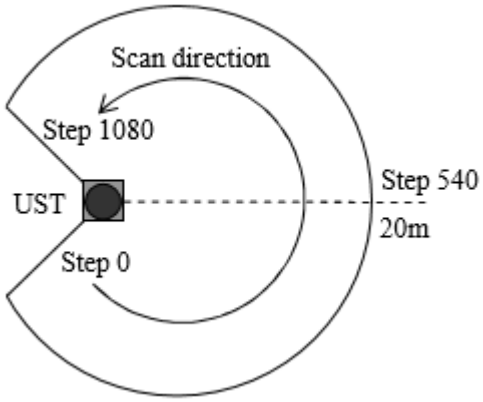


Figure 1.12 : le champ de vision du laser Hokuyo

1.4.3.1 Mode de fonctionnement

Une projection (un tir) est émise par une diode laser, en même temps une horloge est démarrée. Cette impulsion lumineuse va rebondir sur le premier obstacle rencontré sur son chemin. L'impulsion lumineuse, renvoyée par un obstacle, arrive sur un récepteur qui déclenche l'arrêt de l'horloge. À partir de ces informations, nous pourrons déduire la distance séparant le télémètre laser de l'obstacle et connaissant la position angulaire du laser lors de la détection d'obstacle nous pourrons déterminer la position de l'objet dans un plan de paramètres polaires [11].

Figure 1.13 : laser hokoyu Smart-URG

1.5 Conclusion

Ce chapitre est une description générale à l'étude menée dans le cadre de ce projet. Tout d'abord, Nous avons décrit brièvement la problématique traitée, nous avons également comparé les différentes approches de SLAM et donné des généralités sur la robotique mobile, Ensuite nous avons présenté le matériel utilisé pour aboutir à notre but et à la fin de ce chapitre nous avons parlé des différents capteurs.

2

Les algorithmes du V-SLAM

2.1 INTRODUCTION

Pour naviguer dans des environnements inconnus, le robot mobile doit modéliser son environnement sous forme d'une carte et estimer sa propre position. La résolution de ces deux problèmes en même temps déclenche de nombreuses exigences pour les algorithmes de traitement de l'information. D'abord le chapitre commencera par une approche globale de V-SLAM puis on présentera les différentes techniques pour détecter les points d'intérêt et calculer leurs descripteurs, on passera à la présentation des algorithmes d'estimation de transformation. Ensuite on détaillera la probabilité bayésienne pour la détection de fermeture de boucle et enfin, on décrira les deux approches utilisées dans ce projet pour l'optimisation du graphe.

2.2 Un aperçu général de V-SLAM

La Kinect qui est un capteur embarqué sur le Robot fournit à la fois des images couleurs et des images de profondeurs, La détection des points d'intérêts qui sont des points présentant des propriétés locales remarquables se fait à partir des images couleurs grâce à des algorithmes performants comme SIFT, SURF et ORB. Une fois les points d'intérêts sont détectés, un premier alignement est appliqué à l'aide de la bibliothèque FLANN ensuite grâce aux données de profondeur fournis par la camera IR ces points sont projetés dans l'espace en formant un nuage de points en 3D.Pour estimer la pose du robot, les nuages extraits de chaque deux images successives sont mis en correspondance en ayant recours à l'algorithme de RANSAC pour avoir une estimation initiale de la transformation et éliminer les mauvaises associations. Puis cette transformation est affinée en appliquant l'algorithme d'ICP, cette procédure connu sous le nom *Front end,* Par la suite une construction du graphe se fait en modélisant les poses du robot et les amères observés par des nœuds reliés par des contraintes, Dans le cas où une fermeture est détectée (lorsque le robot passe par un endroit déjà exploré), les 2 terminaux de boucle sont alignés et les points redondants sont fusionnés. Cela nous permet d'estimer l'erreur accumulée et de corriger l'erreur, enfin une méthode d'optimisation du graphe est faite pour pouvoir construire une carte plus précise de l'environnement parcouru.

2.3 Problématique de la recherche des points d'intérêt

La problématique de cette partie est liée à la recherche de similarités entre les images. Pour trouver des similarités, il n'est pas efficace de comparer directement les valeurs des pixels entre des images. Cette méthode ne serait pas robuste à des déformations comme celles liées à des changements d'illumination, à l'échelle, aux rotations, aux changements de points de vue. Il est toutefois possible d'essayer de trouver des corrélations entre des fenêtres de pixels. Les méthodes récentes, plus efficaces, consistent à d'écrire chaque image avec un ou plusieurs descripteurs. L'objectif est alors de construire un descripteur à partir des valeurs des pixels, qui sera invariant selon les critères souhaités (rotation, échelle...).

La qualité d'un détecteur de points d'intérêt se mesure par sa répétabilité. C'est-à-dire que l'on doit idéalement retrouver les mêmes points d'intérêt après que l'image ait subi une modification. Il doit aussi répondre à plusieurs critères importants. Parmi ces critères, la robustesse par rapport aux changements de luminosité, d'orientation (illumination, angle de vue, échelle, rotation...) et aussi aux petites variations de la position du point d'intérêt.

Il existe de très nombreux types de détecteurs et/ou descripteurs. Parmi lesquels : SIFT [12], SURF [16], ORB [13], NARF [17], BRIEF, BRISK [10], FAST [15] et Harris Corner [14].

Fournir, dans le cadre de cet état de l'art, une approche théorique rigoureuse est impossible vu que le thème de la détection de points d'intérêt ou l'estimation de la transformation étant extrêmement vastes.

L'algorithme	Détecteur	Descripteur
Harris Corner	Oui	Non
SIFT	Oui	Oui
SURF	Oui	Oui
ORB	Oui	Oui
NARF	Oui	Oui
BRISK	Oui	Oui
BRIEF	Non	Oui
FAST	Oui	Non

Tableau 2.1 : les différents détecteurs et/ou descripteur

2.3.1 SURF

Un peu plus récemment (2006 et 2008), une variante appelé SURF [12] est apparue. Le principal avantage de cette technique est que l'extraction des points d'intérêts se fait plus rapidement, notamment grâce à l'utilisation d'images intégrales.

b. Descripteur

SURF est fondé sur l'approximation de la matrice hessienne afin de détecter les structures de types « *blobs*» et utilise efficacement les images intégrales qui permettent de réduire le nombre de calculs.

Le détecteur localise les blobs là où le déterminant de la matrice hessienne en un point p = (x, y) et à l'échelle σ hessienne qui est définie comme suit atteint un maximum.

$$Det\big(H(x,y,\sigma)\big) = Det\begin{pmatrix} L_{xx}(x,y,\sigma) & L_{xy}(x,y,\sigma) \\ L_{yx}(x,y,\sigma) & L_{yy}(x,y,\sigma) \end{pmatrix} \qquad (2.1)$$

Avec $L_{xx}(x,y,\sigma)$ le résultat de la convolution de la dérivée seconde de la gaussienne $\frac{\partial^2}{\partial x^2}\mathrm{g}(\sigma)$ avec l'image au point p.

On construit une approximation de type «*box*» des dérivées secondes de la gaussienne. Pour pouvoir tirer parti des images intégrales. L'image suivante représente les dérivées partielles de la gaussienne. D'abord finies et discrétisées (les deux images de gauche) et puis approximées par un *box filter* suivant les directions y et x.

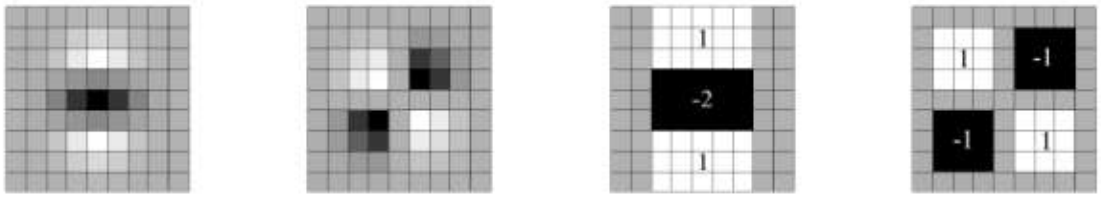


Figure 2.1 : Dérivées partielles secondes de la gaussienne

L'approximation du déterminant de la matrice hessienne calculée en un point x de l'image est stockée dans une «*blob response map*», puis les maxima locaux sont recherchés, afin de trouver des blobs.

Par rapport à ce qui a été fait pour SIFT pour avoir l'invariance aux changements d'échelle, SURF peut procéder différemment grâce aux *box filters* et aux images intégrales qui permettent de réduire le temps de calcul.

Au lieu d'appliquer successivement le même filtre à une image en agrandissant à chaque fois le facteur d'échelle et en divisant la résolution de l'image par deux (sous-échantillonnage) dans chaque octave, on peut utiliser des *box filters* de diverses tailles directement sur l'image d'origine. Les «*blob response maps*» à différentes échelles sont donc construites en agrandissant le filtre plutôt qu'en réduisant itérativement la taille de l'image.

La figure suivante montre sur L'image de gauche la méthode classique avec sous-échantillonnage et filtre de taille constante et sur l'image de droite Les filtres sont de tailles variables.

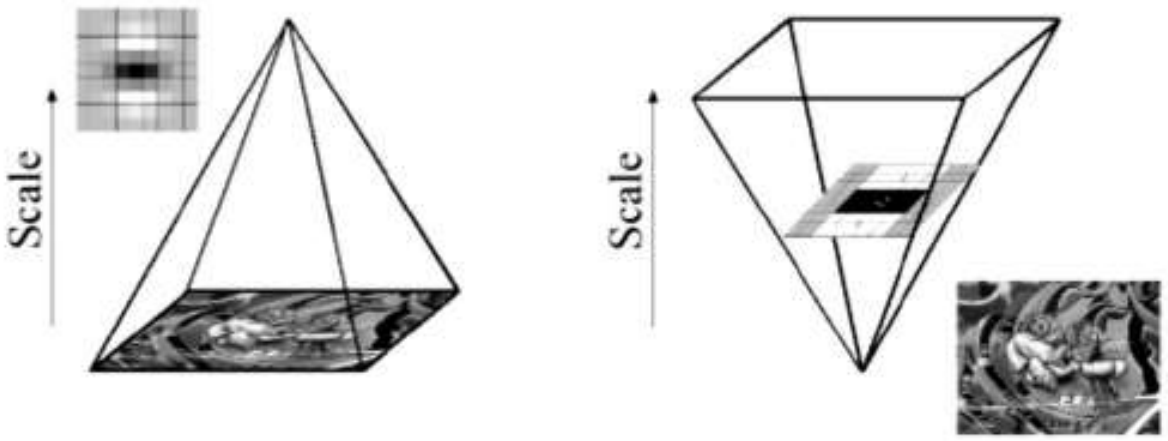


Figure 2.2 : L'espace d'échelles pour le descripteur SIFT et SURF

Les filtres représentés sur la figure 2.3 ont une taille de 9x9. La taille des filtres est progressivement augmentée. Typiquement, on utilisera les filtres 9x9, 15x15, 21x21, 27x27, 39x39, 51x51, 75x75 et 99x99. On pourrait utiliser des filtres encore plus grands mais, en pratique, le nombre de points détectés décroit rapidement avec la taille du filtre.

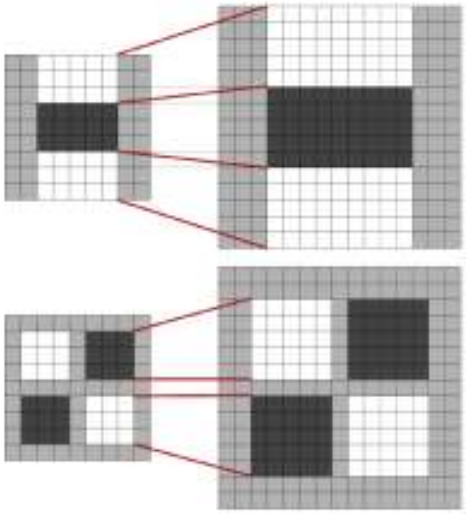


Figure 2.3 : Passage du filtre 9*9 au filtre 15*15

b. Descripteur

Pour rendre le descripteur invariant à la rotation, il faut identifier de manière reproductible une direction principale dans la zone du point d'intérêt. Pour cela, on calcule la réponse à des ondelettes de *Haar* suivant les directions x et y dans le voisinage du point d'intérêt. Les ondelettes de *Haar* étant en fait des *box filters*, on peut de nouveau utiliser les images intégrales pour accélérer le calcul.

Figure 2.4 : Les Ondelettes de *Haar* suivant les directions x et y

Une fois les réponses calculées et pondérées par une gaussienne centrée sur le point d'intérêt, la réponse est représentée par un point dans un espace dont les valeurs en abscisse sont les réponses horizontales et les valeurs en ordonnées la réponse verticale. On fait ensuite tourner une fenêtre d'angle $\pi/3$ autour du point d'intérêt puis on somme l'ensemble des réponses dans cette fenêtre pour déterminer un vecteur d'orientation locale. L'orientation principale est donnée par le vecteur le plus long.

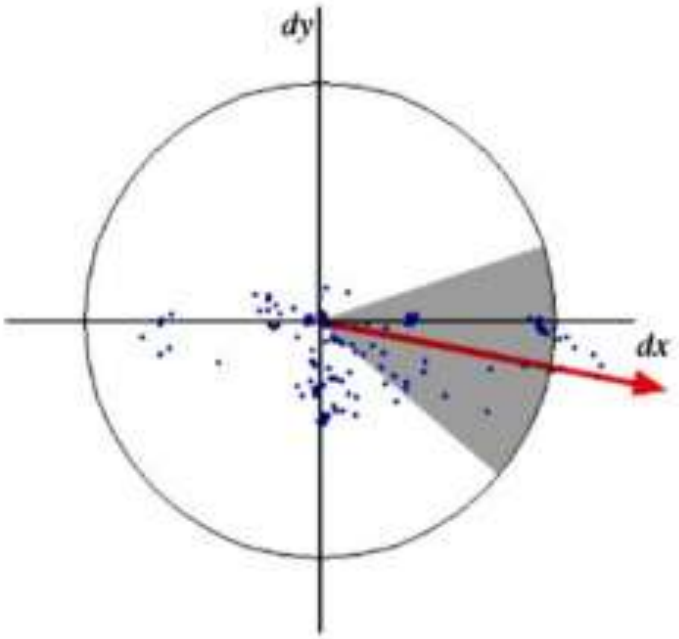


Figure 2.5 : Détection de l'orientation principale

Le calcul du descripteur consiste à construire une zone carré centrée autour du point d'intérêt et orientée suivant la direction principale sélectionnée au point précédent de taille proportionnel à l'échelle à laquelle le point d'intérêt a été trouvé.

La zone est divisée en 16 sous régions de même taille dans chacune desquelles on calcule les réponses en onde de Haar pour 4 points régulièrement espacés.

Chaque sous-région est donc décrite par un vecteur de 4 éléments (Σdx , Σdy , Σldx l, Σldyl). Comme il y a 16 sous-régions, on retrouve un vecteur de 64 dimensions, qui constitue la signature de la région d'intérêt.

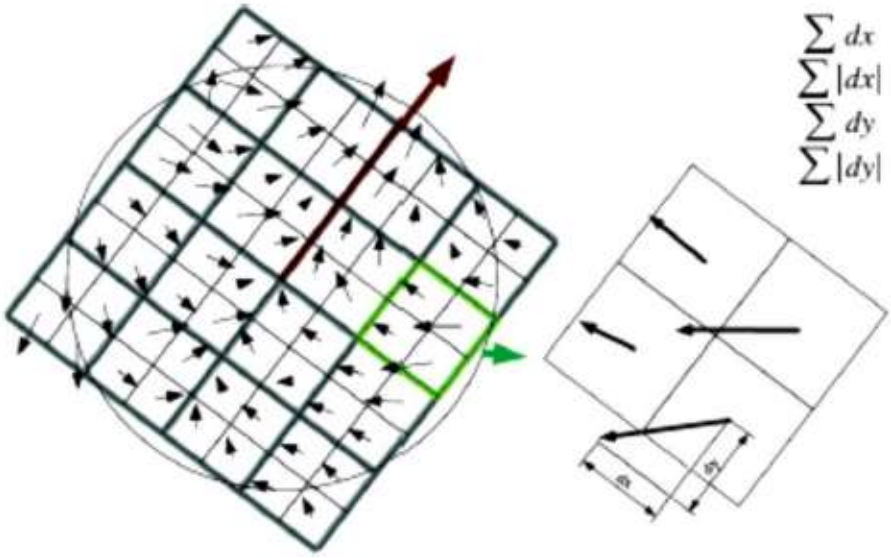


Figure 2.6 : Calcul des éléments du descripteur

2.2.2 SIFT

SIFT (The Scalar Invariant Feature Transform) est une méthode développée par David Lowe [2], très utilisé dans le domaine de la Robotique et la vision par ordinateur pour détecter et identifier les points qui ont des caractéristiques significatives similaires entre les différentes images numérique.

a. Détecteur

La première étape est de créer un espace des échelles d'une image. Cet espace est divisé en octave qui correspondent à une série de niveaux produit à partir de la convolution d'une fonction gaussienne de variance σ multiplié à chaque fois par un facteur constant G (x, y, kσ) avec une image d'entrée I (x, y), en divisant la résolution de l'image par deux dans chaque octave.

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y) \quad (2.2)$$

Où

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \quad (2.3)$$

La localisation des points-clés est obtenue en trouvant les extremums de la fonction Laplacienne des Gaussiens (LoG), Toutefois cette méthode est très gourmande en temps de calcul et comme le montre Lindeberg [12] la fonction DoG (Différence des gaussiennes) représente une bonne approximation.

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \quad (2.4)$$
$$= L(x, y, k\sigma) - L(x, y, \sigma)$$

Ceci présente l'avantage d'autoriser l'utilisation de la technique pyramidale de l'espace des échelles, la soustraction entre deux niveaux successifs dans l'espace des échelles donne la fonction DoG.

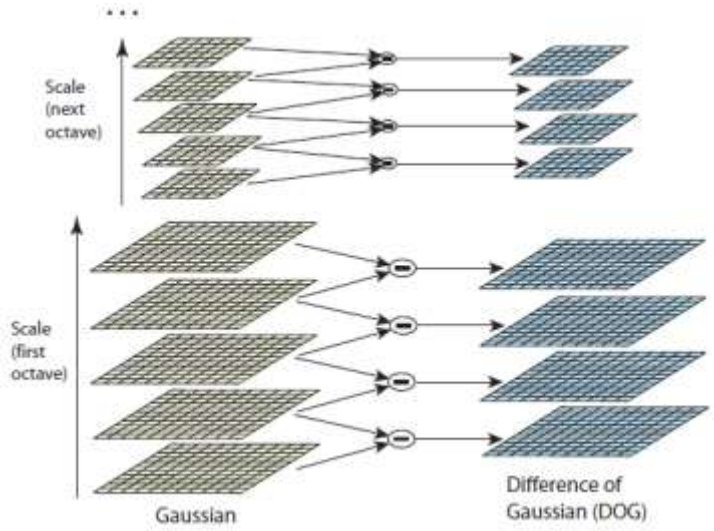


Figure 2.7 : Le calcul de la fonction DoG à partir de l'espace des échelles

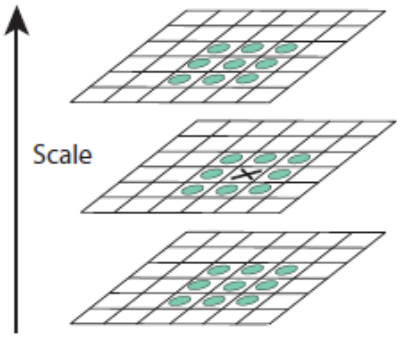


Figure 2.8 : la détection d'extremums dans l'espace des échelles

Les extremums de DoG sont détectés en comparant chaque pixel à ses huit voisins qui appartiennent à son niveau actuel et neuf voisins qui appartiennent aux niveaux supérieur et inférieur.

Les points-clés candidats produits par l'étape précédente sont nombreux. Certains sont instables, de plus leur localisation reste approximative à cause de la faible résolution dans les

octaves supérieurs. De ce fait, des traitements supplémentaires seront faits pour améliorer la localisation et éliminer les points instables.

Une fois Le filtrage effectué, une orientation déterminée localement sur l'image est calculée pour chaque point détecté à partir de la direction des gradients. Ce calcul est d'autant plus important que la direction trouvée sera utilisée pour le calcul des descripteurs et c'est elle qui garantira l'invariance des descripteurs par rotation. Le calcul s'effectue sur L(x, y, σ) en déterminant l'amplitude m(x, y) et l'orientation θ(x, y) du gradient.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (2.5)$$

$$\theta(x,y) = tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{(L(x+1,y) - L(x-1,y)}\right) \quad (2.6)$$

On calcul ensuite un histogramme de 36 intervalles des orientations dans le voisinage, Les pics de l'histogramme dont la valeur est au moins égale à 80% de la valeur maximale sont utilisés pour créer des points supplémentaires qui ne différent que par leur orientation principale.

b. Descripteur

Une fois que les points-clés associés à des facteurs d'échelles et à des orientations sont détectés et leur invariance aux changements d'échelles et aux rotations assurée on arrive à l'étape de calcul des vecteurs descripteurs, traduisant numériquement chacun de ces points-clés.

On considère une zone de 16x16 pixels autour du point d'intérêt subdivisée en 4 zones égales

Pour chacune d'elles on calcule un histogramme des orientations de 8 intervalles entre 0 et 2π.

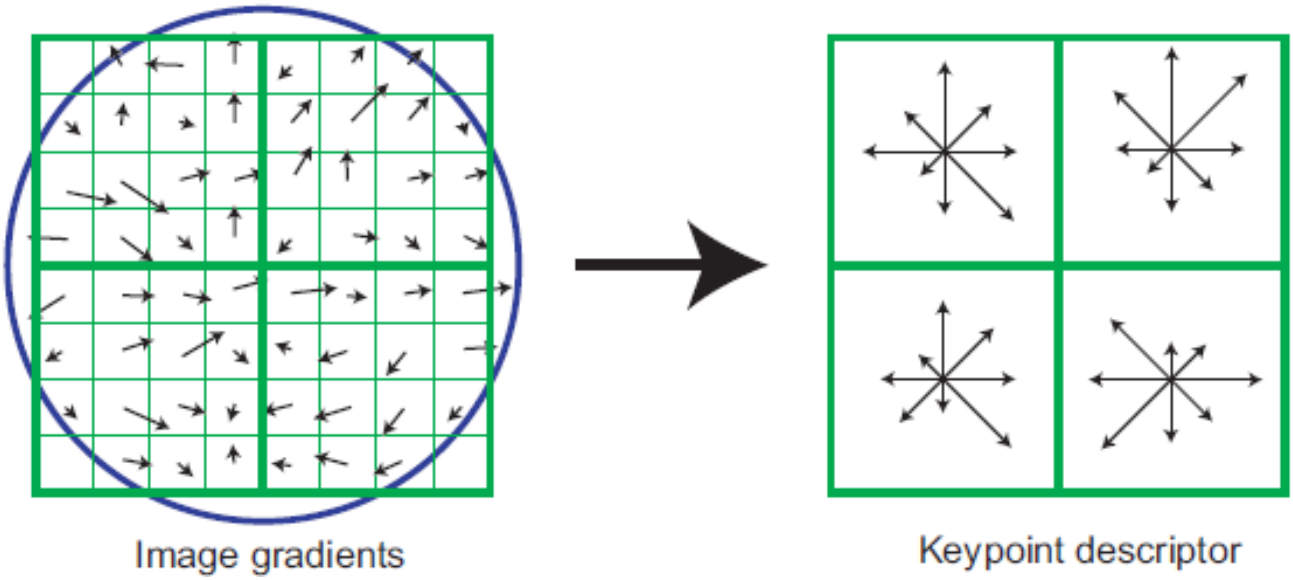


Figure 2.9 : Cette figure montre le calcul de l'histogramme d'une zone de 8x8

On obtient finalement le descripteur de ce point sous la forme d'un vecteur composé des 128 valeurs de l'histogramme. Cette étape rend les descripteurs invariants à d'autres transformations telles que la luminosité et le changement de point de vue 3D.

2.2.3 ORB

a. Détecteur ORB

Le descripteur d'ORB (*Oriented FAST and Rotated BRIEF*) [13] composé d'un détecteur FAST (*Features from Accelerated Segment Test*) et d'un descripteur BRIEF (*Binary Robust Independent Elementary Features*), nécessite des fonctionnalités d'orientation. Rublee et al [2] ont ajouté une orientation efficace calculée aux fonctionnalités FAST car ils ne fournissent pas une composante d'orientation comme SIFT et SURF.

Il s'agit d'un descripteur extrêmement simple à extraire et pourtant efficace. On définit 256 paires de points dans la zone du point d'intérêt. Ces 256 zones sont réparties en utilisant une répartition Gaussienne autour du centre de la zone. Pour chacune de ces paires de points, on compare les valeurs des pixels de chaque extrémité de la paire. Si la première valeur du pixel de la paire est plus faible que la seconde, alors on associe 1 à cette paire, 0 sinon. On construit donc un vecteur binaire de taille 256 qui correspond aux résultats concaténés des comparaisons. Ce descripteur a l'avantage d'être :

- très rapide à calculer (il s'agit uniquement de comparer des valeurs) ;
- très rapide à comparer avec la distance de Hamming ;
- facile à stocker (puisqu'il peut être représenté sur 256 bits).

Ce détecteur FAST Oriented (oFAST) utilise le barycentre d'intensité en tant que mesure d'angle d'orientation. En supposant que l'intensité d'un coin est décalée par rapport à son centre, le vecteur de barycentre d'intensité peut être utilisé pour assigner une orientation. Le barycentre est calculé par

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \quad (2.7)$$

Où

$$m_{pq} = \sum_{x,y} x^p y^p I(x,y) \quad (2.8)$$

m est le moment d'un patch. Le vecteur d'orientation est ensuite construit à partir du centre de l'angle au centre de gravité. L'orientation est alors :

$$\theta = \tan^{-1}\left(\frac{m_{01}}{m_{10}}\right) \quad (2.9)$$

Le rayon r est choisi pour être la taille de path de telle sorte que les moments sont calculés avec x et y restant dans le voisinage circulaire.

$$f_n(p) = \sum_{i=1}^{n} 2^{i-1} \tau(p; x_i, y_i) \quad (2.10)$$

Où τ est le test binaire défini comme :

$$\tau(p; x_i, y_i) := \begin{pmatrix} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{pmatrix} \quad (2.11)$$

et p (x) est l'intensité de p à un point x

Pendant la phase d'adaptation, BRIEF donne de mauvais résultats, pour une rotation dans le plan de plus de quelques degrés. Rublee et al ont ainsi ajouté une étape pour diriger le BRIEF selon l'orientation des points clés. Ici, S une matrice 2xn est définie de telle sorte que :

$$S = \left(\frac{x_1, \ldots, x_n}{y_1, \ldots, y_n}\right) \quad (2.12)$$

Pour tout ensemble de fonctionnalités de tests binaires de n à l'emplacement (x_i, y_i). S est le "dirigé" en utilisant le θ d'orientation du patch et de la matrice de rotation R_Θ

correspondant :

$$S_\Theta = R_\Theta S \tag{2.13}$$

Et ainsi l'opérateur BRIEF piloté devient

$$g_n(p,\Theta) := f_n(p) | (x_i, y_i) \in S_\Theta \tag{2.14}$$

Les caractéristiques BRIEF ont une grande variance et une moyenne proche de 0,5, ce qui les rend très discriminantes. Cependant, le pilotage de ces caractéristiques entraîne une perte significative de la variance et un changement dans la moyenne. Cela signifie que les points-clés de coin orienté présentent une apparence plus uniforme aux tests binaires. Rublee et al, ont enfin ajouté une étape d'apprentissage pour choisir un bon sous-ensemble de tests binaires pour réduire la corrélation et le nom de leurs caractéristiques améliorées est : BRIEF réorientée (BRIEF). Cette méthode recherche tous les tests binaires possibles qui ne sont pas corrélés, ont une variance élevée et de bonnes valeurs moyennes (près de 0,5).

b. Descripteur ORB

Rublee et al. [13] se servirent du détecteur FAST Oriented (oFAST), une modification du détecteur FAST bien connu [15], pour trouver des points clés dans le schéma d'ORB. Les fonctionnalités de test de segment accélérées par le détecteur (FAST) ont été développées pour des applications en temps réel et utilisées avec succès pour l'odométrie visuelle [28]. Il utilise des techniques d'apprentissage automatique pour détecter les caractéristiques d'angle à grande vitesse de calcul. Les caractéristiques sont détectées en deux phases :

Des coins FAST sont détectés en utilisant un algorithme de segment de test simple sur l'image (voir figure 2.10). Ici, un voisinage circulaire de 16 pixels autour d'un pixel central est comparé pour voir si leur intensité tombe au-dessus ou en dessous d'un seuil prédéfini. Si au moins 12 des 16 pixels passent le test, le centre est considéré comme un coin. Le détecteur FAST utilise des techniques d'apprentissage automatique pour classer les coins qui sont plus robustes et permettent un seuil plus détendu (décontracté). Rublee et al utilisent une version modifiée de FAST avec un rayon circulaire de seulement 9 pixels pour une meilleure performance.

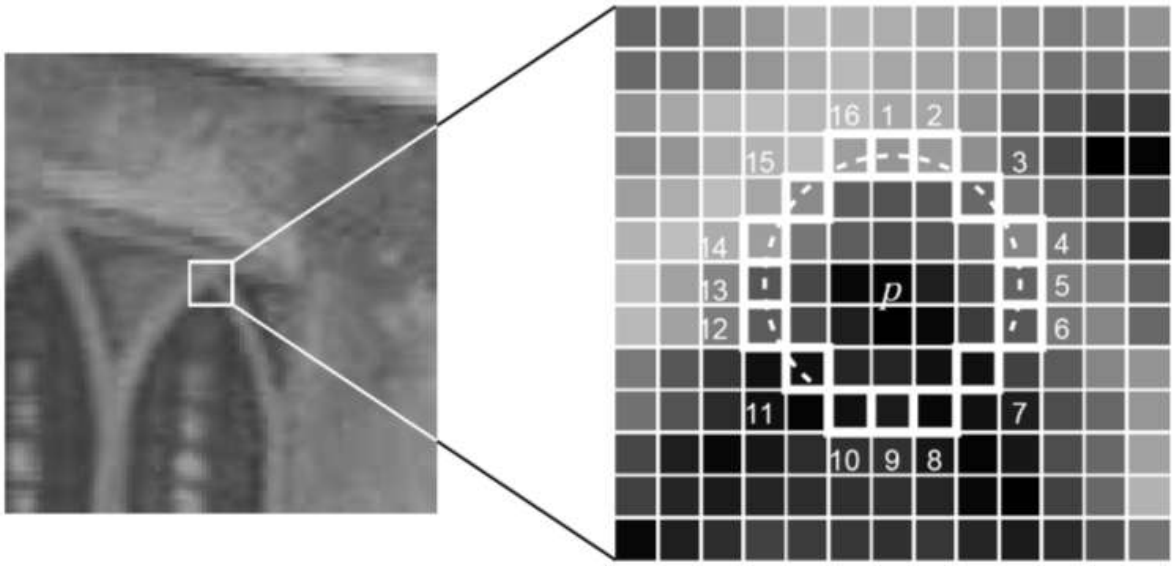


Figure 2.10 : Le détecteur Fast utilise un test de segment par lequel un voisinage circulaire de 16 pixels autour du centre pixel p est testé [29]

Le détecteur FAST ne produit pas une mesure de coins, et a été considéré pour avoir un grand nombre de réponses le long des bords. Ainsi, Rublee et al. [13] utilisent le filtre d'angle Harris [14] pour rejeter les bords et fournir un score raisonnable. Tout d'abord, un seuil est fixé suffisamment faible pour détecter plus d'un nombre de N points spécifiés. Les points clés sont ensuite commandés par le filtre Harris et les N premiers sont sélectionnés.

Le détecteur Harris peut être défini comme suit :

Étant donné

$$M(x,y,\sigma) = \det\begin{pmatrix} L_{x^2}(x,y,\sigma) & L_x(x,y,\sigma)L_y(x,y,\sigma) \\ L_x(x,y,\sigma)L_y(x,y,\sigma) & L_{y^2}(x,y,\sigma) \end{pmatrix} \quad (2.15)$$

Où L_x et L_y sont les dérivées premières des images convolutés L (x, y, σ), une mesure de la réponse d'angle peut être calculée de telle sorte que

$$R = \det(M(x,y,\sigma)) - k(\text{trace}(M(x,y,\sigma)))^2 \quad (2.16)$$

Où k est une constante empirique dans l'intervalle de 0,04 à 0,06. R est grande pour les coins et petite pour les régions plates. Des maximums locaux de R, supérieure à un certain seuil, sont sélectionnés pour représenter les caractéristiques d'angle dans l'image.

FAST aussi ne produit pas des caractéristiques de multi-échelle. Ceci, Rublee et al. [13] ont utilisés une pyramide à l'échelle de l'image (de manière similaire à SIFT), pour trouver des

caractéristiques à chaque niveau de la pyramide.

2.4 RANSAC

RANSAC est une abréviation pour "*RANdom SAmple Consensus*" La méthode a été proposée en 1981 par M. Fischler et Bolles [19]. Il s'agit d'une méthode itérative pour estimer les paramètres d'un modèle mathématique à partir d'un ensemble de données observées qui contient des valeurs aberrantes "*outliers*".

Nous allons utiliser l'algorithme RANSAC dans le but de déterminer la transformation entre deux itérations successives en se basant sur un ensemble de points 3D qui contient des données aberrantes, et à partir de cette transformation, ces points sont isolés. C'est un algorithme assez répandu et qui a montré de très bonnes qualités pour des applications assez diverses.
RANSAC est basé sur le principe suivant : un ensemble de points (minimum trois) dans chaque nuage sont tirés au hasard pour déterminer une base. Une fois les deux bases définies on calcule la matrice de rotation et de translation entre les deux nuages. Les points de l'ensemble du nuage sont alors projetés suivant la transformation calculée de manière à recoller les deux nuages.
On compte ensuite le nombre de paires de points qui sont espacés d'une distance inferieure à un certain seuil. Ce nombre de point donne un score à la transformation. La transformation ayant le meilleur score correspond à la transformation la plus probable. L'algorithme permet finalement de ne conserver que les points correspondant à la meilleure transformation [20].

La difficulté principale avec l'algorithme de RANSAC est de définir le nombre d'itérations et le seuil. Pour le nombre d'itérations, il est possible de définir la valeur idéale selon une probabilité souhaitée. Soit un ensemble de k points, choisis au hasard, si p désigne la probabilité que l'un des itérations N trouverait un ensemble composé uniquement des *« inliers »,* u la probabilité d'avoir un *inlier*, nous avons alors :

$$1 - p = (1 - u^k)^N \qquad (2.17)$$

On peut définir *:* $v = 1 - u$
Comme la probabilité d'avoir une valeur aberrante. De cela, nous pouvons obtenir le nombre d'itérations:

$$N = \frac{log\,(1-p)}{log(1-(1-v^k))} \qquad (2.18)$$

Pour le seuil, cela dépend du problème et de l'application, généralement cela se fait de façon empirique [21].

Algorithme 1 : RANSAC

Données : deux nuages de points 3D

La transformation optimale, bestInliers;

Définir le nombre d'itération *N*

pour *itération* **allant de** 1 **jusqu'à** *N* **faire**

Ensemble prendre k points au hasard

Calculer *la transformation actuelle* à partir de l'ensemble

pour tous les points **faire**

Evaluer *la transformation actuelle* pour tous les points en estimant la distance entre chaque couple de points et calculer son erreur

si *erreur* < *seuil* **alors**
inliers ← *inliers* + *point*
fin si
fin pour
Calculer le nombre des inliers
si *La transformation actuelle* est valide **alors**
Recalculer *La transformation actuelle* à partir des *inliers*
si *La transformation actuelle* est meilleure que *La transformation optimale* **alors**
La transformation optimale ← *La transformation actuelle*
bestInliers ← *inliers*
fin si
fin si

fin pour

retour *La transformation optimale*, *bestInliers*

2.5 ICP

L'algorithme ICP (*Iterative Closest Points*) a été introduit par Besl et McKay [22]. Il nécessite une estimée initiale de la transformation entre deux nuages de points pour pouvoir l'affiner itérativement en minimisant la somme des distances au carré entre un point et son plus proche voisin dans l'autre nuage.

Cet algorithme consiste à calculer, de façon itérative, la matrice de transformation recollant le mieux deux nuages de points. Une connaissance approximative de cette matrice est, par ailleurs, nécessaire, pour son initialisation. Le principe de l'algorithme est assez simple. Il

consiste à appareiller les points deux à deux en utilisant une fonction de cout quadratique (distance euclidienne) entre les points à l'instant k et leurs correspondants à l'instant k+1.
A chaque itération on estime la transformation entre les deux nuages A et B, la source et la cible. A partir de ces couples de points, on déplace le nuage cible conformément à la transformation calculée et la distance entre les deux nuages est alors réévaluée. Celui-ci converge lorsque l'erreur résiduelle de distance est inférieure à un certain seuil. [20]

De cette manière on obtient une estimation fine de la transformation entre les deux nuages de points qui permet un recollage presque parfait.
De nombreuses variantes ont été proposées afin d'améliorer la convergence de l'algorithme.
La convergence de l'algorithme ICP dépend fortement de la qualité de l'estimée initiale fournie, car l'algorithme est susceptible d'être pris au piège dans un minimum local ; Une possibilité fait par Henry [23] pour avoir une bonne estimée initial est d'exécuter RANSAC d'abord pour déterminer une bonne approximation de la transformation puis l'utiliser comme une estimée initiale pour la procédure d'ICP.

Algorithme 2 : ICP

Données : deux nuages de points (P_s P_t),

Estimation initiale de la transformation T_0;
Les critères d'arrêt (nombre maximal d'itérations, seuil minimal)
T(R, t) ← T_0;
Tant que (nombre maximal d'itérations ou seuil minimal) n'est pas atteints **faire**
pour i **allant de** 1 **jusqu'à** N_s **faire**
pour j **allant de** 1 **jusqu'à** N_t **faire**
m_j ← trouver le point le plus proche (Rp_i+t)
Fin pour

Fin pour

si $\| m_j$- (Rp_i+t) $\| \leq d_{max}$ **alors**
$w_i \leftarrow 1$
sinon
$w_i \leftarrow 0$
Fin si
$R, t = \underset{R,t}{\operatorname{argmin}} \sum_{i=1}^{N_s} \sum_{j=1}^{N_t} w_i ||(Rp_i + t) - m_j||^2$
Fin tant que
Retour *La meilleure transformation T(R, t)*

2.6 La fermeture de boucle

La détection de fermeture de boucle est cruciale pour améliorer la robustesse des algorithmes de SLAM. Par exemple, après un long parcours dans des zones inconnues de l'environnement, détecter que le robot est revenu sur une position passée offre la possibilité d'accroître la précision et la cohérence de l'estimation.

Dans ce mémoire on utilise un algorithme de détection de fermeture de boucle qui repose sur un filtrage Bayésien pour le calcul de la probabilité de détection de fermeture de boucle,

L'erreur accumulée au cours du déplacement du robot rend la carte incohérente, entraînant en général une duplication d'une partie de l'information qu'elle contient. Grâce à la détection de fermeture de boucle, une partie de cette erreur peut être corrigée, améliorant au final la carte construite.

L'algorithme détecte si, pour chaque nouvelle image acquise, les points d'intérêt observés correspondent aux points déjà vus. Pour ce faire il compare l'ensemble des points

observés précédemment aux points courants et détermine par un système de vote si la nouvelle instance a déjà été observée ou pas. Si c'est le cas une contrainte est ajoutée entre ce nœud et le nœud correspondant à l'observation précédente. Ceci est d'une importance capitale pour la suite des décisions et des traitements qui seront basés sur la carte comme la planification et la navigation, il faut que cette carte corresponde au mieux à la structure de l'environnement. Sans cela, l'information qu'elle contient ne pourra être utilisée, rendant au final tout le processus de SLAM inutile.

2.6.1 Probabilité de fermeture de boucle

Dans le cadre de la probabilité bayésienne, le problème de détection de fermeture de boucle peut être formulé comme suite : la recherche d'une l'image vienne d'un des lieux déjà visités par le passé dont index satisfait:

$$J = \underset{i=-1,\ldots\ldots,t-p}{\operatorname{argmax}} p(S_{t=i}/\, I^t) \qquad (2.19)$$

Où I^t=I_0, . . .,I_t avec j = -1 si aucune fermeture de boucle a été détectée. Cette recherche n'a pas été effectuée au cours des dernières images p car I_t est toujours semblable à ses voisins (tant qu'ils viennent de lieux à proximité), et ceci entraînerait des détections de fermeture de boucle entre I_t et les images vues récemment (c.-à-d.$I_{t+1}, I_{t+2} \ldots I_{t-(p+1)}$).

$et\ S_{t=i}$ est l'évènement "fermeture de boucle entre l'image courante I_t le lieu I_i à l'instant t"

L'évènement St = −1 correspond à l'évènement "pas de fermeture de boucle à l'instant t", pour les cas où l'image courante ne vient pas d'un lieu existant déjà.

Pour résoudre l'équation (2.19), il faut estimer la probabilité a posteriori $p(S_{t=i}/\, I^t)$ pour tout i $= -1, \ldots,$ t − p .

D'après la loi de Bayes et sous l'hypothèse de Markov cette probabilité peut être décomposée comme suit :

$$p(S_{t=i}/\, I^t) = \eta p(I^t/S_{t=i}) p(S_{t=i}/\, I^{t-1}) \qquad (2.20)$$

Où η est un terme de normalisation.

Soit Z_i l'état des descripteurs à l'instant i. L'indice i est inhérent à l'aspect incrémentiel de la construction des états des descripteurs :

$Z_0 \subseteq Z_1 \subseteq \ldots \subseteq Z_i$, avec Z_0 = ø (c.-à-d. les points d'intérêts extraits depuis l'image I_i sont utilisés pour obtenirZ_{i+1}).

De plus, soit z_i le sous-ensemble de $Z_i$qui caractérisent l'image I_i (i.e. : représente l'image à partir des descripteurs trouvés dans celle-ci).
La séquence d'images I^t peut donc être représenté par la séquence $z^t = z_0, \ldots, z_t$.
Ainsi, La probabilité a posteriori réécrite $p(S_{t=i}/\ z^t)$ peut être exprimée comme suit:

$$p(S_{t=i}/\ z^t) = \eta p(z_t/S_{t=i}) p(S_{t=i}/\ z^{t-1}) \qquad (2.21)$$

Où la probabilité $p(z_t/S_{t=i})$ est considérée comme une fonction de vraisemblance $\int(S_{t=i}/\ z_t)$ de $S_{t=i}$ sachant z_t
Finalement on obtient

$$p(S_{t=i}/\ z^t) = \eta p(z_t/S_{t=i}) \sum^{t-p} p(S_{t=i}/\ S_{t-1=j})\, p(S_{t-1=j}/\ z^{t-1}) \qquad (2.22)$$

Où $p(S_{t=i}/\ S_{t-1=j})$ est le modèle d'évolution temporelle du filtre Bayésien discret
Il est important de noter ici que dans le cadre du filtre Bayésien discret, l'ensemble des lieux formant le modèle de l'environnement évolue dans le temps avec l'acquisition de nouvelles images, divergeant du cadre classique du filtrage Bayésien où cet ensemble serait statique.

2.7 Les différents algorithmes d'optimisation

Le problème de SLAM peut être défini comme une optimisation par la méthode du moindre carré d'une fonction d'erreur qui peut être décrite par un modèle de graphe, en combinant la théorie des graphes et la théorie probabiliste. Un tel graphe est appelé graphe de pose, où les nœuds (ou sommets) représentent une variable d'état décrivant les poses des caméras, et les arêtes représentent les contraintes liées une paire de nœuds. Non seulement la représentation graphique donne une assez intuitive et compacte représentation du problème, mais on adopte une optimisation mathématique, où le but est de trouver la configuration la plus probable des nœuds. Ici, nous présentons différentes approches:

2.7.1 L'algorithme TORO

L'algorithme consiste à trouver la configuration des nœuds du graphe qui minimise l'erreur introduite par les contraintes.

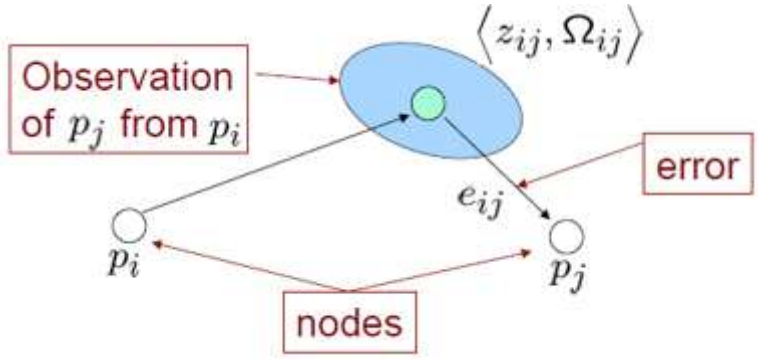


Figure 2.11 : Principe de l'algorithme TORO

Ainsi en notant :

x = $(x_1 \cdots x_n)$ le vecteur décrivant la configuration des nœuds,

z_{ij} : La contrainte entre le nœud i et j, qui correspond à l'arrête i j,

Ω_{ij} : La matrice modélisant les incertitudes sur les contraintes,

f_{ij}(x) : la fonction définissant la contrainte non bruitée entre les points i et j.

On peut définir l'erreur sur la contrainte entre les nœuds j et i, notée e_{ij} , par :

$$e_{ij}(x) = f_{ij}(x) - z_{ij} \quad (2.22)$$

On peut aussi définir le résidu $r_{ij}(x)$ par :

$$r_{ij}(x) = -e_{ij}(x) \quad (2.23)$$

L'intérêt de l'utilisation du résidu est que ce vecteur donne la 'direction' dans laquelle il faut changer le graphe (l'erreur diminue dans ce sens).

Si on fait l'hypothèse d'un bruit gaussien on peut exprimer le log négatif de la probabilité de F_{ij} par :

$$\begin{aligned} F_{ij} &= (f_{ij}(x) - z_{ij})^T \Omega_{ij} (f_{ij}(x) - z_{ij}) \\ &= e_{ij}(x)^T \Omega_{ij} e_{ij}(x) \\ &= r_{ij}(x)^T \Omega_{ij} r_{ij}(x) \end{aligned} \quad (2.24)$$

En supposant que les observations sont indépendantes, le log négatif de la probabilité pour une configuration x est :

$$\begin{aligned} F(x) &= \sum_{<i,j> \in C} F_{ij}(x) \\ &= \sum_{<i,j> \in C} r_{ij}(x)^T \Omega_{ij} r_{ij}(x) \end{aligned} \quad (2.25)$$

Finalement l'objectif est de trouver la configuration x* qui maximise la probabilité des observations, ce qui s'écrit :

$$x^* = \text{argmin } F(x) \quad (2.26)$$

Où * est le symbole pour désigner la solution optimale [20].

2.7.1.1 Descente de gradient stochastique (SGD)

Pour minimiser la fonction F une approche présente dans par Olson [24] consiste à sélectionner de manière itérative les contraintes et à déplacer les nœuds dans le sens permettant une diminution de l'erreur. Les nœuds sont mis à jour selon la formule :

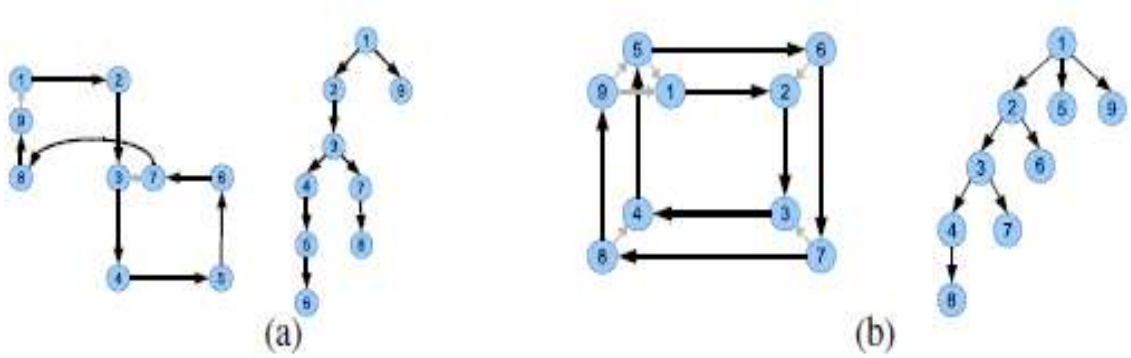
$$x^{t+1} = x^t + \lambda . H^{-1} J^T{}_{ji} \Omega_{ji} r_{ji} \quad (2.27)$$

Ou x représente l'ensemble des variables décrivant la localisation de la pose dans le graphe. H est la hessienne du système qui représente la courbure de la fonction d'erreur. J est la jacobéenne. Le facteur λ sert simplement à éviter des oscillations dans l'algorithme en diminuant progressivement la part du résidu utilisée dans la mise à jour des variables.

2.7.1.2 Paramétrisation

La qualité de la paramétrisation joue un rôle important pour l'efficacité de l'algorithme. En effet avec la méthode SGD il faut à chaque itération définir quelle partie du graphe doit être mise à jour. Avec la méthode présentée par Olson [24] le problème est que pour l'optimisation d'une contrainte entre les nœuds i et k, faudra mettre à jour tous les nœuds j pour j=i à k.
Dans TORO une autre méthode exploitant au mieux la topologie du graphe a été développée utilisant un arbre pour la représentation des données.
Notre algorithme utilise une paramétrisation à base d'arbre pour décrivant la configuration des nœuds dans le graphe, l'arbre de paramétrisation est construit comme suit :

1- Nous attribuons un identifiant unique à chaque nœud sur la base des horodatages et la succession des nœuds.
2- Le premier nœud est la racine de l'arbre (donc pas de parent)
3- En tant que parent d'un nœud, nous choisissons le nœud avec le plus petit identifiant pour lequel une contrainte pour le nœud courant existe.



Figures 2.12 : (a) et (b) illustre les graphiques et les arbres correspondants.

(1) et (2) Deux petits exemples graphiques et les arbres utilisés pour déterminer les paramétrisations. Les petites connexions grises sont des contraintes additionnelles résultent de la détection de fermeture de boucle et les connections noires résultent du calcul de transformation entre deux poses successives.

Cet arbre a une série de propriétés intéressantes lors de l'application de cet algorithme d'optimisation pour trouver une configuration des nœuds avec une erreur minimale. Ces propriétés sont les suivantes:

1- L'arbre peut être construit de façon incrémentale: lors de l'ajout d'un nouveau nœud, il n'est pas nécessaire de changer l'arbre existant.

2- Dans le cas où le robot se déplace à travers les boucles, l'interaction entre les mises à jour des nœuds appartenant à chaque boucle dépend du nombre de nœuds qui sont en commun.

3- Lorsque le robot traverse une zone déjà explorée, des contraintes seront ajoutées entre les nœuds et les nouveaux ajoutés précédemment, la longueur du chemin dans l'arbre entre ces nœuds sera faible. Cela signifie que seul un petit nombre de nœuds doivent être mis à jour.

La deuxième propriété est illustrée sur la figure 2 (a). Les deux boucles dans cette image ne sont reliées que par la contrainte entre les nœuds 3 et 7. Ils sont les seuls nœuds qui sont mis à jour.

La troisième propriété est illustrée sur la figure 2 (b). Ici le robot revisite une zone. Les nœuds 1 à 4 sont choisis comme parents pour les nouveaux nœuds. Il en résulte des trajets courts dans l'arbre lors de la mise à jour des positions des nœuds lors du passage par des zones connues. La complexité de l'approche présentée dépend de la longueur de la trajectoire, et non sur la taille de l'environnement [25].

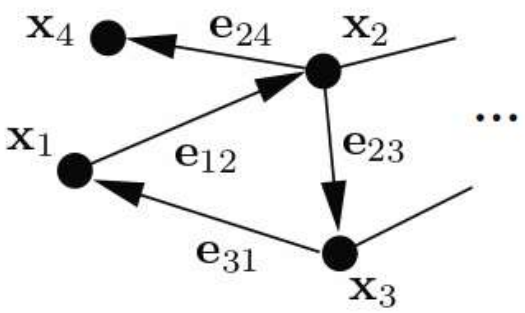
2.7.2 g2o

(*General Framework for Graph Optimization*), ou g2o est une bibliothèque développée par Kummer le et al. [26] et dédié aux méthodes de SLAM.

g2o [26], est une bibliothèque écrit sous C ++ pour effectuer l'optimisation par les moindres carrés non linéaires des problèmes qui sont modélisés sous forme d'un graphe. Cette bibliothèque fait l'abstraction de nombreux nœuds et contraintes par des classes qui peuvent être directement appelées et utilisées, comme VertexSE3Expmap pour représenter les poses du robot dans l'espace euclidien SE(3), *Vertex SBA Point XYZ* pour représenter les points 3-D, *Edge Project XYZ 2 UV* pour représenter les observations de points 3D dans le plan d'image fournis par la camera. En outre, des algorithmes des solveurs d'optimisation typiques sont mises en œuvre. Avec la bibliothèque g2o, pour faire le SLAM il est nécessaire de définir les nœuds et les contraintes, en les ajoutant au solveur fournies par g2o qu'il exécutera tous les trucs d'optimisation. g2o est une bibliothèque largement utilisée en SLAM, adopté dans de nombreux travaux.

Les problèmes de SLAM exigent un back-end pour affiner la carte et les poses construits dans son front-end. Le back-end est généralement repose sur un filtrage (comme EKF) ou l'optimisation du graphe .De nos jours, l'optimisation du graphe est beaucoup plus populaire.

L'idée générale de l'optimisation du graphe est d'exprimer le problème de SLAM comme une structure graphique. Comme la figure ci-dessous, le graphe contient deux types d'éléments, les nœuds (sommets) et des contraintes (arêtes). Pour les problèmes de SLAM, la pose du robot ou d'un point d'intérêt dans la carte est désignée comme un nœud, tandis que les observations et le modèle géométrique entre deux poses du robot ou entre une pose et un point d'intérêt ou entre deux points d'intérêts sont désignés par des contraintes qui reliaient certains les nœuds.



$$\begin{aligned}\mathbf{F}(\mathbf{x}) = &\ \mathbf{e}_{12}^{\top}\,\boldsymbol{\Omega}_{12}\,\mathbf{e}_{12} \\ &+ \mathbf{e}_{23}^{\top}\,\boldsymbol{\Omega}_{23}\,\mathbf{e}_{23} \\ &+ \mathbf{e}_{31}^{\top}\,\boldsymbol{\Omega}_{31}\,\mathbf{e}_{31} \\ &+ \mathbf{e}_{24}^{\top}\,\boldsymbol{\Omega}_{24}\,\mathbf{e}_{24} \\ &+ \dots\end{aligned}$$

Figure 2.13 : Un exemple illustrant comment représenter une fonction sous forme d'un graphe

Étant donné un graphe, l'optimisation de ce graphe vise à trouver une estimation optimale des configurations des nœuds qui minimisent les erreurs qui ont déterminé par les contraintes. Par

conséquent, SLAM back-end est transformé en une minimisation par des moindres carrés, qui peut être décrit par l'équation suivante [26]:

$$F(x) = \sum_{<i,j>\in C} e_{ij}(x_{ij}, z_{ij})^T \Omega_{ij} e_{ij}(x_{ij}, z_{ij}) \quad (2.28)$$

$$x^* = \text{argmin } F(x)$$

Ou

$x = (x_1 \cdots x_n)$ le vecteur décrivant la configuration des nœuds.

z_{ij} : La contrainte entre le nœud i et j, qui correspond à l'arrête ij.

Ω_{ij} : La matrice modélisant les incertitudes sur les contraintes.

e_{ij}: L'erreur sur la contrainte entre les nœuds j et i.

2.8 CONCLUSION

Ce chapitre est une présentation des différents algorithmes qu'on va utiliser pour résoudre le problème du SLAM. A partir de ces algorithmes nous pourrions avoir un aperçu sur la résolution de ce problème qui comprend les étapes suivantes :

- L'acquisition des données à l'aide des mesures que donne le capteur
- Extraction des points d'intérêt
- Estimation de pose à travers la variation relative des points entre deux itérations
- La mise à jour de la carte en fonction des nouvelles poses et les mesures correspondantes.

Le but de ce chapitre n'est pas de ré-établir de manière rigoureuse toute la théorie derrière ces algorithmes, Il est toutefois important de poser les bases théoriques des principes de fonctionnement afin de pouvoir interpréter les résultats obtenus.

Dans le chapitre suivant nous expliquerons en détaille la contribution de ces algorithmes pour la construction en temps réel des cartes en 3D très précises pour modéliser des environnements intérieur

3

L'approche V-SLAM

3.1 INTRODUCTION

Le chapitre précédent portait sur la théorie derrière les algorithmes et méthodes utilisés dans ce projet. Cela donne un aperçu général sur le traitement fait par l'approche de V-SLAM. En robotique mobile, plusieurs types de représentation de l'environnement ont été développés. Tous d'abord on va présenter le processus de traitement du V-SLAM, en détaillant chaque étape, ensuite on introduira les différentes techniques du SLAM en 2D pour pouvoir le comparer avec le SLAM en 3D et monter ses inconvénients et ses limitations pour valider notre choix.

3.2 V-SLAM

Le grand défi pour un robot autonome c'est être capable d'interagir avec son environnement, la localisation et la cartographie simultanée avec le capteur Kinect embarqué sur le robot, appelé RGB-D SLAM ou V-SLAM, a été développé à cet effet, ce qui déclenche de nombreuses exigences pour les algorithmes de traitement de l'information. Par conséquent, la gestion des données de capteur est très importante.

3.2.1 INTRODUCTION

Dans la problématique de la localisation et de la cartographie simultanées, le robot doit inférer sa position dans une carte de l’environnement tout en construisant cette carte au fur et à mesure de sa progression. C’est notamment l’aspect concurrentiel qui rend cette tâche difficile.

Le V-SLAM est divisé en deux parties: "Front-end" et "back-end». Le "front-end" permet de construire la carte, et le "back-end" permet d'optimiser cette carte.

3.2.2 “Front-End”

Entre deux itérations la Kinect permet l'acquisition de deux images RGB et de deux images de profondeur de la scène observée. Ensuite, La détection des points d'intérêts et l'extraction des descripteurs sont faits à partir des deux images couleurs, des algorithmes performants permettent de mettre en relation ces images à partir de ces points, grâce aux images de profondeur, les points d'intérêts sont projetés dans l'espace 3D et forment des nuages, l’association des points de ces deux nuages donne une estimation de la pose en calculant la transformation entre ces deux nuages.

Les algorithmes du front-end sont montrés dans la figure suivante:

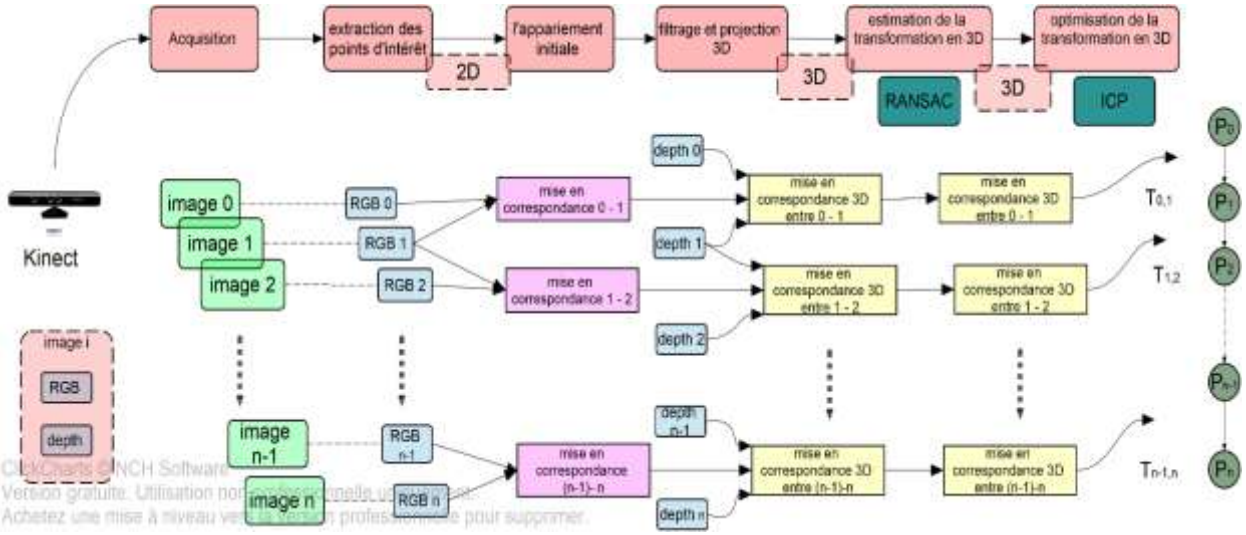


Figure 3-1: Les algorithmes de front-end de RGB-D SLAM

3.2.2.1 La détection des points d'intérêts et l'extraction des descripteurs

L'avantage de la Kinect est de fournir à la fois des images couleurs et des images de profondeur. Il y a deux aspects concernant le calcul des points d'intérêt: la détection d'un point d'intérêt, qui identifie une zone d'intérêt, et son descripteur, qui caractérise cette zone. Typiquement, le détecteur identifie une région contenant une forte variation d'intensité lumineuse, Le centre de cette région représente un point d'intérêt. Le descripteur est généralement calculé en mesurant les principales orientations des points environnants, conduisant à un vecteur multidimensionnel qui identifie le point d'intérêt donné, Ceci se fait aisément avec la bibliothèque Open CV.

3.2.2.2 La mise en correspondance en 2D

Une fois les points sont calculés, il est possible de mettre en correspondance les deux images. La première mise en correspondance est faite en 2D par la recherche du plus proche voisin,

Cette étape consiste à mettre en relation les deux images à partir des descripteurs calculés précédemment en cherchant la distance euclidienne minimale entre les points.

Pour augmenter la robustesse, les points d'intérêt dont le rapport de la distance du plus proche voisin et la distance du deuxième proche voisin est supérieure à un seuil donné sont rejetés.

3.2.2.3 Calcul des coordonnées des points

Une fois que la mise en correspondance en 2D soit faite, il est possible de trouver une transformation reliant les deux ensembles de points c.-à-d. l'opération qui projette chaque point

de l'image source au point correspondant dans l'image cible. Cette transformation est composée d'une matrice de rotation et un vecteur de translation, elle peut être écrite en utilisant une matrice 4x4 des coordonnées homogènes. Nous pouvons alors projeter un point P où P = $(x, y, z, 1)^T$ en appliquant simplement cette transformation au point:

Pour faciliter les étapes qui suivent on va définir les coordonnées comme suit :

$$\begin{cases} x: profondeur, positive\ vers\ l'avant \\ y: hauteur, positive\ ascendant \\ z: largeur, positive\ \text{à}\ droite \end{cases}$$

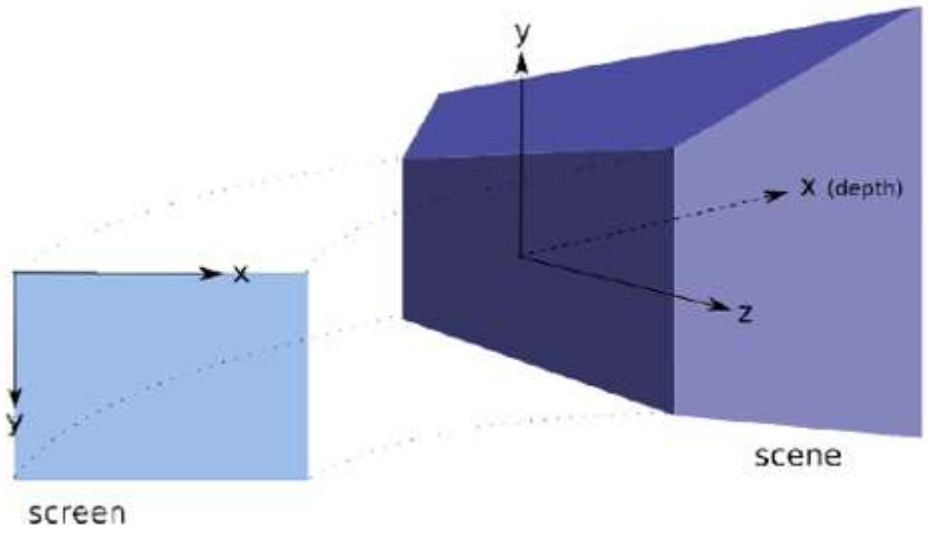


Figure 03-2: Les deux différentes coordonnées de système, de "l'écran" (comme on le voit par les capteurs RGB-D) à la scène 3D dans le monde réel.

De la valeur de profondeur (donnée en mm par OpenNI) et les coordonnées de l'écran (image RGB de résolution de 640x480 pixels), il est possible de calculer les coordonnées des points en 3D.

Soit (u, v) les coordonnées du point en pixels, nous avons alors:

$$P(x,y,z)\begin{cases} x = profondeur \\ y = -\left(v - \frac{480}{2}\right) * \frac{profondeur}{distance\ focale} \\ z = \left(u - \frac{640}{2}\right) * \frac{profondeur}{distance\ focale} \end{cases}$$

3.2.2.4 Estimation de la transformation

Après avoir déterminé les coordonnées 3D des points, On dispose donc de deux nuages en 3D formés par les points d'intérêt, la transformation du mouvement peut être estimée selon

la position des points dans les deux nuages ce qui permet d'estimer efficacement les déplacements de la camera entre deux instants.

La recherche de la transformation qui permet de recoller convenablement ces deux nuages de points se fait avec la combinaison RANSAC plus ICP.

L'algorithme RANSAC que nous avons détaillé dans la section 2.3 est utilisé pour déterminer les points qui permettent d'estimer cette transformation entre les deux nuages le plus correctement possible.

L'estimation de mouvement avec RANSAC est en générale assez bonne mais dans le cadre de la construction d'une carte il est préférable d'affiner cette estimation avec une autre méthode robuste comme ICP. Ainsi on obtient une première estimation du mouvement avec une méthode rapide puis on l'affine par la suite avec une méthode précise initialisée avec ce résultat. On bénéficie donc d'une vitesse de calcul relativement bonne et d'une précision correcte du fait du second algorithme.

3.2.3 Back-End

Dans le chapitre précédent, nous avons vu comment déterminer une transformation entre deux itérations. A partir des transformations, une estimation de pose de la caméra peut être calculée. Chaque pose est ensuite insérée dans un graphe sous forme d'un nœud. Une fois une fermeture de boucle est détectée, de nouvelles contraintes peuvent être insérés dans le graphe, Cette nouvelle contrainte permet de recalculer la configuration du graphe et de réduire considérablement l'erreur accumulée depuis le départ, Le graphe peut alors être optimisée en minimisant une erreur, et les poses sont mises à jour en fonction des nouvelles données après l'optimisation, ce qui réduit la dérive globale.

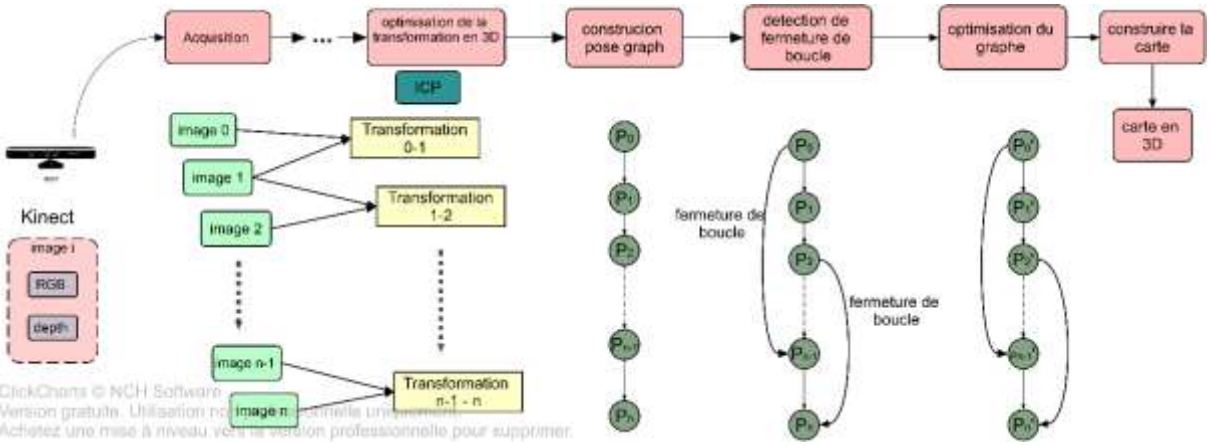


Figure 03-3 : Les algorithmes de back-end de RGB-D SLAM

3.3.3.1 L'Estimation de la pose

Connaissant la pose initiale, la première étape consiste à déterminer une estimation de n'importe quelle pose après une succession de transformations. Considérant une séquence finie d'itérations [image0, …, imageN], soit P_k la pose liée a k^{eme} itération tel que k ∈ [0; N], Cette pose peut être représentée par une transformation homogènes 4x4, où R est une matrice de rotation 3x3 et t est un vecteur de translation:

$$P_k = \begin{bmatrix} & & & t_x \\ & R & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Pour i > 0, nous avons la transformation T_{i-1}^{i} qui liée la position P_{i-1} à la position Pi. Chaque transformation T est représentée par une matrice homogène 4x4. Si P_0 détermine la position de départ, on peut alors calculer la position Pi en combinant toutes les transformations :

$$P_k = \prod_{i=k}^{1} T_{i-1}^{i} P_0 \tag{3.2}$$

Comme un choix arbitraire, nous pouvons définir la position initiale d'être à l'origine du système de coordonnées, qui est donnée par la I_4 de la matrice d'identité.

$$P_0 = I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

3.3.3.2 La fermeture de boucle

Pour chaque pose, un nœud est inséré dans un graphe avec une contrainte reliant la nouvelle pose avec la précédente. La contrainte représente la transformation entre les deux itérations.

Afin de minimiser l'erreur, l'optimisation du graphe repose sur des contraintes entre les nœuds. Un événement intéressant se produit lorsque le robot se déplace à une position déjà visité dans le passé. Ceci est référé par la détection de fermeture de boucle. Ceci est illustré sur la figure 3.8. A chaque fois une fermeture de boucle est détectée entre deux poses, une nouvelle contrainte est insérée dans le graphe.

Sans faire une hypothèse sur la trajectoire suivie par le robot, et simplement en gardant l'histoire des images passées, il est possible de vérifier si l'image actuelle correspond à une image passée. Si l'observation actuelle contient suffisamment de similitudes avec une autre

observation vue dans le passé, une transformation peut être calculée entre ces images et une nouvelle contrainte peut être insérée de celle-ci. Ceci peut être répété plusieurs fois. Grâce à ces nouvelles contraintes, l'erreur cumulée du graphe peut être considérablement réduite.

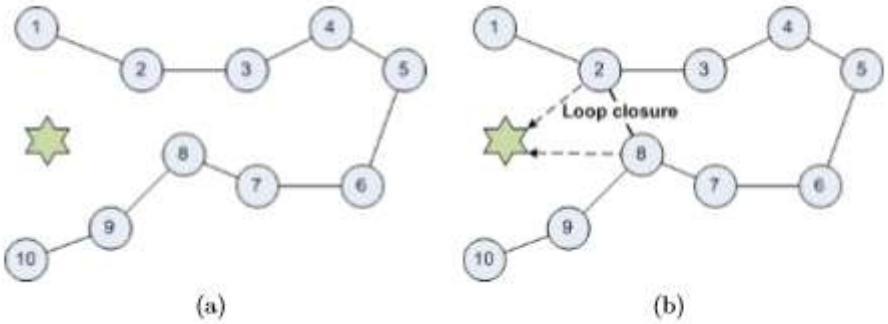


Figure 3-4: Détection de fermeture de boucle. (a) Les poses sont reliées entre elles par des contraintes, chacune d'elle représente une transformation. (b) L'observation d'un point d'intérêt commun déjà vu dans le passé, peut déclencher la création d'un nœud.

3.3.4 Optimisation du graph

La détection de la fermeture de boucle conduit à des contraintes nouvelles qui sont insérés dans le graphe reliant les poses connexes. L'optimisation du graphe consiste à trouver la configuration optimale des nœuds en tenant compte toutes les contraintes données, comme illustré sur la figure suivante:

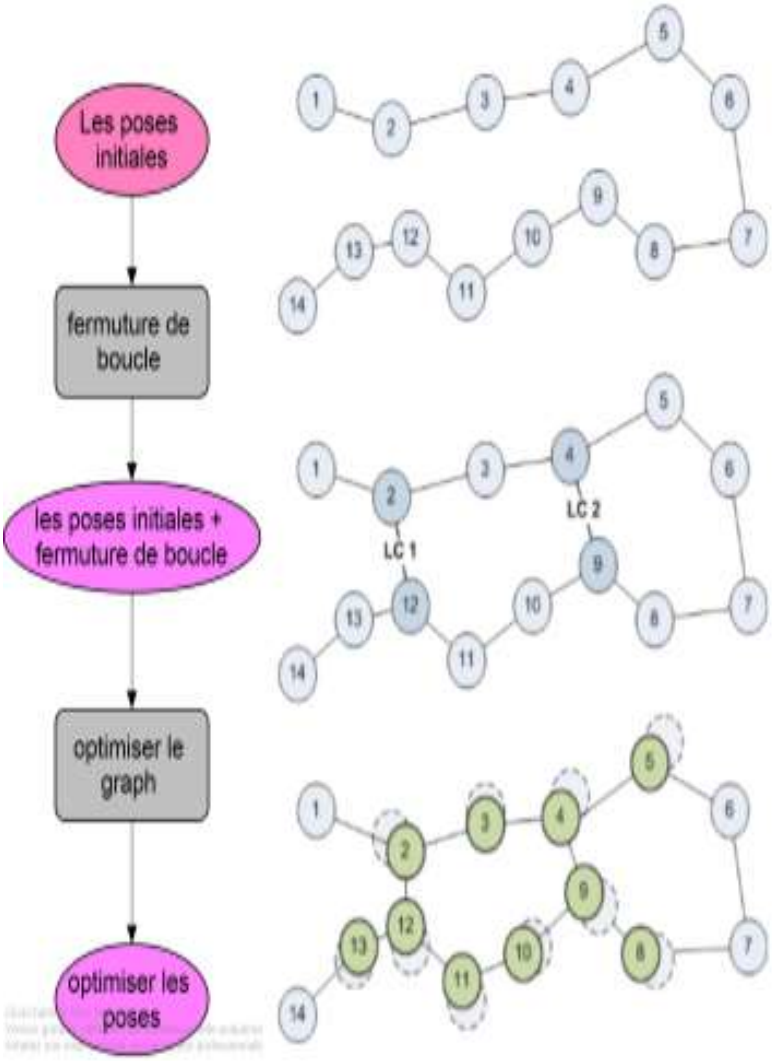


Figure 3-5: la procédure d'optimisation graphique

Une fois le graphe a été optimisé, les nœuds sont corrigés et les nouvelles estimations de la pose de la caméra peuvent être extraites.

3.3.5 Construction de graph

Enfin, une fois que les poses de la caméra sont déterminées, la reconstruction de la scène peut être effectuée. Pour chaque image, un nuage de points 3D peut être généré à partir des données RGB et de profondeur, fournissant les informations de couleur pour chaque point. Toutefois, afin de reconstruire toute la scène, Ces nuages doivent être combinés ensemble à partir d'un point de vue unique. Ce processus est appelé l'enregistrement des nuages de points. Chaque nuage de points est transformé en projetant tous ses points en fonction de la position

de la caméra correspondante, donnée par l'équation :

$$P_k = \prod_{i=k}^{1} T_{i-1}^{i} P_0 \qquad (3.4)$$

Cela se fait par rapport à la première pose, qui est prédéfinie. Les nuages de points transformés sont ensuite concaténés ensemble pour construire la scène.

3.3 SLAM en 2D

La cartographie et localisation simultanée (SLAM) est la technique la plus utilisée actuellement pour la navigation de robots autonomes. Le robot crée une carte de l'environnement et, simultanément, se localise grâce à celle-ci. La carte ainsi que la position du robot sont totalement inconnues au départ. A priori le robot ne possède aucune information sur le type de l'environnement (statique ou dynamique), ni sur sa topologie. La carte de l'environnement est créée à partir de ses observations qui sont en réalité une collection d'amers (des objets remarquables de l'environnement) dont il estime la position à partir de sa propre position et des données acquises par ses capteurs. Ces mêmes amers, utilisés conjointement avec l'odométrie, permettent d'estimer la position du robot, Les estimations sont donc dépendantes les unes des autres.

Donc le robot a besoin de construire de manière incrémentale son modèle du monde car il utilise ce modèle à tout instant [5]. Son fonctionnement est représenté par la figure suivante :

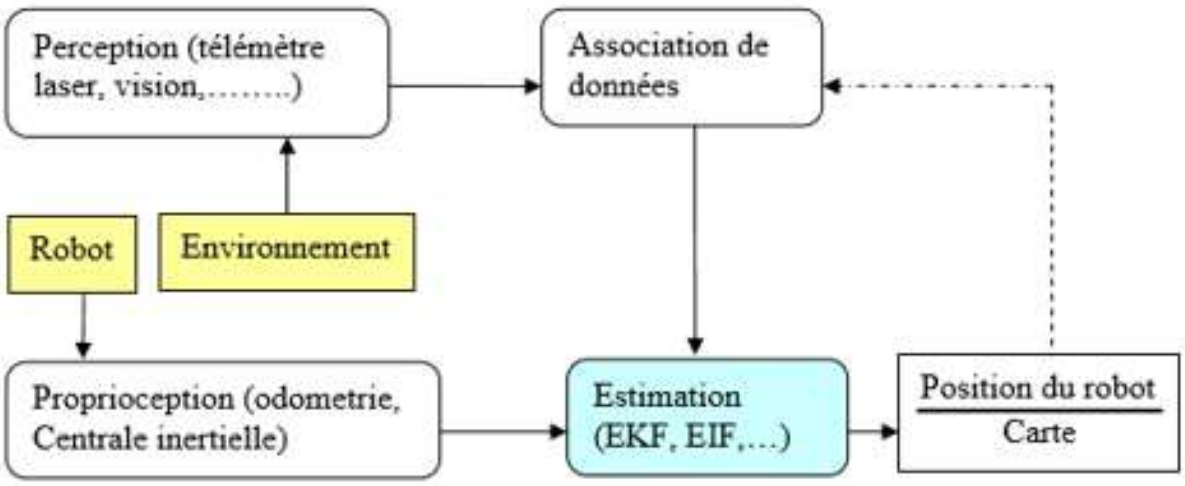


Figure 3-6 : la procédure d'optimisation graphique

La figure (3.6) présente une vue générale d'un algorithme SLAM. Les différents blocs sont :

- **Perception :** Les algorithmes de perception fournissent à partir des données issues d'un

capteur tel qu'un télémètre laser, une caméra . . ., un ensemble d'observations des objets présents dans l'environnement qui constitue des amers.

- **Association de données :** Étant donné une observation, le robot doit décider de quel amer il s'agit si c'est un amer déjà présent dans la carte, ou bien si c'est un nouvel amer à ajouter à la carte.

- **Proprioception :** Les données proprioceptives sont des mesures internes au robot qui renseignent sur l'évolution de son état. Pour le SLAM, nous nous intéressons à la position du robot dont l'évolution est typiquement fournie par des capteurs de l'odométrie.

- **Estimation :** L'algorithme d'estimation doit intégrer les données issues de la perception ainsi que de la proprioception afin de fournir une estimée de la position du robot et des positions des amers, ainsi que les incertitudes associées à ces estimées.

3.3.1 Représentation 2D d'un environnement d'intérieur

La cartographie en robotique mobile est un outil qui permet au robot de modéliser l'environnement dans lequel il évolue. Il y a plusieurs types de cartes établies, parmi eux nous citons les cartes métriques, topologiques et grille d'occupation. Ces cartes permettent seulement au robot de se localiser de manière globale, et de planifier la trajectoire de façon sous optimale. Les algorithmes du SLAM peuvent être classés en deux grandes catégories selon le modèle de représentation de l'environnement.

3.3.1.1 La carte métrique

Dans une carte métrique, l'environnement est représenté par un ensemble d'objets auxquels sont associées des positions dans un espace métrique, Les objets mémorisés dans la carte peuvent être très divers. Dans certaines implantations, ces objets correspondent aux obstacles que le robot pourrait rencontrer dans son environnement. La carte de l'environnement correspond alors directement à l'espace libre, c'est-à-dire à l'espace dans lequel le robot peut se déplacer. L'avantage principal des cartes métriques est de permettre de représenter l'ensemble de l'environnement, et non un petit sous-ensemble des lieux, en plus cette représentation est facilement interprétable par un humain.

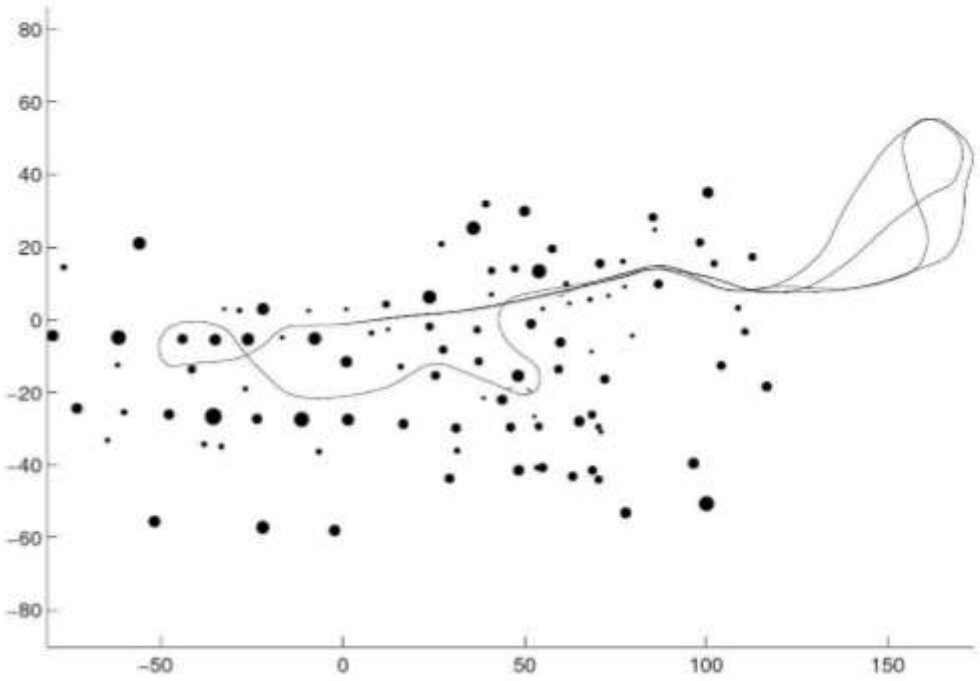


Figure 3-7: Exemple de carte métrique 2D (vue de haut) composée d'amers ponctuels (cercles). La trajectoire d'un véhicule au sein de cette carte est également donnée. Source : [38].

3.3.1.2 La carte topologique

Dans les cartes dites topologiques, l'environnement est segmenté sous la forme d'un graphe dont les nœuds correspondent à des lieux distincts, alors que les arêtes encodent les relations entre nœuds voisins. Cette représentation est notamment plus adaptée aux environnements de grande taille, et elle permet une navigation symbolique pour atteindre un lieu en particulier.

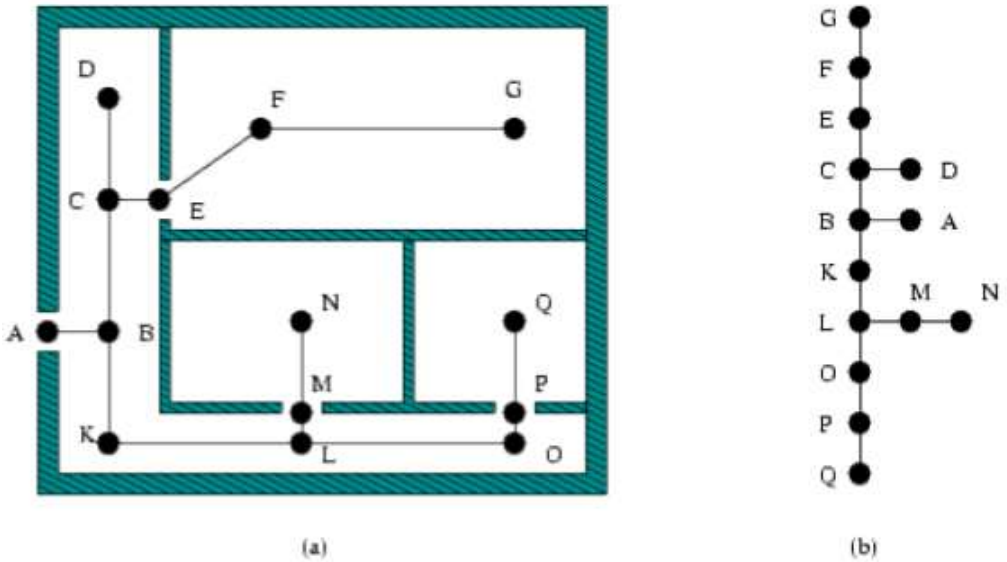


Figure 3-8: Exemple de carte topologique 2D (vue de haut). (a) projection dans le plan. (b) Structure topologique uniquement [39]

3.3.1.3 Grille d'occupation

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 (probabilité, histogramme, etc.) indiquant si l'espace correspondant est plutôt libre ou occupé. La figure 3.9 illustre une grille d'occupation. [9]

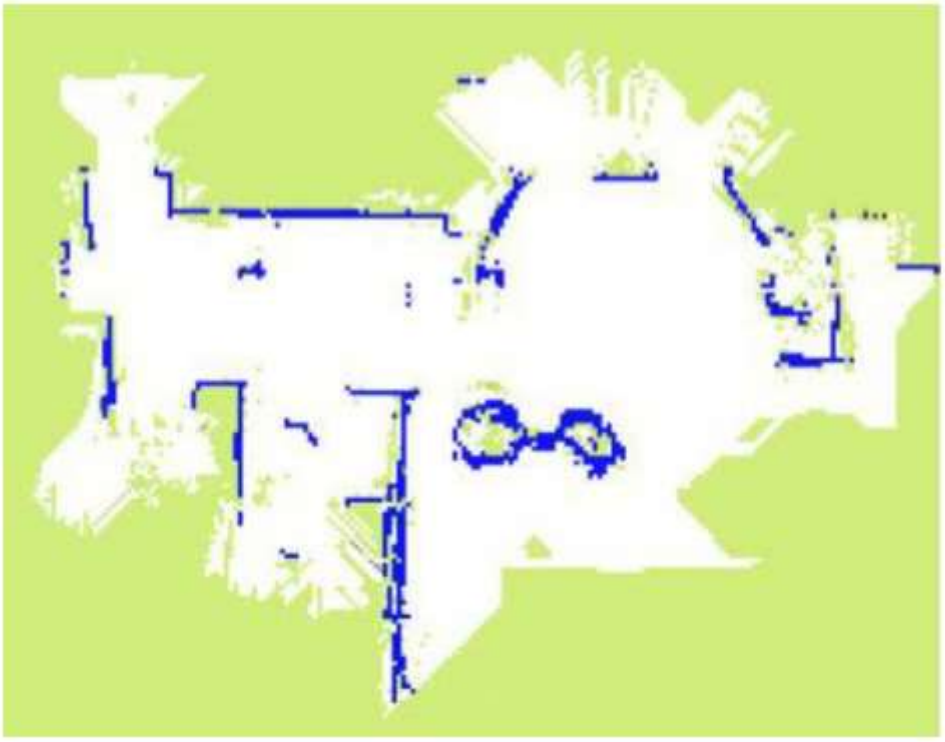


Figure 3-9: Exemple d'une grille d'occupation 2D. Les valeurs d'occupation sont codées comme suit : (Vert : inconnu ; blanc : libre ; bleu : occupé) La superficie est (10 x 10m), une cellule correspond à (5 x 5cm)

3.3.2 Les différentes techniques du SLAM 2D

Plusieurs techniques du SLAM 2D sont disponibles sous ROS. Les cinq techniques les plus populaires et les plus utilisées du SLAM discutées dans ce projet sont disponibles sous ROS.

Tous les algorithmes reconnus pour la cartographie du robot ont une caractéristique commune: ils dépendent tous des probabilités. L'avantage de l'application des probabilités est la robustesse au bruit de mesure et la capacité de représenter l'incertitude dans le processus de mesure et d'estimation. La plupart des modèles probabilistes utilisés pour résoudre le problème de la cartographie reposent sur la probabilité de Bayes [30].

Le filtre de Kalman (KF) est l'une des approches les plus populaires qui a une forte implémentation du filtre Bayes [30]. Le cycle du filtre de Kalman est constitué de deux étapes fondamentales :

1- Étape de prédiction : durant laquelle on estime l'état du système à l'instant t en utilisant l'estimation corrigée à l'instant (t-1).
2- Étape de correction : durant laquelle on corrige l'estimation de l'état du système à l'instant t en utilisant les informations sensorielles reçu à l'instant t.

Le filtre de Kalman Étendu est une amélioration du filtre de Kalman basique qui permet de résoudre le problème de la non-linéarité dans le SLAM 2D.

Une série de tests sur les propriétés de convergence de la solution à base d'EKF au problème du SLAM 2D non linéaire est effectuée dans [31].

Le filtre particulaire (PF Un filtre particulaire est un filtre récursif qui permet d'estimer l'état a

Le filtre de Kalman Étendu est une amélioration du filtre de Kalman basique qui permet de résoudre le problème de la non-linéarité dans le SLAM 2D.

Une série de tests sur les propriétés de convergence de la solution à base d'EKF au problème du SLAM 2D non linéaire est effectuée dans [31].

Le filtre particulaire (PF Un filtre particulaire est un filtre récursif qui permet d'estimer l'état a postériori en utilisant un ensemble de particules (l'hypothèse de Markov [32]) et ainsi il est capable de traiter les systèmes fortement non linéaires avec un bruit non gaussien, La probabilité a posteriori est représentée par un ensemble de particules et à chaque particule est associé un facteur d'importance. Montemerlo et al. [33] a proposé une nouvelle approche appelée FastSLAM. Qui combine le filtrage particulaire avec d'autres méthodes et il y a aussi les algorithmes de graph-based SLAM, qui couvrent les faiblesses de PF et les techniques de EKFs [34].

Par la suite une brève description des cinq techniques du SLAM sont menées: Hector SLAM, Gmapping, KartoSLAM, CoreSLAM et LagoSLAM.

3.3.2.1 Hector SLAM

Hector SLAM consiste à combiner un système 2D SLAM basé sur un scan matching robuste et la technique de navigation 3D à l'aide d'un système de détection inertielle [35].

L'estimation de la pose en 2D est basée sur l'optimisation de l'alignement des extrémités de faisceau laser avec la carte conçue jusqu'à maintenant. Les extrémités du faisceau sont projetées dans la carte réelle et les probabilités d'occupation sont estimées. Le scan matching est résolu en utilisant une équation de Gauss-Newton, qui cherche la meilleure transformation qui fait la mise en correspondance des faisceaux laser et de la carte.

3.3.2.2 Gmapping

Le Gmapping est une technique du SLAM proposé par Grisetti et al. , basé sur le filtre particulaire de Rao-Blackwellization, cette approche calcule la distribution la plus précise en se basant sur la probabilité d'observation de l'information la plus récente issues du capteur, l'odométrie et le scan matching. Cela permet de tirer les particules de manière plus précise, ce qui réduit le nombre de ces dernières ce qui diminue énormément le temps de calcul.

3.3.2.3 KartoSLAM

KartoSLAM4 est une approche de Graph-basedSLAM développé par SRI International, Karto Robotics, il utilise comme solver [40] une décomposition matricielle optimisée et non itérative de Cholesky pour les systèmes linéaires de partition.

3.3.2.4 CoreSLAM

Est une alternative pour l'algorithme TinySLAM qui a été créé dans le but de gagner plus de stabilité et de fiabilité, tout en respectant un cadre d'exécution en temps réel avec une faible perte de la performance [36]. L'algorithme est divisé en deux étapes différentes: le calcul de distance et la mise à jour de la carte.

3.3.2.5 LagoSLAM

La base des algorithmes de Graph_based SLAM est la minimisation d'une fonction non linéaire après la construction du graph. Cependant, ce processus d'optimisation est fortement lié à l'estimation initiale de convergence. Carlone et al. [37] ont développé une nouvelle approche appelée LagoSLAM6 (Approximation linéaire pour Graph Optimization), dans lequel le processus d'optimisation ne nécessite aucune estimation initiale. En outre, n'importe quelle technique d'optimisation peut être utilisée. En fait, l'algorithme disponible sous ROS donne possibilité de choisir entre trois techniques d'optimisation différentes: base-Tree Network

Optimizer (TORO), G2O [26] et LAGO [37].

3.4 Comparaison entre le SLAM en 2D et le SLAM en 3D

Les points suivants résument les principales conclusions de la cartographie en 2D et en 3D :

- La cartographie en 3D fournit plus de détaille et un dimensionnement fiable et donne aussi des bons résultats sur formes des objets présents dans la scène.
- La cartographie en 3D se fait en couleur contrairement à la cartographie 2D
- Les capteurs utilisés pour le SLAM 3D sont moins sensible au bruit que ceux utilisés pour le SLAM 2D.
- Nous pouvons voir la carte 3D sous différentes angles, alors que la carte en 2D permet d'avoir seulement une vue de dessus.
- On peut distinguer n'importe quelle forme quelconque dans la carte 3D ce qui est impossible dans la carte 2D

Une représentation fiable de la scène est nécessaire pour calculer la position du robot et vice versa une bonne estimation de la pose du robot donne à une carte précise.

3.5 Conclusion

Dans ce chapitre nous avons présenté l'approche que nous utilisons pour résoudre notre problème du V-SLAM qui est basé sur la méthode de la théorie des graphes (Graph-Based) ; on a résumé les étapes principales du V-SLAM implémenté qui sont l'extraction et l'association des points d'intérêt à partir des images couleurs acquises par la Kinect, la construction 3D et l'estimation de la pose, détection de fermeture de boucle et enfin la construction de la carte, nous avons clôturé ce chapitre par une comparaison entre SLAM en 2D et SLAM en 3D.

4

Résultats expérimentaux et discussion

4.1 INTRODUCTION

Ce chapitre présente le résultat et l'analyse des différents algorithmes utilisés dans ce projet. Les performances des différentes méthodes sont comparés les uns contre les autres pour trouver une approche combinée efficace.

Les différentes approches de détection/description des points d'intérêt sont évaluées en premier en comparant les trois approches entre elles. Cela fournira des points fiables pour l'estimation de la transformation entre deux nuages successifs de points d'intérêts. Nous montrerons par la suite l'efficacité d'ICP pour le recollage des nuages de points et à la fin de ce chapitre une implémentation d'un module de SLAM sera faite.

4.2 ROS

Le système d'exploitation utilisé dans notre projet est nommé ROS (Robot Operating System) Dans ce chapitre, nous présentons et les différents outils utilisés par ROS pour exécuter les tâches robotiques.

4.2.1 Définition de ROS

ROS est une open source qui doit s'interfacer avec le system d'exploitation LINUX et Comme son nom l'indique, ROS (Robot Operating System) est un système d'exploitation pour robots qui fournit des bibliothèques et des outils pour aider les développeurs de logiciels à créer des applications robotiques. Il fournit une abstraction matérielle, des pilotes de périphériques, les bibliothèques, les outils de visualisation, le passage de messages, la gestion des paquets, et plus encore. Cet outil offre plusieurs fonctionnalités à différents niveaux, il assure notamment l'abstraction et la communication entre les différents modules robotiques à travers un mécanisme spécial très riche [1].

4.2.2 Fonctions de base de ROS

Un grand nombre de fonctions fréquemment associées aux ROS, telles que les bibliothèques de navigation et des outils et d'algorithmes, ces fonctions donnent un ensemble puissant pour travailler avec ROS facilement, rappelons quelques outils ou algorithmes que le programmeur de ROS retrouvera souvent :

- **Gazebo** : Gazebo est un simulateur 3D, il est capable de créer un plan 3D sur votre ordinateur avec des robots, des obstacles et de nombreux autres objets.
- **Stage** : un simulateur 2D (pour créer un plan 2D).
- **ROS Graph:** une interface pour visualiser comment les choses sont connectées.
- **Rviz** : un système de visualisation 3D en temps réel (il n'inclut pas de moteur physique, comme Gazebo).
- **Rxgraph** : Il affiche l'architecture de nœud de l'application.
- **Le package tf** : permet de manipuler des coordonnées et des transformations (pour faire le modèle cinématique inverse et à manipuler des matrices).
- **Point Cloud Library :** reconstruction d'environnement 3D à partir de mesures d'un laser.
- **rxbag :** permet de visualiser graphiquement les données enregistrées dans les fichiers

bag.

4.3 Présentation du package

RTAB-Map (Real-Time Mapping Aspect-Based) Cartographie temps réel basée sur l'apparence de l'environnement est une approche RGB-D Graph-based SLAM basée sur un détecteur de fermeture de boucle incrémental bayésien. Le détecteur de fermeture de boucle calcule la probabilité qui détermine si l'image provient d'un nouveau lieu, ou bien si elle appartient à un lieu existant. Quand une hypothèse de fermeture de boucle est acceptée, une nouvelle contrainte est ajoutée au graphe de la carte, ensuite une optimisation du graph est appliquée pour minimiser les erreurs introduite au graphe. Une approche de gestion de la mémoire est adoptée pour limiter le nombre d'emplacements mémoire utilisés pour la détection de fermeture de boucle et de l'optimisation graphique.

4.3.1 Les nœuds de RTABmap

a. rtabmap

C'est le nœud principal de ce paquet. Ce nœud assure la construction progressive et l'optimisation de la carte lorsqu'une fermeture de boucle est détectée.

b. rtabmapviz

Ce nœud démarre l'interface de visualisation de RTABMap. Il a le même but que rviz mais avec des options spécifiques pour RTAB-Map.

c. Visual Odometry

Des trucs d'odométrie commun pour les nœuds rgbd_odometry et stereo_odometry. Ces nœuds enveloppent l'approche d'odométrie de RTAB-Map. L'approche est basée sur les caractéristiques visuelles présentes dans les images pour calculer l'odométrie. Quand une transformation ne peut pas être calculée, une transformation nulle est envoyée pour informer que l'odométrie n'est pas mise à jour ou perdu.

d. rgbd_odometry

Ce nœud enveloppe l'approche d'odométrie de RTAB-Map. En utilisant des images RGBD, l'odométrie est calculée en utilisant les points d'intérêts extraits des images RGB avec leurs informations de profondeur issues des images de profondeur. La mises en correspondances les images consécutives permet le calcule de transformation entre ces images.

e. stereo_odometry

Ce nœud enveloppe l'approche d'odométrie stéréo de RTAB-Map. En utilisant des images stéréo, l'odométrie est calculé à partir les caractéristiques visuelles extraites (points d'intérêt) des images gauches avec leur informations de profondeur calculée en trouvant les mêmes

caractéristiques sur les images droites. Utilisation de la fonction des correspondances, une approche de RANSAC calcule la transformation entre les images gauche consécutives.

f. camera

Un nœud permet l'acquisition d'images depuis une caméra USB (OpenCV est utilisé).

g. map_assembler

Ce nœud est pour des utilisations avancées seulement, il peut être utilisé pour l'optimisation du graphe en dehors du nœud de rtabmap.

4.4 Les bibliothèques utilisées

Diverses bibliothèques open source sont également utilisées pour fournir des fonctionnalités nécessaires dans ce travail :

4.4.1 Eigen

Librairie fournit des outils d'algèbre linéaire comme de calculs matriciels. Cette librairie est utilisée dans notre projet pour enregistrer et manipuler les matrices de transformation entre deux itérations.

4.4.2 OpenNI

La gestion de la camera Kinect et de ses fonctionnalités sont assurées grâce aux pilotes OpenNI nous pouvons donc utiliser ROS pour traiter et recevoir les informations capturées par la Kinect. Les images couleurs et de profondeur fournies par sont combinés avec les informations d'étalonnage pour former un nuage de points. Les pilotes OpenNI, développés principalement par Prime-Sense et Willow Garage, travaille avec d'autres logiciels utilisés dans ce projet comme PCL et Open CV

4.4.3 OpenCV

La librairie Open CV très utilisée dans le domaine du traitement des images. Elle propose des fonctions pour manipuler les images et les filtrer,

Open CV est utilisé dans ce projet pour le stockage et la manipulation des données d'image 2D. Il fournit également implémentations des détecteur / descripteur utilisés dans ce projet

4.4.4 PCL

La librairie Point Cloud Library (PCL) offre quant à elle des fonctions de traitement pour des données 3D. Il s'agit d'une bibliothèque dédiée au traitement de données type nuage de points (Point Cloud Librairy).

4.4.5 g2o

General Framework for Graph Optimisation (G2O) s'agit d'une bibliothèque open source, dédié aux méthodes de Graph-based SLAM, elle fournit des fonctionnalités pour la construction

graphique ainsi que l'optimisation.

4.4.6 FLANN

Fast Library for Approximate Nearest Neighbors (FLANN) est une bibliothèque qui contient une collection d'algorithmes pour la recherche rapide du plus proche voisin dans les grands ensembles de données de dimension élevées. Cette bibliothèques est utilisée dans ce projet pour faire la mise en correspondance les points en 2D dans chaque deux images successives.

Les résultats sont obtenus sur un ordinateur PC avec les spécifications suivantes

CPU : Intel Core i5

Mémoire : 4GB

Disk Dure : 500 GB

Système d'exploitation : Ubuntu 14.04

4.5 Comparaison des points d'intérêt

Dans cette étude, nous analyserons les algorithmes de détection/description des points d'intérêt par rapport à l'efficacité, la qualité et la robustesse.

Les trois méthodes de détection/description des points d'intérêts SIFT, SURF et ORB (décrite dans la section 2.2) sont comparées par apport aux nombres des points détectés et aussi au temps mis pour la détection et à la description.

Il y a plusieurs paramètres à prendre en compte lors de l'utilisation de chacune de ces méthodes, l'implémentation d'OpenCV fournit des valeurs par défaut qui correspondent aux documents originaux qui présente chacune de ces approches.

SIFT utilise trois couches dans chaque octave, le seuil de contraste, utilisé pour filtrer les points faibles dans les régions basses-contraste, est fixé à 0,04. Le seuil pour filtrer les points situés sur les arêtes est 10. Le sigma de la gaussienne appliquée à l'image d'entrée à la première octave est maintenue par défaut à 1,6.

SURF utilise deux couches dans chaque octave, avec un nombre total de 4 octaves; le seuil de détecteur Hessien utilisé a pour valeur 500, donc tous les points avec un hessien plus grandes que ça sont conservés.

ORB utilise 8 niveaux de la pyramide avec un facteur d'échelle de 1.2, facteur d'échelle peut améliorer résultat avec des valeurs plus proches de 1, mais la performance souffre et des

valeurs plus élevées entraînent des caractéristiques avec des scores pauvres. Un seuil pour la taille de la frontière où les caractéristiques ne sont pas détectées est réglé sur 31, la même valeur que la taille du patch utilisé par le descripteur BRIEF orientée.

Il y a beaucoup d'expériences menées afin d'évaluer les performances de chacune de ces méthodes en se basant dans notre analyse sur les points suivants :

1- La rapidité

Pour mesurer la vitesse de détection /description des points d'intérêts, le temps de traitement de l'image de chaque méthode est enregistrée à chaque itération, tel que le temps le plus faible correspond à la vitesse la plus élevée

2- Nombre de points d'intérêts

Dans cette expérience, le nombre de points d'intérêt pour chaque méthode de détection est enregistré. Le plus grand nombre de points est un inconvénient majeur dans la détection des points, car plus de points à apparier plus le temps à consommer.

3- Répétabilité

Elle est basée sur le calcul du nombre de points de la même image pour pouvoir déterminer si le même nombre des points est répété.

Dans cette section nous allons tester les performances de chaque algorithme sur une seule image d'une scène. Pour se faire nous allons esquisser trois scénarios .représentant les différentes conditions de l'environnement.

- Le premier scénario est fait sous des conditions de luminosité et d'orientation normale.
- Le deuxième scénario consiste à changer la luminosité de l'environnement.
- Le troisième scénario se déroule dans un environnement ou la luminosité et l'orientation de la camera sont changées.

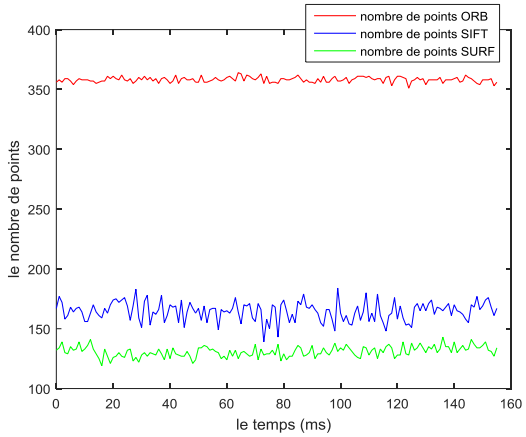
(1) (2) (3)

Figure 4-1: les différents scénarios

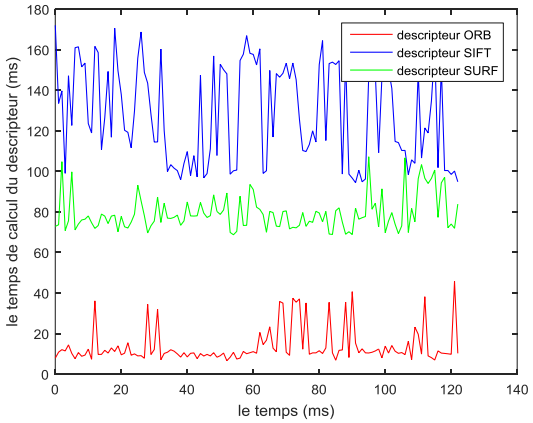
A chaque déroulement d'un scénario le nombre des points détectés et le temps mis pour la détection et la description sont calculés.

Les graphes suivants représentent les résultats obtenus :

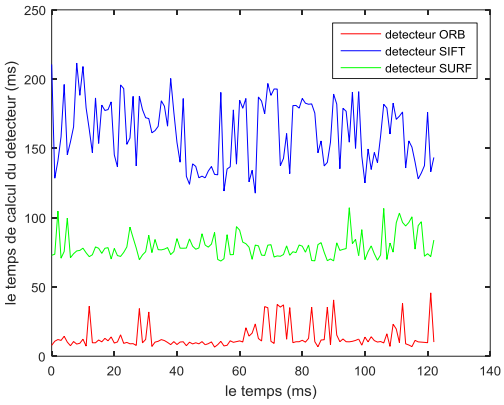
1er scénario



a : Le nombre des points d'intérêt extrait



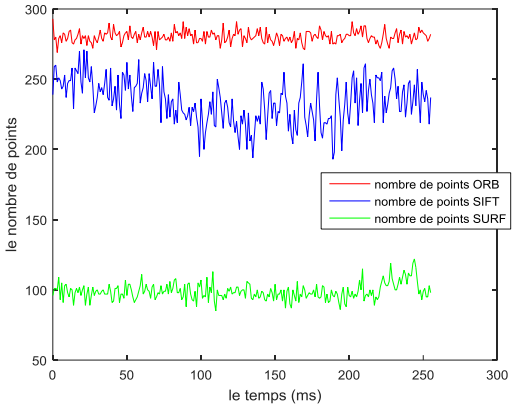
b : le temps de calcul du descripteur



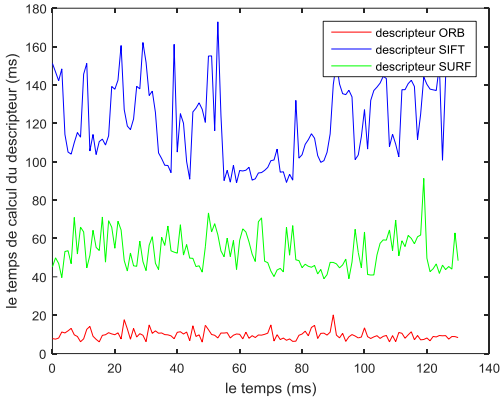
c : Le temps de calcul du détecteur

Figure 4-2: scénario 1

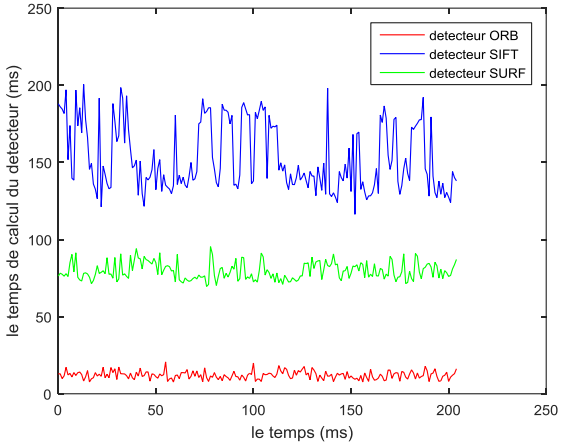
2ème scénario



a : Le nombre des points d'intérêt extrait



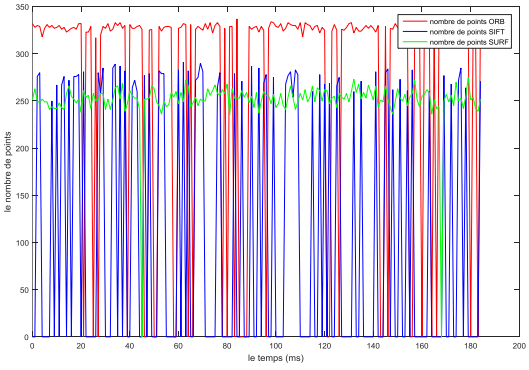
b : le temps de calcul du descripteur

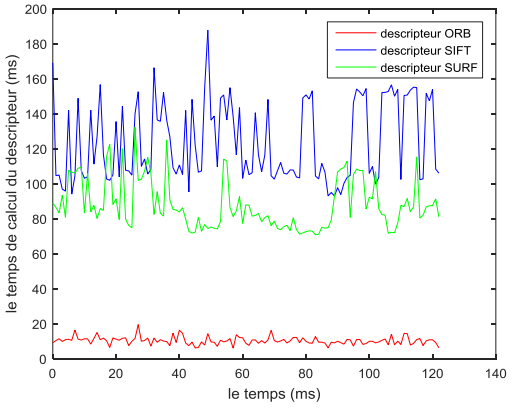


c : Le temps de calcul du détecteur

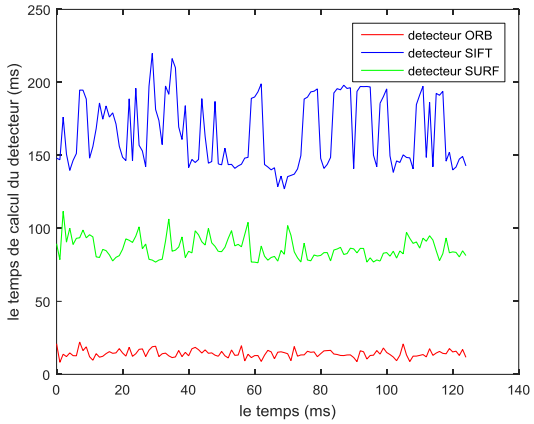
Figure 4-3: scénario 2

3ème scénario





a : Le nombre des points d'intérêt extrait **b** : le temps de calcul de descripteur



c : Le temps de calcul détecteur

Figure 4-4 : Scénario 3

Les tableaux suivants sont établis pour résumer les résultats donnés par les graphes:

1er scénario

Descripteur et détecteur	Nombre de points d'intérêt extraits	Le temps de calcul de détecteur (ms)	Le temps de calcul de descripteur (ms)
ORB	350-378	5.651-31.3499	6.036-17.9889
SIFT	310-346	117.667-211.627	94.413-172.016
SURF	177-213	68.713-133.247	50.6949-97.322

Tableau 4.1 : le nombre de points extraits, le temps de calcule détecteur et descripteur pour scénario 1

2ème scénario

Descripteur et détecteur	Nombre de points d'intérêt extraits	Le temps de calcul de détecteur (ms)	Le temps de calcul de descripteur (ms)
ORB	267-293	7.622-20.668	5.9741- 20.185
SIFT	193- 271	116.3210- 200.6	86.4928-172.913
SURF	83-122	67.112-95.391	38.265-91.3181

Tableau 4.2 : le nombre de points extraits, le temps de calcule détecteur et descripteur pour scénario 2

3ème scénario

Descripteur et détecteur	Nombre de points d'intérêt extraits	Le temps de calcul de détecteur (ms)	Le temps de calcul de descripteur (ms)
ORB	0-339	8.265-22.182	8.265-22.182
SIFT	0-291	127.033-219.9170	92.9289-187.71
SURF	0-275	73.8709-111.732	73.8709-111.732

Tableau 4.3 : le nombre de points extraits, le temps de calcule détecteur et descripteur pour scénario 3

Lors de l'analyse des algorithmes de détection/description des points d'intérêt par rapport à l'efficacité, la qualité et la robustesse. Les performances globales montrent que :

- l'algorithme d'ORB a une bonne performance par rapport à la vitesse car il met le moins de temps pour la détection et la description des points d'intérêt par rapport aux deux autres algorithmes.
- ORB extrait un nombre extrêmement élevé de points d'intérêt ce qui conduit a d'augmenter la durée du processus d'appariement.
- L'algorithme d'ORB à la plus petite plage de variation du nombre des points d'intérêt détecté cela montre que cet algorithme a la meilleure répétabilité.
- Encore une fois, ORB montre une très bonne performance par apport à la robustesse vue le nombre de points inchangé dans les trois cas, Cela montre aussi que ORB est insensible au changement de point de vue ou la luminosité.

- le nombre de points détectés par l'algorithme de SIFT change considérablement, Cela montre que cet algorithme est sensible aux changements de l'orientation et d'éclairage. Car un changement provoquera des nouveaux objets qui apparaissent dans l'image telle que les ombres.

4.5 Amélioration due à ICP

Une autre série de tests a été faite pour évaluer plus spécifiquement l'amélioration de l'estimation de pose due à ICP.

On considère les deux images suivantes :

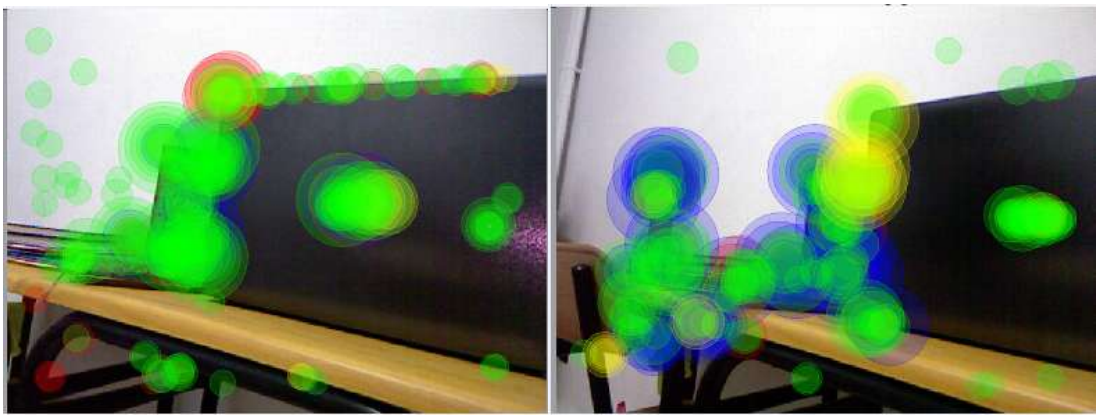


Figure 4-5: Deux images correspondent à une seule prise en changeant le point de vue

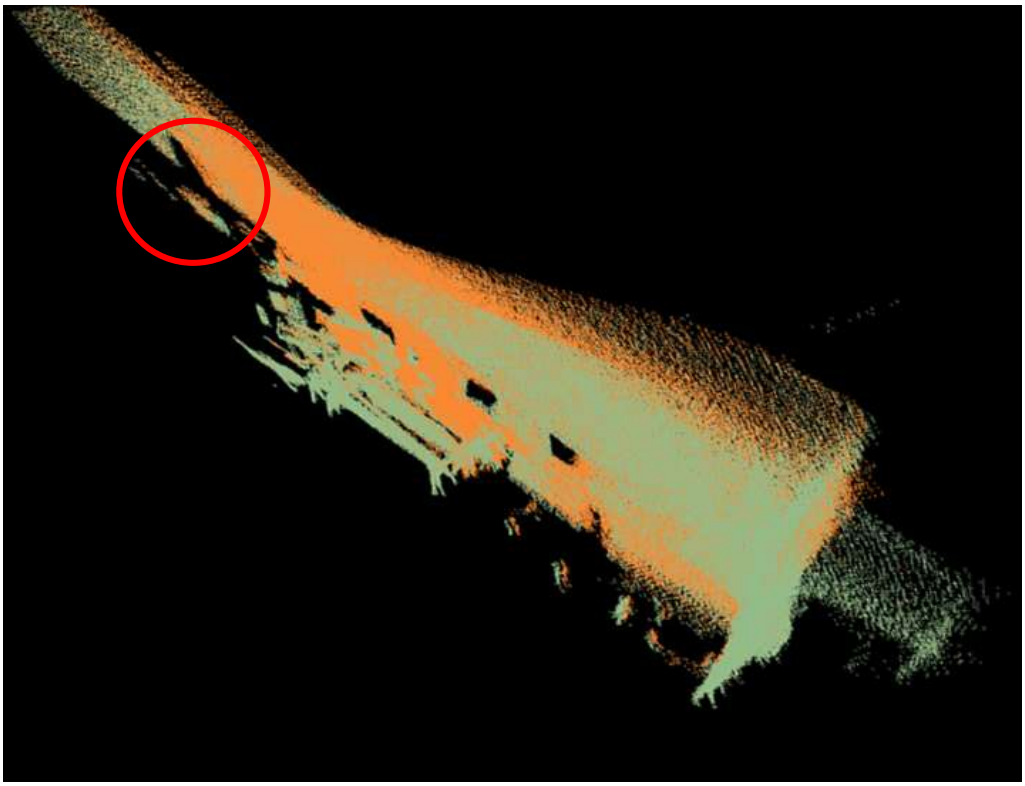


Figure 04-6: Recollage fait par RANSAC

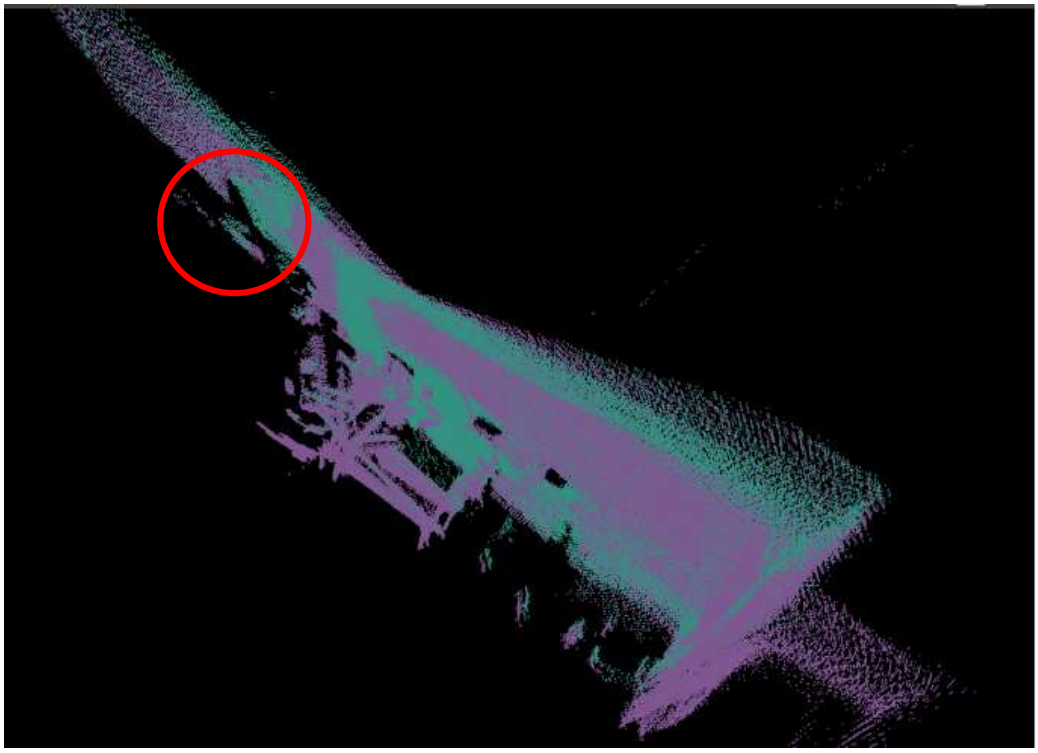


Figure 4-7: Recollage fait par RANSAC+ICP

Les résultats montrent qu'ICP permet de mieux recoller les nuages de points (pas de dédoublement de plans). Donc les algorithmes RANSAC +ICP permet d'estimer la transformation le plus correctement possible.

L'amélioration de la pose par ICP est significative mais dans certains cas il peut être intéressant de ne conserver que RANSAC pour gagner en vitesse d'exécution (dans le cas de scène simple).

4.6 Comparaison des deux algorithmes d'optimisation

Cette section présente une comparaison des deux méthodes d'optimisation graphique, en se basant sur le calcul d'erreur d'optimisation de chaque méthode.

Le scénario :

Le robot fait la cartographie d'une scène deux fois en suivant la même trajectoire (pseudo circulaire) et en changeant à chaque fois l'approche d'optimisation, le graphe correspond à la première expérience est optimisé par la méthode du TORO et le deuxième par la méthode de g2o

La figure suivante présente le graphe avant et après l'optimisation en utilisant les deux approches.

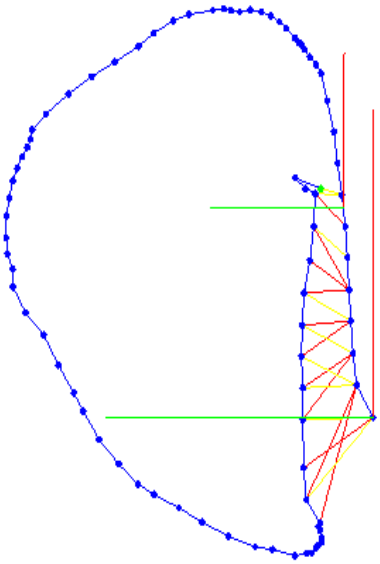


Figure 4-8 : Le graphe avant l'optimisation

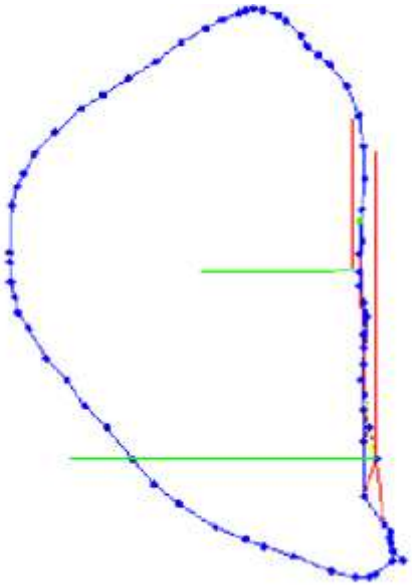


Figure 4-9: Le graphe après l'optimisation par la méthode du g2o

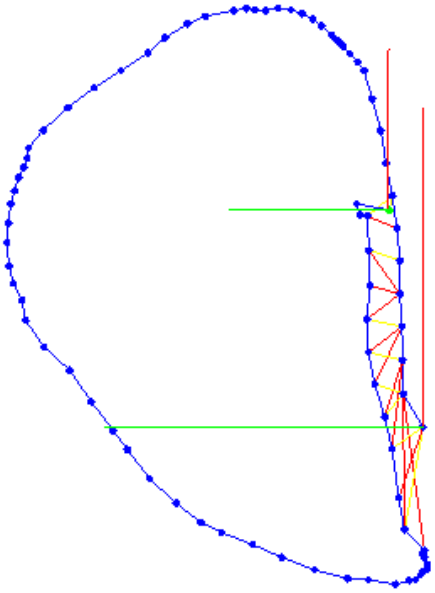


Figure 4-10: Le graphe après l'Optimisation par la méthode du TORO

On constate que les graphes ne sont pas optimisés de la même manière car chaque méthode a son propre algorithme.

La figure suivante représente le calcul de l'erreur d'optimisation en fonction du temps :

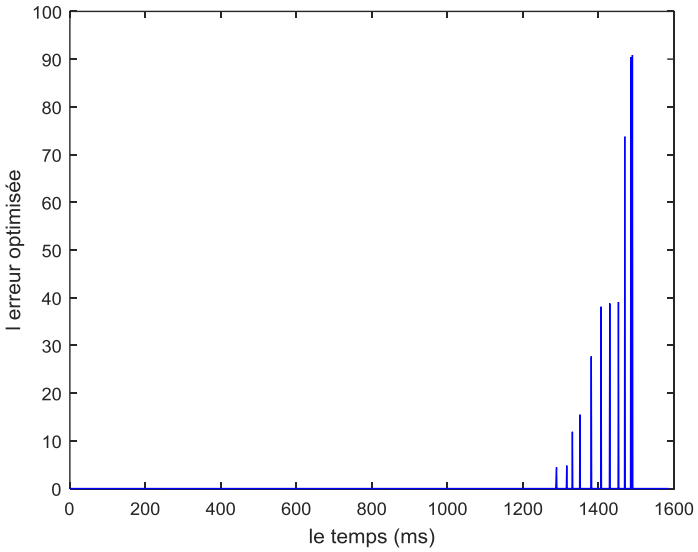


Figure 4-11 : L'erreur optimisée par g2o

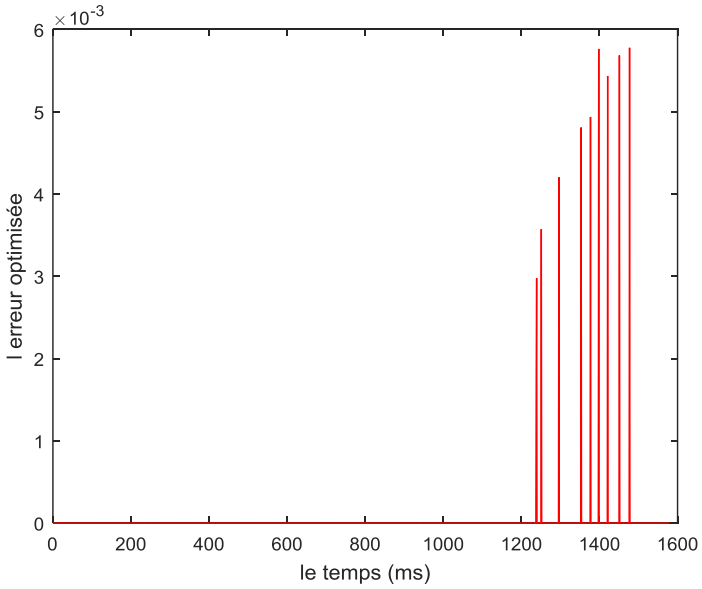


Figure 4-12: L'erreur optimisée par la méthode du TORO

Lorsque le robot revient à sa position initiale une fermeture de boucle est détectée, on constate a partir de ces deux graphes que l'erreur optimisée en utilisant la méthode de g2o est beaucoup plus grande que celle optimisée par l'autre méthode, on conclue donc que la méthode d'optimisation de g2o est plus fiable.

4.7 L'implémentation d'un module SLAM Visuel

Les résultats faits précédemment nous permettent d'établir notre choix par apport aux différents algorithmes étudiés.

On lance le nœud RTABMap pour faire la cartographie et la localisation simultanées et on mesure en parallèle le temps de traitement de chaque algorithme.

La procédure est faite comme suit :

- l'acquisition des données à l'aide du capteur Kinect
- l'extraction et la détection des points en utilisant le détecteur/descripteur ORB
- le calcul de la transformation avec RANSAC et à l'affiner par ICP
- calculer des poses initiales et les traduit en nœuds et en contraintes.
- détecter les fermetures de boucle et insérer les contraintes correspondantes dans le graphe
- optimiser le graphe avec g2o et recalculer les nouvelles poses du robot.
- Reconstruction globale de la scène

Nous allons prouver la procédure précédente, en faisant la cartographie d'un hall au CDTA.

Le graphe suivant représente le temps de calcul de la transformation avec RANSAC + ICP :

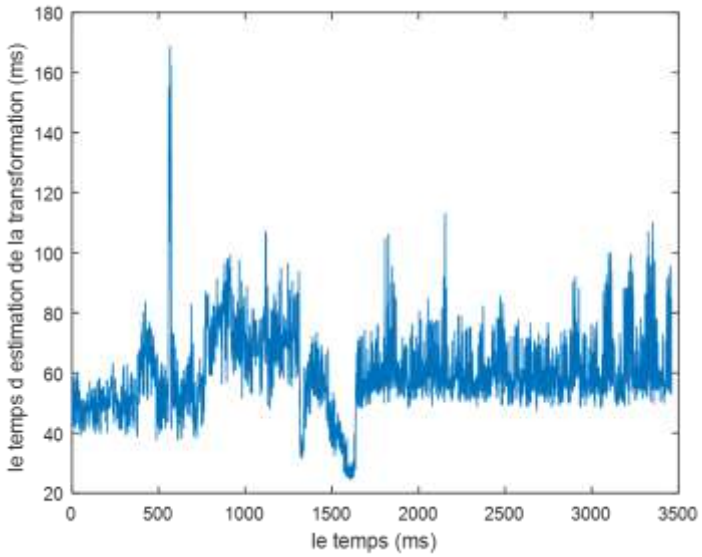


Figure 4-13: Le temps d'estimation de la transformation

En général le temps de calcul de la transformation est entre 40 et 80 ms et cela est principalement dû aux nombres appariés.

Après le calcul de la transformation une représentation des données sous forme de graphe est faite en modélisant les poses du robot sous forme des nœuds qui sont insérées dans un graphe et reliées entre elles par des contraintes correspondant aux déplacements estimés.
Bien que la précision des méthodes RANSAC et ICP combinées soit bonne, on constate une dérive de la position au cours du temps due à l'accumulation des erreurs de mesure.
Après un long parcours dans des zones inconnues de l'environnement, le robot est revenu sur une position passée, cela offre la possibilité d'accroître la précision et la cohérence de l'estimation. C'est ce qu'on appelle la détection de la fermeture de boucle.

Lorsqu'une nouvelle image est acquise, la détection de fermeture de boucle permet de déterminer si elle provient d'un nouveau lieu, ou bien si elle appartient à un lieu existant Pratiquement on estime la ressemblance entre les points d'intérêt détectés dans l'image courante et dans les images précédentes. C'est ce que montre la figure suivante :

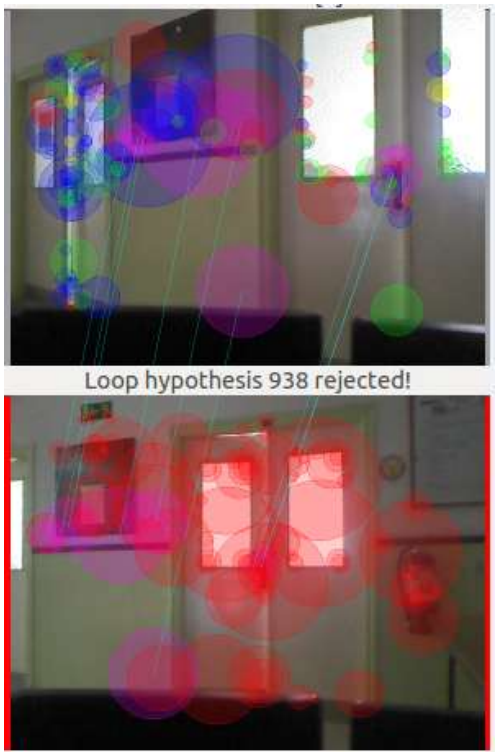


Figure 4-14: Détection de la fermeture de boucle

Si la fermeture de boucle est détectée des contraintes additionnelles sont ajoutées au graphe reliant ce nœud à celui correspondant à l'observation précédente de la zone, On peut voir ça sur les figures suivantes : La première figure représente la trajectoire suivit par le robot, La deuxième figure représente le graphe construit à partir des poses du robot et la troisième explique l'effet de la détection de la fermeture de boucle ou des nouvelles contraintes sont ajoutées au graphe

Figure 4-15: La trajectoire du robot

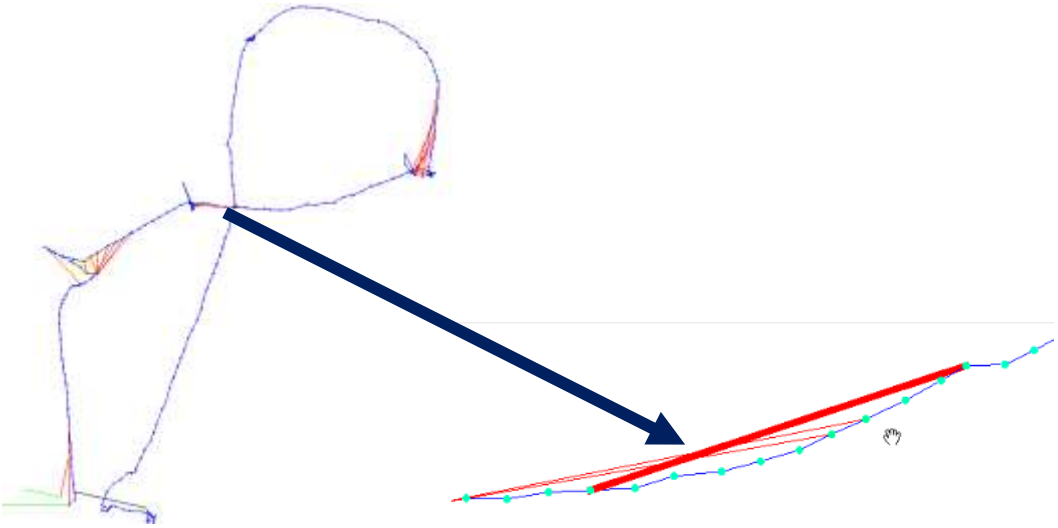


Figure 4-16: Les contraintes ajoutées après la détection de la fermeture de boucle

La flèche corresponde à l'ajout d'une contrainte entre deux nœuds lors de la détection de la fermeture de boucle

Après la construction du graphe il faut calculer la nouvelle configuration des nœuds du graphe Diverses méthodes sont disponibles sur g2o, aussi bien pour le processus d'optimisation, et le problème de résolution. La méthode utilisée ici est celle de Levenberg-Marquardt avec un solveur Cholmod linéaire.

Les graphes suivants donnent respectivement : l'erreur optimisée et le temps total de traitement.

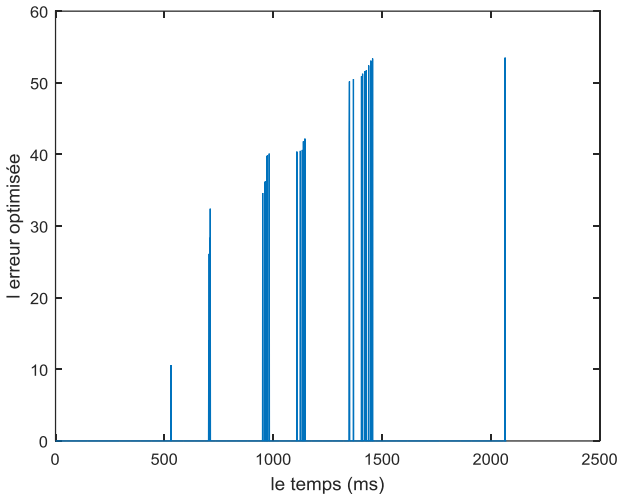


Figure 4-17: L'erreur optimisée en fonction du temps

On constate que à chaque détection d'une fermeture de boucle une optimisation est introduite afin de reconfigurer les nœuds du graphe pour réduire considérablement l'erreur accumulée depuis le départ ce qui permet de rétablir une estimation correcte de la pose.

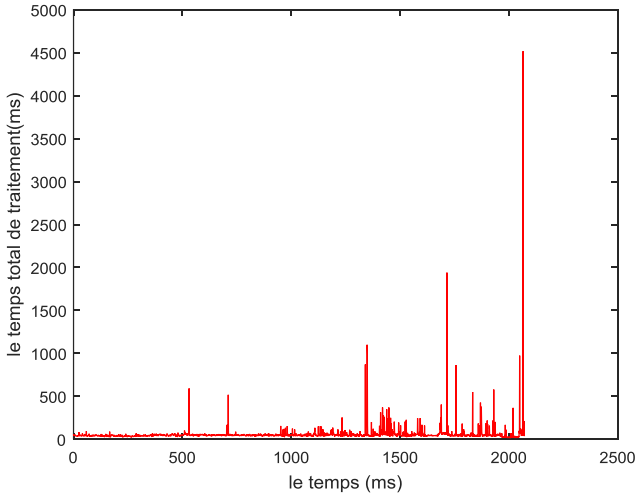
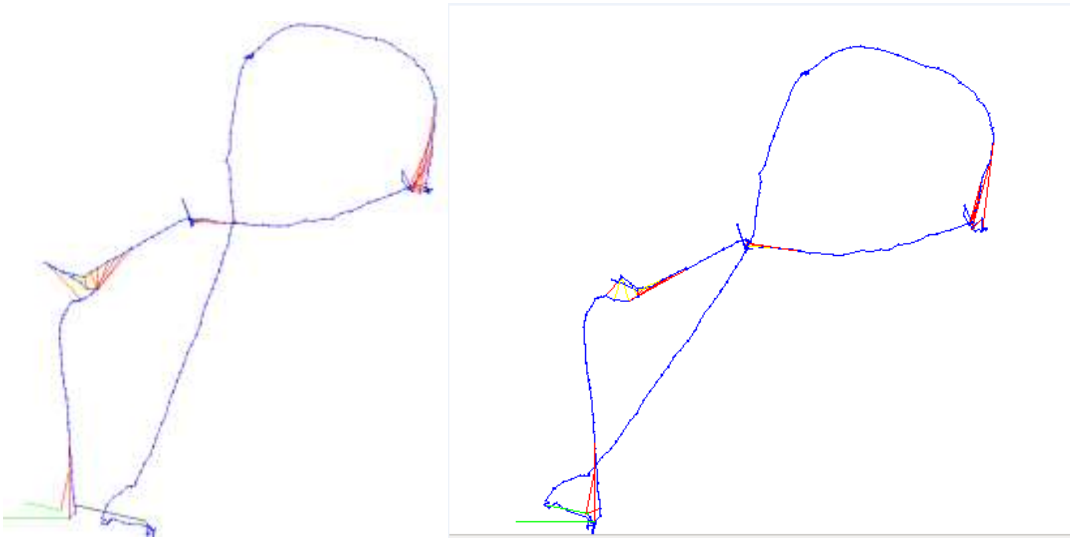


Figure 4-18: le temps total du traitement

le graphe montre que a chaque detection de la fermeture de boucle le temps de traitement augmente a cause du temps mis a l'optimisation d'erreur .

La figure suivante montre le graphe avant et après optimisation :



(a) **(b)**

Figure 4-19: (a) le graphe non-optimisé, (b) le graphe optimisé

Les figures suivantes montrent le déroulement du processus de la cartographie du hall au CDTA

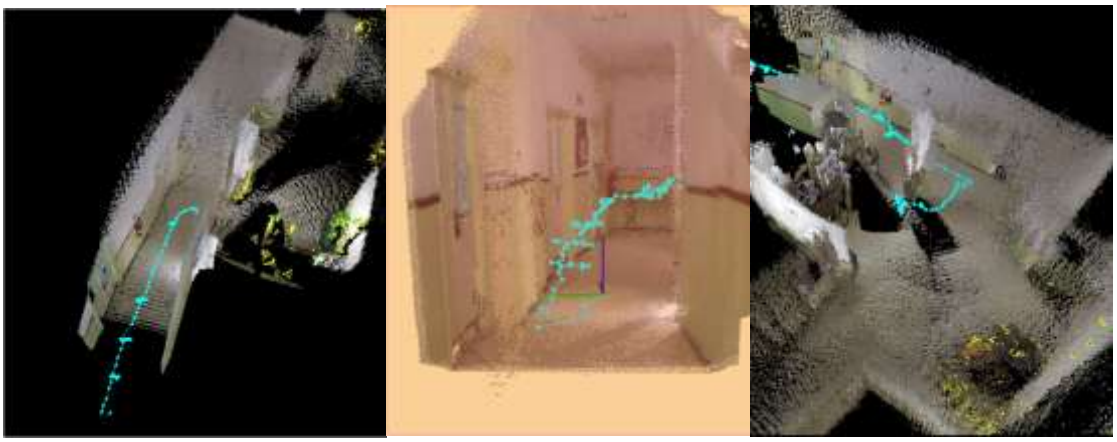


Figure 4-20: La carte vue sous différentes angles

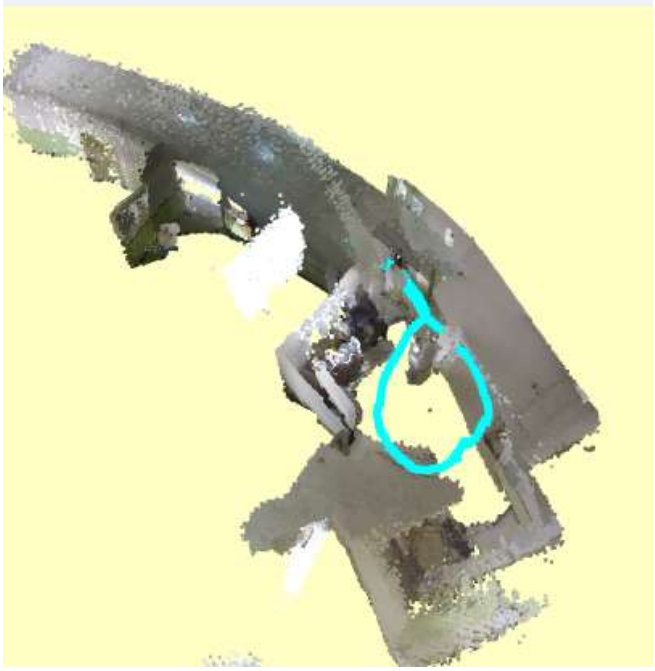


Figure 4-21: La carte finale

4.8 CONCLUSION

Dans ce chapitre on a représenté les expériences menées pour confirmer notre choix par apport aux algorithmes utilisés pour la résolution du problème de SLAM ,Les résultats du test ont montré que le détecteur ORB montre une très bonne performance par apport à la robustesse et la rapidité de l'extraction et la description des points par la suite on a marqué l'avantage d'ICP par rapport à RANSAC est qu'il permet d'estimer les mouvements avec une grande finesse et enfin on a expliqué les bases théoriques évoquées dans le chapitre précédent du SLAM Visuel.

Conclusion générale

Nous avons présenté dans ce projet les tâches de localisation et de cartographie, regroupées sous le terme de SLAM qui constitue un pilier fondamental du fonctionnement autonome d'un véhicule mobile robotisé.

L'objectif de notre travail était d'implémenter un module V-SLAM pour le robot B21r équipé d'une caméra RGB-D « Kinect Microsoft » pour la localisation et la cartographie d'un environnement réel, ce module est mis en œuvre de façon modulaire sous l'environnement ROS pour permettre facilement échanger et tester des approches alternatives.

Dans les travaux développés au cours de ce projet, nous nous sommes intéressés aux SLAM 3D basé sur la vision d'un environnement intérieur. Nous avons commencé dans le chapitre 1 par définir la problématique de la localisation et cartographie simultanées et son importance pour l'autonomie du robot mobile puis on est passé à décrire rapidement des différents types de robots et aussi les différentes types de capteur.
Nous avons ensuite présenté en chapitre 2, les algorithmes qui peuvent être adopté afin de résoudre le problème de SLAM.

Dans le chapitre 3 nous avons présenté et détaillé le squelette de notre algorithme de V-SLAM.

Enfin dans le chapitre 4, après une analyse des différents algorithmes à partir des expériences menées on a pu décider sur les algorithmes qui donnent les meilleures performances.

On a adopté l'algorithme de ORB pour l'extraction des points d'intérêt vue sa vitesse et sa robustesse après l'extraction des points d'intérêt une transformation entre deux nuages de points est calculer pour cette étape on a combiné les deux algorithmes de RANSAC et ICP.

Pour pouvoir estimer la transformation avec une grande finesse. Une fois les transformations soient calculées un graphe est initialisé, la détection de fermeture de boucle sert à insérer des nouvelles contraintes sur le graphe pour améliorer la robustesse du SLAM .pour la dernière étape d'optimisation du graphe on a adopté la méthode de g2o vue sa fiabilité par apport à la méthode du TORO.

PERSPECTIVES

Comme perspective de notre projet, nous proposons de développer les points suivants :

- fixer le seuil de l'algorithme de RANSAC généralement se fait de manière empirique. Il existe de nombreuses variantes du RANSAC qui tentent à remédier à ce problème, comme la méthode MLESAC qui utilise les M-estimateurs pour fixer les paramètres d'une manière robuste.
- Ajouter des capteurs supplémentaires: dans ce travail, le seul capteur utilisé était le capteur Kinect fournissant des données RGB-D. Une possibilité serait d'utiliser d'autres capteurs et les combiner pour obtenir un meilleur résultat.
- Implémentation de GPU : permet une extraction optimale des points d'intérêt ce qui va réduire le temps de calcul considérablement, Diverses mises en œuvre GPU existent pour SIFT et SURF. cette solution tire profit du traitement de l'information en parallèle sur les plates-formes spécifiques, tels que CUDA ™ pour les cartes graphiques NVIDIA.

Bibliographie

[1] Oussama El Hamzaoui, Localisation et cartographie simultanées pour un robot mobile équipé d'un laser à balayage : CoreSLAM, l'école nationale supérieure des mines de Paris, 2012.

[2] P. Maybeck, ''The kalman filter: An introduction to concepts'',Autonomous Robot Vehicles, pp. 194 204, 1990.

[3] VIRGILE HÖGMAN, Building a 3D map from RGB-D sensors, Computer Vision and Active Perception Laboratory Royal Institute of Technology (KTH), Stockholm, Sweden.

[4] Robert Holmberg, "Design and Development of Powered-Castor Holonomic Mobile Robots", Stanford University, 2000.

[5] A.J. Davison. Real-time Simultaneous Localisation and Mapping with a SingleCamera.InProceedingsoftheNinthIEEEInternationalConference on Computer Vision, volume 2, pages 1403–1410, 2003.

[6] M. Renaud BARATE «Apprentissage de fonctions visuelles pour un robot mobile par programmation génétique », L'UNIVERSITÉ PIERRE ET MARIE CURIE, 2008.

[7] A.Maxwell « Indoor mobile robots», IROBOT.

[8] O.Innocent, H.Huosheng « An interactive user interface to the mobility object manager for RWI robots », University of Essex United Kingdom.

[9] Baba, A. (2007). Cartographie de l'Environnement et Suivi Simultané de Cibles Dynamiques par Un Robot Mobile. Thèse de doctorat, Université de Toulouse - Paul Sabatier. (document), 2.4.3, 2.5.

[10] Bertrand PECUCHET «Kinect for Xbox 360», L'UNIVERSITÉ de L'ille.

[11] B.Bayle « Robotique Mobile », Télécom Physique Université de Strasbourg, 2010-2011.

[12] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," International journal of computer vision, vol. 60, no. 2, pp. 91–110, 2004.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in Computer Vision (ICCV), 2011 IEEE International Conference on, pp. 2564–2571, IEEE, 2011.

[14] C. Harris and M. Stephens, “A combined corner and edge detector.” in Alvey vision conference, vol. 15, p. 50, Manchester, UK, 1988.

[15] E. Rosten, R. Porter, and T. Drummond, “Faster and better: A machine learning approach to corner detection,” Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 32, no. 1, pp. 105–119, 2010.

[16] Herbert Bay, Tinne Tuytelaars et Luc Van Gool, « SURF: Speeded Up Robust Features », Proceedings of the European Conference on Computer Vision, 2006, p. 404-417.

[http://tcts.fpms.ac.be/publications/tfes/2012/sp.pd.

[17] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D Range Image Features for Object Recognition. In Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJInt.Conf. on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 2010.

[18] Leutenegger, S.; Chli, M.; Siegwart, R.Y. BRISK: Binary robust invariant scalable keypoints. In Proceedings of the 2011 IEEE International Conference on Computer Vision (ICCV), Barcelona, Spain, 6–13 November 2011; pp. 2548–2555.

[19] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6):381–395, 1981.

[20] Romain Drouilly, SLAM Visuel 3D pour robot mobile autonome, Master de Sciences Mention « Imagerie, Robotique et Ingénierie pour le Vivant », Septembre 2011.

[21] J.Laplace « Présentation de robot operating system », Club des développeurs et TI pro, décembre 2011.

[22] P.J. Besl and N.D McKay, "A Method for Registration of 3D Shapes", Proc. of IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 2, pp.239-256, 1992

[24] E. Olson, J.J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Int. Conf. On Robotics & Automation*, pages 2262–2269, 2006.

[25] A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent (Giorgio Grisetti Cyrill Stachniss Slawomir Grzonka Wolfram Burgard) *University of Freiburg, Department of Computer Science, 79110 Freiburg, Germany.*

[26] Rainer Kuemmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard g2o: A General Framework for Graph Optimization IEEE International Conference on Robotics and Automation (ICRA), 2011.

[27] Calonder, M.; Lepetit, V.; Strecha, C.; Fua, P. BRIEF: Binary robust independent

elementary features. In Computer Vision—ECCV 2010; Daniilidis, K., Maragos, P., Paragios, N., Eds.; Springer Heidelberg: Berlin, Germany, 2010; Volume 6314, pp. 778–792.

[28] G. Klein and D. Murray, “Parallel tracking and mapping for smallar workspaces,” in Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on, pp. 225–234, IEEE, 2007.

[29] S. Thrun et al., « Robotic mapping : A survey », Exploring artificial intelligence in the new millennium, pp. 1-35, 2002.

[30] S. Thrun., W. Burgard, D. Fox., Probabilistic Robotics, MIT Press, 2005.

[31] S. Huang, G. Dissanayake. Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM, In IEEE Trans. on Robotics, 2(5), Oct. 2007.

[32] S. Thrun, D. Fox, W. Bugard, F. Dellaert, Robust Monte Carlo Localization for Mobile Robots, In Artificial Inteligence, 128, 2001.

[33] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, In AAAI National Conference on Artificial Intelligence, 2002.

[34] S. Thrun, M. Montemerlo. The GraphSLAM Algorithm With Applications to Large-Scale Mapping of Urban Structures, In Proc. of the Int. Journal on Robotics Research, 2005.

[35] S. Kohlbrecher, J. Meyer, O. Von Stryk, U. Klingauf. A Flexible and Scalable SLAM System with Full 3D Motion Estimation, In the Int. Symp. on Safety, Security and Rescue Robotics (SSRR), Nov. 2011.

[36] B. Steux, O. El Hamzaoui. tinySLAM: A SLAM algorithm in less than 200 lines C-language program, In Proc. of the Int. Conf. on Control Automation Robotics & Vision (ICARCV), Dec. 2010.

[37] L. Carlone, R. Aragues, J.A. Castellanos, and B. Bona. A linear approximation for graph-based simultaneous localization and mapping, In Proc. of the Int. Conf. Robotics: Science and Systems, 2011.

[38] N. Tomatis, I. Nourbakhsh, and R. Siegwart. Hybrid simultaneous localization and map building : a natural integration of topological and metric. Robotics and Autonomous Systems, 44 :3–14, 2003.

[39] D. Dufourd. Des cartes combinatoires pour la construction automatique de modèles d’environnement par un robot mobile. PhD thesis, Institut National Polytechnique de Toulouse, 2005.

[40] R. Vincent, B. Limketkai, M. Eriksen. Comparison of indoor robot localization techniques in the absence of GPS, In Proc. of SPIE: Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV of Defense, Security, and Sensing Symposium, April 2010.

Référence

[1] http://wiki.ros.org/fr

[2] http://blog.photographies-naturelles.fr/wiki-Robot_Operating_System.html

Annexe A

Kinect

Le capteur Kinect est une barre horizontale reliée à une petite base avec un pivot motorisé, conçu pour être placé au-dessus ou en dessous de l'affichage vidéo (téléviseur, écran d'un ordinateur). Le dispositif comporte une caméra RGB, un capteur de profondeur constitué par un projecteur laser et une caméra infrarouge (IR) et un microphone.

A.1 Description de la Kinect

a. Le microphone

Le capteur 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 parasites grâce à un traitement de données efficace.

b. La caméra RGB

La première des deux caméras embarquées dans la technologie Kinect est une caméra couleur RGB « standard » avec un capteur de type CMOS. Elle se situe au centre de la barre horizontale (voir la figure 1.11). Elle permet une prise d'image avec une fréquence de 30 Hz, en couleur 32 bits et en résolution VGA de 640 × 480 pixels.

Le système RGB est simple, il combine les 3 couleurs primaires tel que le rouge, le vert et le bleu, suivant l'association de ces 3 couleurs on peut obtenir une image blanche ou un panel infini de couleur qui se situe dans le spectre visible de l'œil humain (cf. figure A.1).

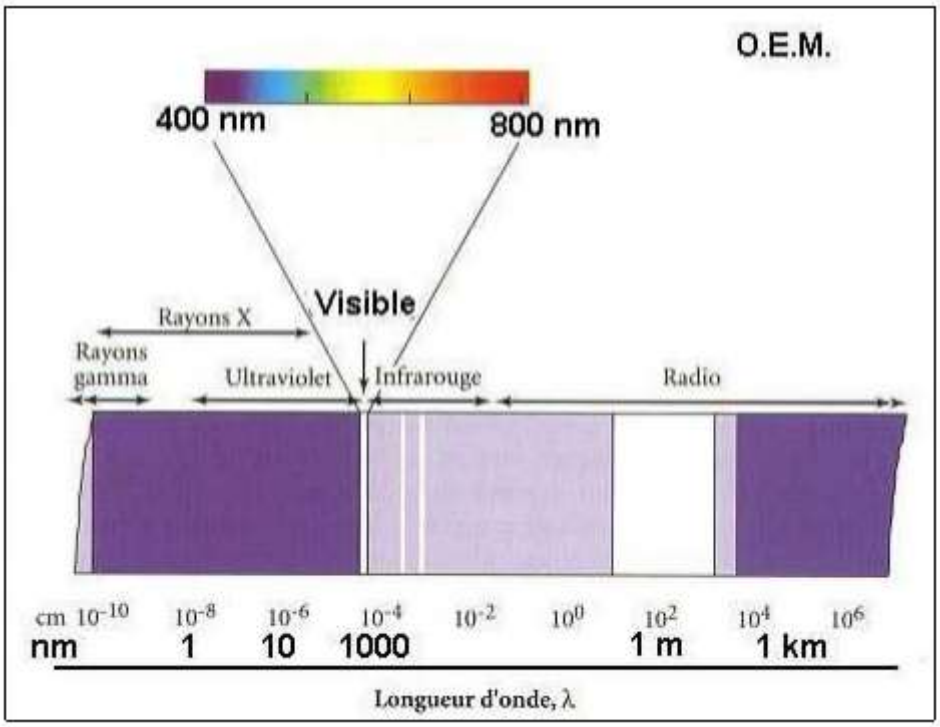


Figure A.1 : Spectre de la lumière

Ensuite nous pouvons parler du capteur photographique de type CMOS utilisé dans la Kinect. Ces capteurs sont le résultat de l'intégration de cellules composées d'une photodiode et d'une logique d'amplification puis d'obturation. Ils sont complexes à fabriquer mais sont produits selon des techniques classiques de micro-électronique et de ce fait peuvent avoir des résolutions très importantes (jusqu'à 24 mégapixels).

Le capteur CMOS se compose d'une matrice de cellules photosensibles qui conservent leur charge et les transfèrent elles-mêmes au convertisseur, là ou d'autres capteurs comme le CCD se compose d'une matrice de cellules photosensibles qui transfèrent la charge vers un collecteur qui transfère à son tour l'ensemble des charges vers le convertisseur.

Le capteur CMOS a l'avantage de ne pas transmettre une simple charge. Son photosite intègre un amplificateur de tension lui permettant de convertir lui-même la charge électrique reçue.

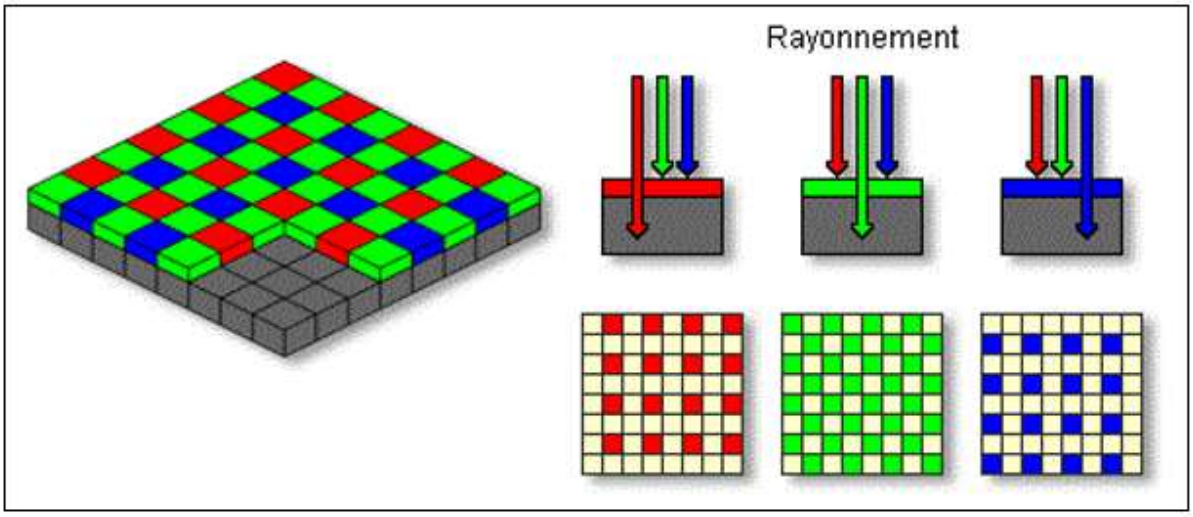


Figure A.2 : Capteur CMOS recouvert d'une grille photosensible de Bayer

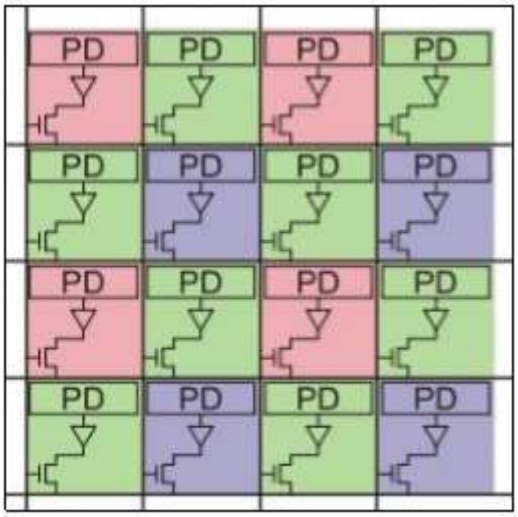


Figure A.3 : Surface d'un capteur CMOS

Les capteurs CMOS fonctionnent par champ de photodiode (PD) disposé en grille de Bayer (cf. figure A.2), chacune étant sensible à une seule des couleurs primaires (rouge, vert, bleu). C'est un champ de « 0 » ou « 1 » qui recouvre alors le capteur. C'est donc une combinaison des valeurs de chaque photodiode qui permet d'obtenir la couleur estimée de chaque pixel [10].

Le schéma suivant résume le principe du système RGB et les processus pour l'acquisition d'une image avec un haut contraste et de très bonnes couleurs.

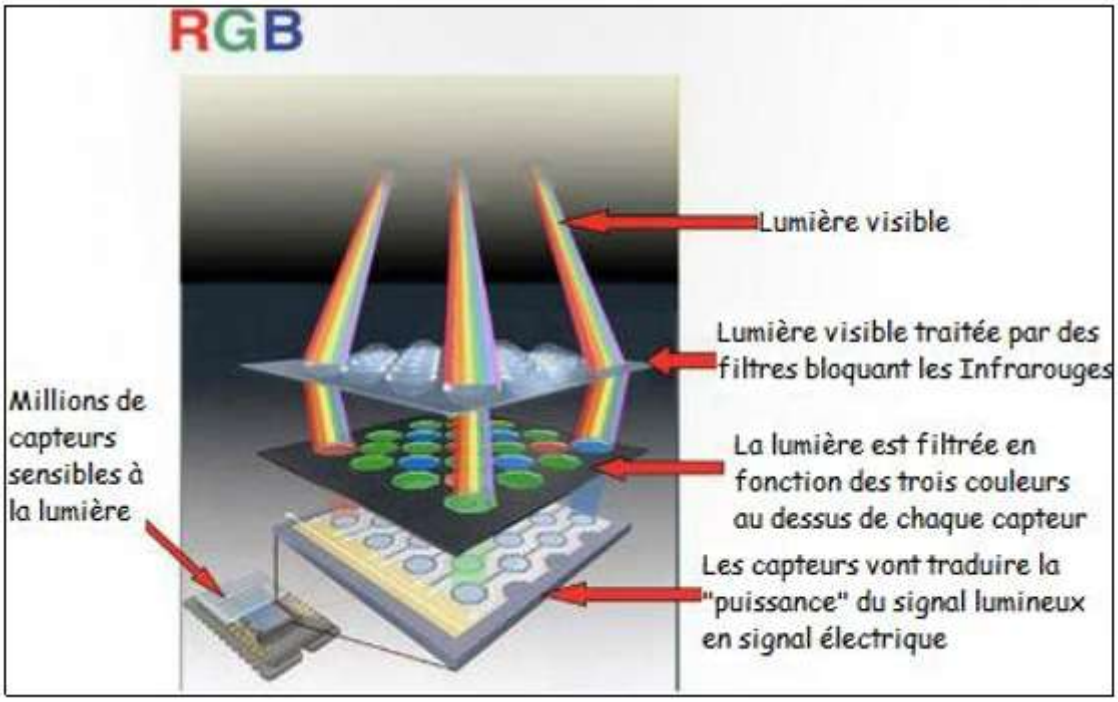


Figure A.4 : Processus du système RGB

c. Le capteur de profondeur

Un capteur CMOS infrarouge fonctionne quasiment de la même manière qu'un capteur CMOS RGB sauf qu'il laisse uniquement passer les infrarouges grâce à un filtre. Voyons comment le dispositif infrarouge de Kinect fonctionne.

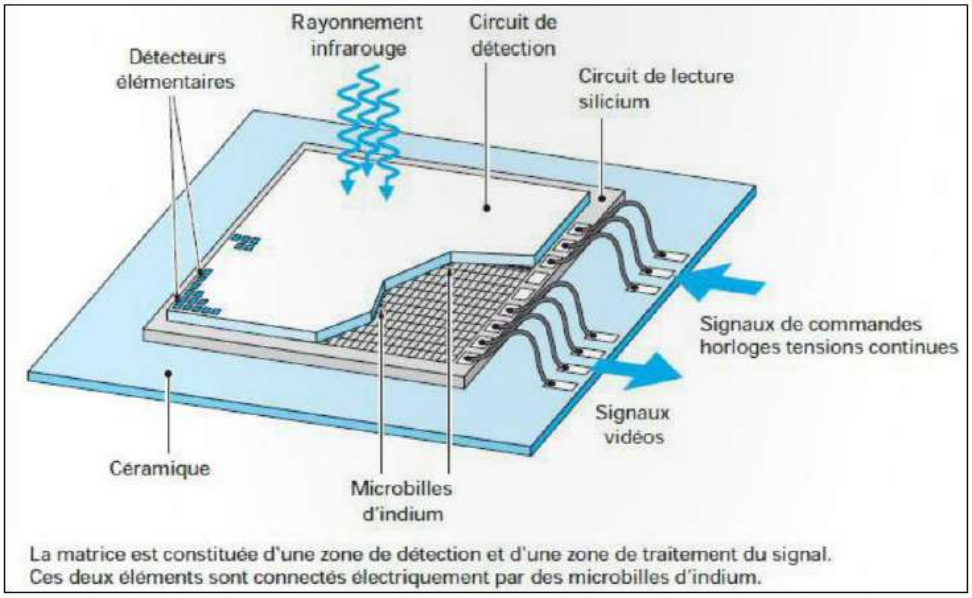


Figure A.5 : Capteur CMOS infrarouge

Actuellement, la majorité des matrices de détecteurs infrarouges est conçue comme sur la

figure A.5, c'est-à-dire en suivant une architecture dite hybride constituée de deux éléments distincts :

- Une zone de détection constituée de M x N détecteurs élémentaires dont le rôle consiste à transformer le flux de photons infrarouge incident en une image « électrique ».
- Une zone de traitement du signal obtenue à l'aide d'un circuit intégré réalisé en silicium. Ce circuit, appelé circuit de lecture sert à collecter et traiter le signal de chaque détecteur et à le convertir dans un format exploitable par l'électronique de mesure (en générale une tension).

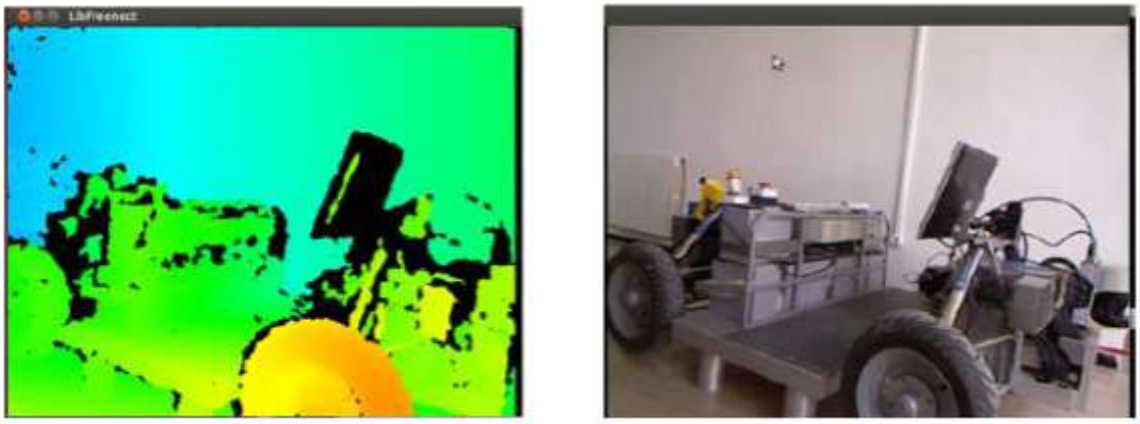
Cette architecture offre deux avantages :

- Elle permet d'utiliser le silicium, le matériau le mieux adapté pour la détection.
- Elle permet d'atteindre des facteurs de remplissage optique proches de 100%.

La technique utilisée ici pour connecter point à point la mosaïque de détecteurs élémentaires au circuit de lecture en silicium est un contact électrique par l'intermédiaire de microbilles d'indium [10].

En outre, il est possible de cartographier avec précision tout objet éloigné de 1 à 2 mètres de la caméra jusqu'à environ 4-5 mètres. Au-delà de 5 mètres, le rayonnement IR réfléchi devient trop faible pour être mesurable avec précision. Pour tout objet dont la distance est inférieure à 1 mètre, le phénomène inverse est observé et le signal devient totalement saturé. Par exemple dans l'image de profondeur de la figure A.6, les zones en rouge sont des régions pour lesquelles la Kinect n'arrive pas à estimer la distance à cause d'une mauvaise réflectivité due à la structure ou la matière de certains objets comme le verre. Dans ce cas la valeur de la distance des points par rapport à la caméra est nulle. On peut donc en déduire de cette image que les points les plus proches de la camera (qui tendent vers 0) sont marqués en rouge et les plus éloignées sont marqués en bleue-ciel.

Les autres couleurs (jaune et vert) sont donc des valeurs intermédiaires. Aussi l'avantage de l'utilisation de l'infrarouge est de pouvoir récupérer une carte de profondeur de la scène indépendamment des conditions de luminosité sans interférence avec la lumière ambiante (lumière du visible).



(a) **(b)**

Figure A.6 : (a) Image d'une carte de profondeur (b) Image couleur

A.2 Caractéristiques de la Kinect

Ci-dessous sont présentées les différentes caractéristiques techniques de la Kinect :

- Capteurs :

- 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 grâce au socle

- 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 16 bits à 30 FPS (images par seconde) pour le capteur infrarouge.
- 640 × 480 32 bits à 30 FPS pour le capteur RGB.
- Audio 16 bits à 16 kHz grâce aux 4 microphones.

- 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 :

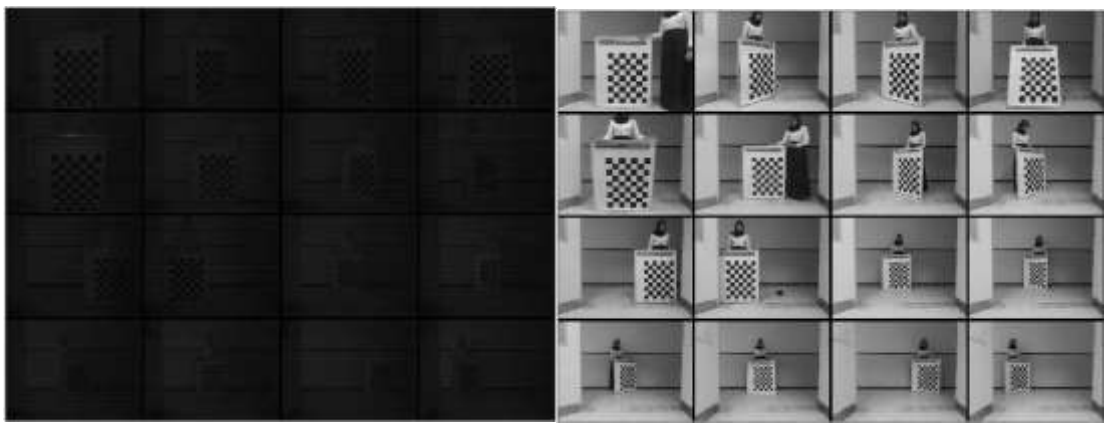
- Chat vocal Xbox Live et chat vocal dans les jeux (nécessite un compte Xbox Live Gold).
- Suppression de l'écho.

- Reconnaissance vocale multilingue.

A.3 Calibrage de la Kinect

Le capteur Kinect est une caméra RGB-D et possède de ce fait deux capteurs optiques : une lentille infrarouge et une lentille couleur. De la même manière que pour un appareil photo classique, ces lentilles peuvent être affectées par des distorsions géométriques plus ou moins marquées.

Pour que la Kinect soit prête à l'utilisation, elle doit être étalonnée (calibrée). Ceci se fait en deux phases : la première consiste à calibrer les deux caméras toutes seules (RGB et IR) pour obtenir les paramètres intrinsèques et la deuxième phase consiste à déterminer les paramètres extrinsèques, en considérant les deux caméras comme un banc stéréoscopique dans lequel nous nous intéressons à la distance entre les deux caméras. La procédure consiste à déterminer les différentes erreurs fournies par les défauts des lentilles. Dans ce cas, nous devons aligner le tableau des pixels de chaque caméra avec son axe optique. En utilisant Toolboox calib-gui, nous prenons des différentes vues d'une mire dans lesquelles les images IR sont prises avec un couvert du projecteur IR (figure A.7).



(a) (b)

Figure A.7 : Différentes vues d'une mire de calibrage (a) images IR (b) images RGB.

A.2 Calibrage des deux caméras IR et RGB

Le modèle sténopé modélise une caméra idéale (Projection Perspective Simple). Le système optique induit des distorsions géométriques qui affectent la projection des points. Ainsi, un point projeté dans une image ne correspond pas au modèle sténopé. La distorsion est d'autant plus élevée que le champ de vue de la caméra est grand. Cependant, il est possible de modéliser la distorsion en enrichissant le modèle sténopé par des termes supplémentaires (le modèle

devient non linéaire).

Le calibrage permet de déterminer les paramètres intrinsèques et extrinsèques d'une ou plusieurs caméras en utilisant une mire dont le modèle est connu à l'avance. Pour réaliser le calibrage de deux caméras stéréoscopiques, les paramètres à estimer sont (figure A.8) :

- Les paramètres intrinsèques de chaque caméra ;
- Les paramètres extrinsèques (la matrice de rotation et de translation et de la caméra IR par rapport à la caméra RGB).
- Les coefficients de distorsion.

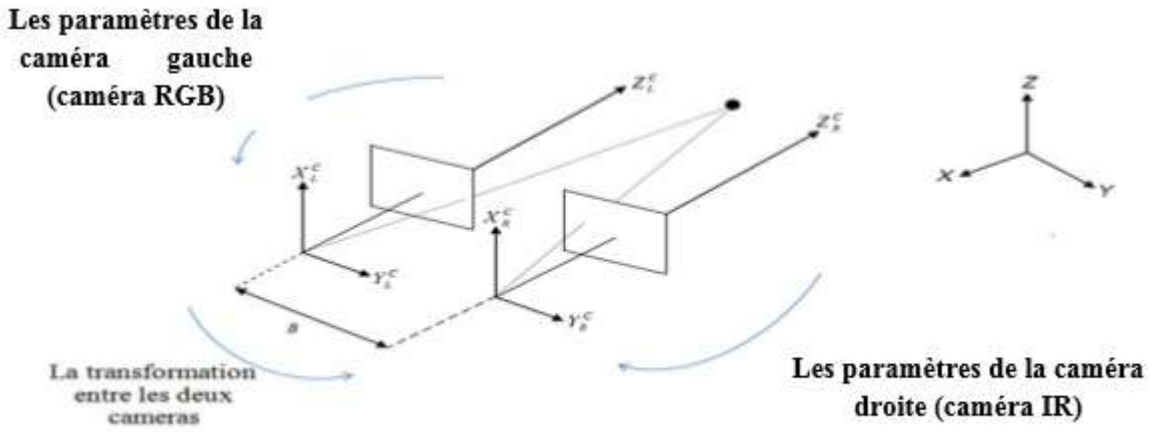


Figure A.8 : Projection perspective

Pour déterminer les paramètres du modèle de la caméra, nous la placerons devant une mire (objet étalon) de calibrage. C'est un ensemble de points dont les coordonnées sont parfaitement connues dans un repère de la mire qui est différent du repère caméra. Chaque point de la mire se projette dans l'image, puis nous mesurons ses coordonnées dans le repère image.
Le point Q est défini par ses coordonnées (x_w, y_w, z_w) dans le repère univers. Il est également précisé par ses coordonnées (x_c, y_c, z_c) dans le repère caméra.

A.2.1 Les paramètres intrinsèques

Les paramètres intrinsèques décrivent les paramètres internes d'un appareil tels que la distance focale, les paramètres de lentille radiaux, centre de l'image, le facteur d'inclinaison. A partir de la figure A.9, le centre de projection (le point auquel tous les rayons qui se croisent) est désignée par le centre optique ou le centre de la caméra et la ligne perpendiculaire au plan de l'image passant par le centre optique de l'axe optique. En outre, l'intersection du plan d'image avec l'axe optique est appelé le point principal.

Le changement de repère qui lie les coordonnées du point dans le repère image (figure A.3) avec le repère lié à la caméra est défini par les paramètres intrinsèques de la caméra.

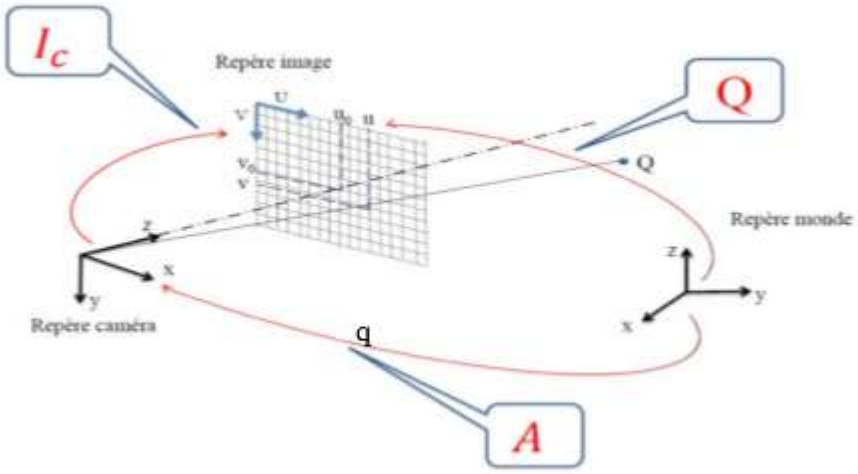


Figure A.9 : Projection perspective

En projection perspective, un point Q est transformé en son homogène q dans le plan image par :

$$\begin{cases} u = u_0 - k_u\, f\, \frac{x_c}{z_c} \\ v = v_0 + k_u\, f\, \frac{y_c}{z_c} \end{cases} \qquad \text{(A.1)}$$

Avec : $Q = I_c * A$.

(u, v): sont les cordonnées dans le repère image,

(u_0, v_0): est le centre d'image.

f : la distance focale, (k_u, k_v): sont des facteurs d'échelle.

(x_c, y_c): Sont les coordonnées du point dans le repère caméra.

Soit en rotation matricielle :

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} -k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \qquad \text{(A.2)}$$

$$= \begin{bmatrix} -k_u f & 0 & u_0 & 0 \\ 0 & k_v f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \quad \text{(A.3)}$$

Avec : $u = U/W$ et $v = V/W$.

On pose :

$$I_c = \begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{(A.4)}$$

Avec : $\alpha_u = -k_u f$ et $\alpha_v = k_v f$.

C'est une application linéaire de l'espace projectif vers le plan projectif exprimant la transformation perspective :

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = I_c \begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix} \quad \text{(A.5)}$$

Dans ce modèle figure quatre paramètres $\alpha_u, \alpha_v, u_0 \; et \; v_0$ ce sont les paramètres qui vont être estimés par le calibrage. Notons que la distance focale ne peut être calculée explicitement.

Nous introduisons des coordonnées caméra normalisées par rapport à Z telles que :

$$\begin{cases} x_c = \frac{x_c}{z_c} \\ y_c = \frac{y_c}{z_c} \\ z_c = 1 \end{cases} \quad \text{(A.6)}$$

À partir de cela la relation entre les coordonnées image et les coordonnées caméra est :

$$\begin{cases} u = \alpha_u x_c + u_0 \\ v = \alpha_v y_c + v_0 \end{cases} \quad \text{(A.7)}$$

A.2.2 Paramètres extrinsèques

Les paramètres extrinsèques indiquent la position extérieure et l'orientation d'une caméra dans le monde 3-D. Normalement, un objet dans l'espace est défini par son propre cadre de coordonnées. En mettant en place un système de coordonnées fixe connu comme le cadre de référence, la position et l'orientation d'un point quelconque sera exprimé dans ce cadre. Ceci est important pour initialiser un système dans l'espace euclidien que le mouvement de corps rigide est calculé par rapport à une position donnée initiale et l'orientation. Par conséquent, il se pose la nécessité d'aligner tout objet perçu dans l'espace par rapport à un repère orthonormé de référence initialisés qui dans notre cas est considéré comme celui de la caméra. Tout point rigide dans l'espace est soumis à deux transformations de base; une traduction définissant la distance parcourue et / ou de rotation définissant la torsion soumis.

Les paramètres extrinsèques permettent de définir la transformation géométrique entre le repère caméra et la scène (repère objet) voir figure A.4, ils sont représentés par la matrice de l'équation A.8.

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(A.8)}$$

Dans ce cas, nous devons extraire la matrice de transformation entre les deux repères des caméras à partir le Toolboox calib-gui qui nous permet d'obtenir la matrice de rotation et de translation ainsi que les positions de la mire par rapport à la caméra pendant le calibrage (figure A.10).

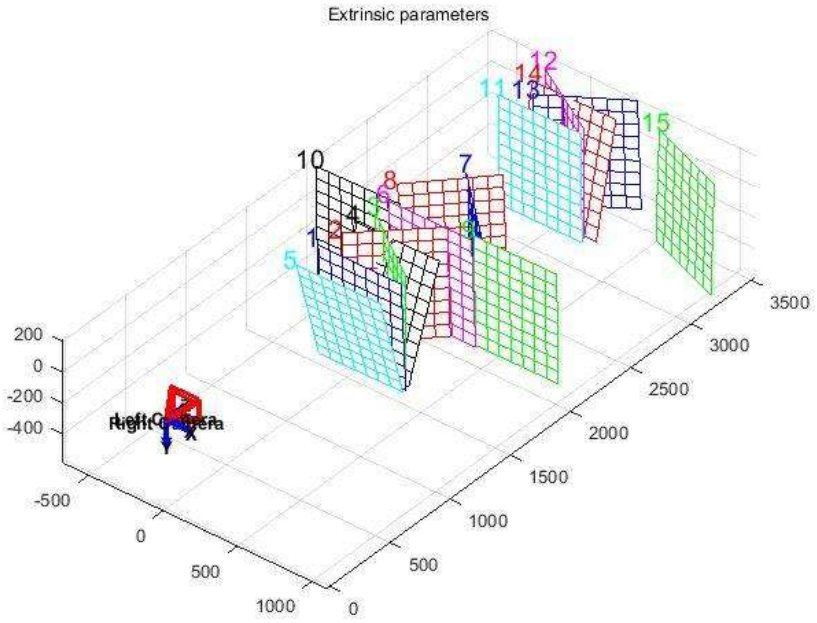


Figure A.10 : Simulation de position des deux caméras (RGB, IR) par rapport à la mire.

Les résultats de calibrage obtenus sont représentés dans le tableau I-1, ils montrent que la matrice de rotation entre les deux caméras est égale à l'identité. La première composante du vecteur de translation (selon x) est égale à 22.63778 mm , dont on sait a priori (mesure prise manuellement) qu'elle est égale à 22 mm, donc nous pouvons dire que notre estimation des paramètres est juste avec une petite erreur de 0.63778, voir figure A.11.

Caméra droite IR	Le focal	$\alpha_u = 604.35681$ $\alpha_v = 604.32652$
	Le point central	$u_0 = 324.51007$ $v_0 = 256.74468$
Caméra gauche RGB	Le focal	$\alpha_u = 530.30989$ $\alpha_v = 529.14101$
	Le point central	$u_0 = 317.338$ $v_0 = 243.88434$
Transformation entre les deux caméras	Vecteur de rotation	$\alpha = 0.01972$ $\beta = 0.00505$ $\gamma = -0.0045$
	Vecteur de translation	$t_x = -22.63778$ $t_y = -1.81018$ $t_z = 21.51823$

Table A.1 : Résultats de calibrage stéréoscopique de la Kinect.

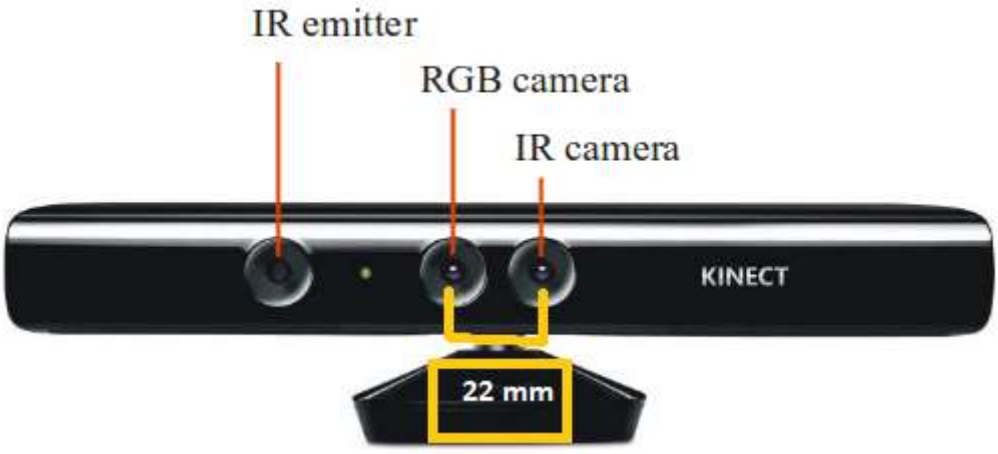


Figure A.11 : Distance Base ligne

ANNEX B

B.1 ROS

De nos jours, la recherche robotique est beaucoup plus simple qu'avant, et cela grâce au développement et à l'évolution des outils utilisés notamment les systèmes d'exploitation et les plates-formes robotique soft et hard.

Avant les OS robotiques, chaque concepteur de robot, chaque chercheur en robotique passait un temps non négligeable à concevoir matériellement son robot ainsi que le logiciel embarqué associé. Cela demandait des compétences en mécanique, électronique et programmation embarquée. Généralement, les programmes ainsi conçus correspondaient plus à de la programmation embarquée, proche de l'électronique, qu'à de la robotique proprement dite.

B.1.1 Historique de ROS

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. Pendant ce temps, des chercheurs de plus de vingt institutions ont collaborés avec les ingénieurs de Willow Garage dans un modèle de développement fédéré [2].

B.1.2 Apport de ROS pour la robotique

L'écriture de logiciels pour les robots est difficile, d'autant que la richesse et la portée de la robotique continuent de croître. Différents types de robots peuvent être très variables côté matériel, ce qui rend la réutilisation de code non triviale, ROS favorise la réutilisation du code de sorte que les développeurs en robotique et les scientifiques ne font pas de réinventer la roue tout le temps. Donc l'idée principale de l'outil ROS est de proposer des fonctionnalités standardisées qui sont valables et utilisables dans tous les types de robots, tout comme un système d'exploitation classique pour PC [21], un autre avantage de l'OS robotique comme ROS est de rassembler des savoir-faire de différentes disciplines. En effet, concevoir et programmer un robot, c'est:

- Gérer le matériel en écrivant les pilotes,
- Gérer la mémoire et les processus,

- Gérer la concurrence et la fusion de données,
- Proposer des algorithmes de raisonnement abstrait faisant largement appel à l'intelligence artificielle.

B.1.3 L'organisation général de ROS

La philosophie de ROS se résume dans les 5 grands principes suivants :

***Peer to Peer* (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 languages: ROS est neutre d'un point de vue langage et peut être programmé en différents langages. La spécification de ROS intervient au niveau message.

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 rewrappe 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'un runtime monolithique, ROS a adopté un design microkernel qui utilise un grand nombre de petits outils pour faire le build et le run des différents composants ROS.

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 librairies. Cette organisation facilite en plus le test unitaire. Enfin, ROS utilise du code (pilotes et algorithmes) issus d'autres projets open source :

- Simulateur Player/stage
- Librairie de traitement d'image et de vision artificielle : OpenCV

- Algorithme de planification : OpenRave
- …

Gratuit et open source : ROS passe des données grâce à de la communication interprocess. De ce fait, les modules n'ont pas besoin d'être liés dans un unique process, facilitant ainsi l'usage des différentes licences.

Nous avons déjà expliqué les raisons de ce choix précédemment. Notons toutefois que l'architecture choisie est cohérente avec ce choix : ROS passe des données grâce à de la communication interprocess. De ce fait, les modules n'ont pas besoin d'être liés dans un unique process, facilitant ainsi l'usage des différentes licences.

B.1.4 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 (nous verrons ci-dessous ce qu'est un nœud), les librairies externes, des données, des fichiers de configuration et un fichier de configuration xml nommé manifest.xml.
- **Manifest** : c'est un fichier qui fournit des informations concernant les packages, il définit les dépendances et les bibliothèques utilisées par le compilateur.
- **La stack** : Une stack est une collection de packages. Elle propose une agrégation de fonctionnalités telles que la navigation, la localisation… Une stack est un répertoire qui contient les répertoires des packages ainsi qu'un fichier de configuration nommé stack.xml.

B.1.5 Les notions de base de ROS

Le principe de base d'un OS robotique est de faire fonctionner en parallèle un grand nombre d'exécutables 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 continu 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

a. Les nœuds

Dans ROS, un nœud est un exécutable qui peut correspondre à un capteur, un moteur, un algorithme de traitement, de surveillance. Chaque nœud qui se lance se déclare au Master et ils peuvent communiquer avec d'autres processus en utilisant des topic, des services, ou le serveur de paramètres. L'utilisation de nœuds dans ROS nous fournit la tolérance de panne et séparation entre le code et les fonctionnalités rendant le système plus simple.

b. 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

c. Les topics

L'échange de l'information s'effectue soit de manière asynchrone via un topic ou de manière synchrone via un service.

Un topic est un système de transport de l'information basé sur le système de l'abonnement / publication (subscribe / publish).

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.

d. 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).

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

e. 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

La description des services est stockée dans *nom_package/srv/monServiceType.srv*. Ce fichier décrit les structures de données des requêtes et des réponses.

f. 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.

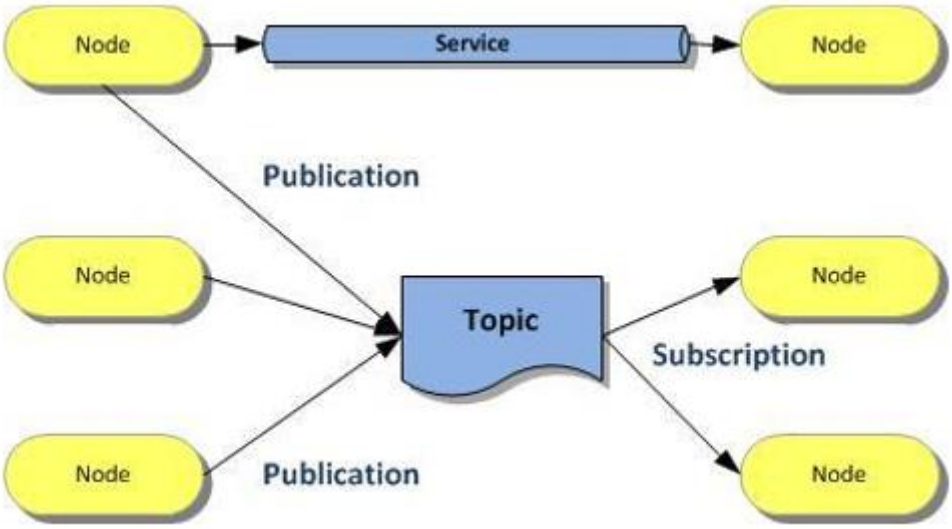


Figure B.1 : exemple d'un système de traitement et de calcul ROS.

B.1.6 La notion (URDF)

D'autres notions que nous pourrons découvrir avec ROS, il s'agit de l'URDF (*Unified Robot Description Format*), un format XML permettant de décrire sous la forme d'un fichier standardisé un robot complet. Le robot ainsi décrit peut être statique ou dynamique et des propriétés physiques et de collision peuvent y être ajoutées.

Outre le standard, ROS propose plusieurs outils permettant de générer ou valider ce format. L'URDF est utilisé par exemple par le simulateur Gazebo pour représenter le robot.