ENSTA Bretagne

Rapport de projet de fin d'études

M'Hamed Fadil ENNOUHI

Détection d'obstacle, sur un robot de manutention, à l'aide d'un capteur de vision 3D.

Balyo

Encadrant : Benoit Zerr

Tuteur : Vincent Rebaud

Paris 2017

Remerciements

Je remercie mon tuteur Vincent Rebaud pour son aide. Sa patience et ses connaissances en vision robotique et en algorithmique générale m'ont permis de beaucoup progresser durant mon stage à Balyo.

Je tiens à remercier également mon professeur encadrant Benoit Zerr pour son aide tout au long de mon cursus à l'ENSTA Bretagne. Son engagement et son attrait pour l'aspect pratique ont fait que ses cours aient été parmi mes favoris à l'ENSTA.

Enfin je tiens à remercier tous l'ensemble de l'entreprise Balyo pour leur accueil. En particulier l'équipe Recherche et Développement avec laquelle j'ai passé le plus de temps et dont les membres m'ont non seulement proposé leur aide à plusieurs reprises mais qui m'ont accueilli dans les activités extérieures au travail. Cela a rendu ma période de stage loin de ma famille et mes amis plus agréable.

Abstract

This paper is a report of the work done during my internship at Balyo. The goal was to develop algorithms for the processing of data outputed by a 3D camera.

The main algorithms that I have worked on are the auto-calibration algorithm and the obstacle detection algorithm. I have mainly coded in c++ on an ubuntu Operating system. I have also used in my projects the Point Cloud Library (PCL) along with the Robot Operating System (ROS). These two libraries enabled me to make fast prototypes and helped a lot by providing already existing functions ready to use thus letting me focus on the problem at hand by lowering the coding burden.

My mission regarding the obstacle detection algorithm was to find the right set of parameters that lowers false positive detection without letting through any obstacle. If I managed to provide a set of parameters that filtered the majority of noise and false detection, some types of errors could not be filtred.

On the other hand, I wrote the auto-calibration algorithm from scratch with the help of my tutor. Testing on fork mounted cameras showed positive, providing correct coordinates of the 3D camera in the truck's coordinate system.

Résumé

Ce rapport présente le travail que j'ai réalisé pendant mon stage à Balyo. L'objectif était de développé des algorithmes de traitement de données d'une caméra 3D.

Les algorithmes principaux sur lesquelles j'ai travaillé sont l'algorithme d'auto-calibration et celui de détection d'obstacle. J'ai principalement écrit mon code en c++ sur une machine ayant Ubuntu comme système d'exploitation. J'ai également utilisé la librairie Point Cloud Library (PCL) et l'intergiciel Robot Operating System (ROS). Ils m'ont permis de créer des prototypes de manière rapide et m'ont également aidé en procurant des fonctions utiles. Cela m'a permis de plus me concentrer sur la problématique en allégeant la charge de code à produire.

Ma mission concernant l'algorithme de détection d'obstacle était de trouver les paramètres adéquats qui permettent de diminuer la probabilité de détection de faux positif sans manquer d'obstacle. Si j'ai réussi à trouver une combinaison de paramètres permettant de diminuer le bruit et les fausses détections, certains types d'erreur ne peuvent pas être filtrés.

L'algorithme d'auto-calibration a constitué le principal travail que j'ai effectué étant donné que je l'ai écrit en entier avec l'aide de mon tuteur. Les tests sur une caméra montée sur les fourches d'un chariot ont montré des résultats prometteurs. En effet l'algorithme détermine les coordonnées correctes de la caméra dans le repère du chariot.

Introduction

La vision en robotique est un sujet majeur et qui connais récemment des avancées importantes. En effet grâce aux progrès à la fois en puissance de calculs des systèmes embarqués mais aussi à ceux réalisés au niveau des capteurs et de l'intelligence artificielle, de plus en plus de robots bénéficient de la vision.

Les applications sont nombreuses dans le domaine de la santé de la sécurité et du transport. Le présent travail décrit une application de cette technologie dans la robotisation du transport de marchandise dans les entrepôts. Dans la suite de ce rapport seront détaillé la problématique posé, les solutions ainsi que les pistes d'améliorations suggérées.

Table des matières

Chapitre 1

Présentation de Balyo

1.1 Historique

Balyo a été créée en 2005 par Raul Bravo et Thomas Duval avec pour mission augmenter la compétitivité de ses clients en automatisant leurs flux intra-logistique.

Balyo fait cela principalement en robotisant de chariots de manutentions. En effet le nom Balyo vient de la mythologie grec Balios, un cheval d'Achille, en référence au transport de marchandise. La zone de test des chariots également est appelé le "Paddock" pour prolonger cette métaphore.

FIGURE 1.1 – Locaux de Balyo

Ce qui différencie Balyo de ses concurrents c'est qu'elle prend des chariots déjà existant et les munis d'une panoplie de robotisation au lieu de construire depuis zéro un chariot autonome. La panoplie de robotisation est composé d'une MOVEBOX (ordinateur) et de plusieurs capteurs (Lasers, caméras, lecteurs code bar). Grâce à cette approche Balyo a pu conclure un partenariat avec Linde en Europe, l'un des plus grands producteurs de chariots de manutentions et Hyster-Yale aux États-Unis d'Amérique.

En plus de bénéficier de la capacité et qualité de production des chariots Linde et Hyster-Yale, Balyo profite par la même occasion des agences et maintenance de ses grandes sociétés présentes partout dans le monde, chose qui facilite l'implantation de

Balyo dans de nouveaux pays.

L'année 2017 est probablement l'année la plus importante de Balyo car elle fit son introduction en bourse en Juin 2017 et réussi à lever plus de 40 millions d'euros.[1]

1.2 Structure

Même après plus de dix ans d'existence, Balyo reste une entreprise qui évolue continuellement et rapidement, préservant sa culture de start-up. Un organigramme représentant toute l'entreprise n'est pas disponible. Ci-dessous un diagramme regroupant les différents pôles du département ingénierie. Sous tutelle de M. Vincent Rebaud j'étais rattaché au pôle Robot Software, chargé du développement de nouvelles fonctionnalités du chariot. Ces nouvelles fonctionnalités sont testées par le pôle validation avant d'être mises à disposition de l'équipe installation. Enfin une équipe d'ingénieurs installation est dispatchée chez le client pour pouvoir définir les réglages spécifiques des chariots pour un fonctionnement optimal dans le site du client.

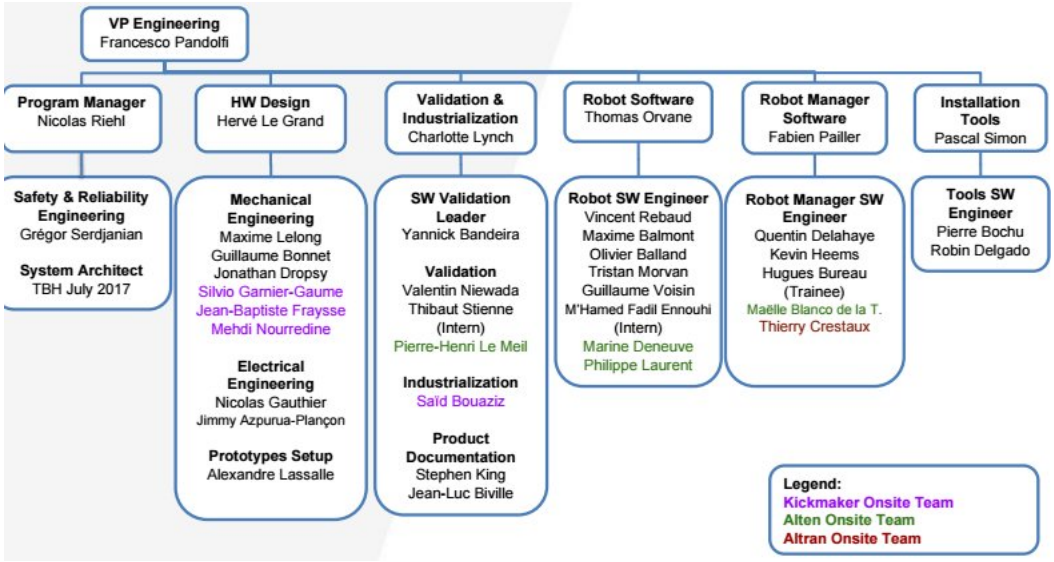

FIGURE 1.2 – Organigramme du département d'ingénierie

1.3 Locaux

Balyo est installée à Moissy-Cramayel, en périphérie parisienne. Le choix s'est porté sur des bureaux relativement loin de la capitale pour des raisons d'espace. En effet pour pouvoir modifier et tester les chariots, un espace considérable est nécessaire.

Dans ces locaux actuelle Balyo possède plusieurs zones de test adaptées à chaque mission, chariot et site clients. Les ingénieurs de recherche et développement ainsi que ceux de validations bénéficient ainsi d'une proximité des robots qui leur économise du

temps et permet d'être également proche de l'équipe production qui peut intervenir rapidement en cas de problème mécanique sur un chariot.

FIGURE 1.3 – Zone de test des chariots

1.4 Services et Produits

Balyo conçoit et installe des solutions autonomes afin de déplacer des biens chez ses clients. Il s'agit principalement des chariots transpalettes autonomes. Balyo propose également une solution de communication et de centralisation de contrôle des convoyeurs, des machines d'emballages et des portes automatiques.

La demande dans ce secteur surpasse largement l'offre c'est pour cela que Balyo est en perpétuelle croissance aussi bien en Europe qu'aux États-Unis et en Asie. Pendant ma période de stage, Balyo avait des contrats avec de grands groupes tels que BMW et Renault. Ces grandes entreprises ont choisi de faire confiance à Balyo car contrairement à ses concurrents, elle propose une solution innovante, simple d'utilisation et qui ne requiert pas d'infrastructure à installer dans les entrepôts des clients.

En plus du SDK que le département de Recherche et développement fournis pour le robot, un superviseur est également proposé. Ce dernier permet de diriger une flotte de machine mais aussi d'assigner les missions, de gérer les priorités et de remonter les problèmes rencontrés lors du fonctionnement.

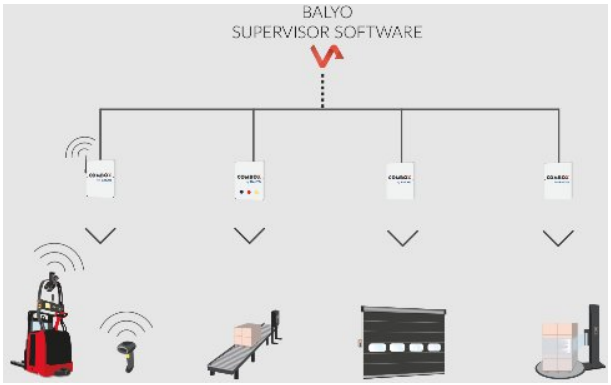

FIGURE 1.4 – Produits Balyo

Chapitre 2

Chariots

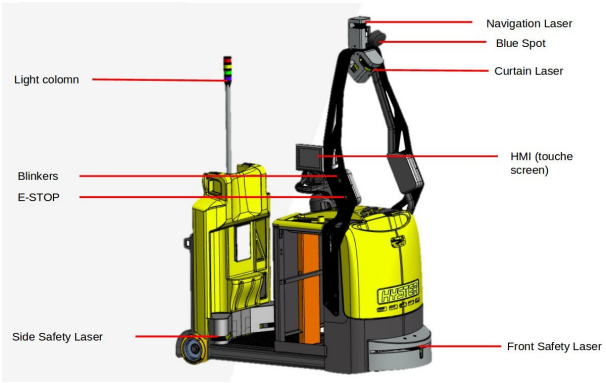

FIGURE 2.1 – Chariot équipé par Balyo

2.1 Types

Il existe plusieurs chariots développés par Balyo, de taille et de capacité différentes mais ils peuvent être regroupés en deux catégories. Les chariots avec et sans fourche. Les chariots sans fourche ont pour but de déplacer, à la manière d'un train des cadis à roues transportant les biens du client à l'intérieur de l'usine (outils, matière première). Les chariots à fourche quant à eux déplacent des palettes dans des entrepôts.

2.2 Base roulante

La plupart de chariots sont composé pour la partie motorisation d'une roue disposant de deux moteurs électriques lui permettant de tourner à la fois pour faire assurer la traction du chariot mais aussi la direction. Ils disposent également de galets positionnés sous les fourches ou à la base du chariot.

Les chariots fonctionnant en intérieur il est nécessaire que la motorisation soit

électrique, ils disposent donc de batteries (plomb-acide) plus ou moins grande et lourde, allant jusqu'à 1 500Kg pour le chariot Linde K ci-dessous.

FIGURE 2.2 – Chariot Linde K

2.3 Capteurs

Pour pouvoir évoluer dans leur environnement en toute sécurité, les chariots ont besoin de plusieurs données sur l'espace qui les entoure et leurs propres coordonnées (position, vitesse, hauteur des fourches...). Ces informations sont donc fournies par une panoplie de capteurs que Balyo installe en production.

2.3.1 Laser de navigation

Au point le plus haut du chariot on trouve le laser de navigation qui est un LIDAR plan permettant de déterminer la position du chariot par rapport aux murs de la salle où se trouvent le chariot, la cartographie étant déjà faite au préalable le chariot se repère à tout instant ainsi.

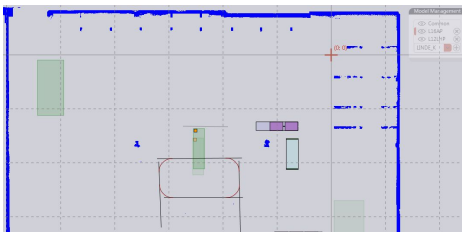

FIGURE 2.3 – Cartographie dans l'éditeur de circuit

La technique Simultaneous Localization And Mapping (SLAM)[2], est utilisée par Balyo pour cartographier l'entrepôt. Le lidar de navigation ainsi que les odomètres du chariot sont donc exploités pour établir cette cartographie. Balyo ne donne pas plus de détails sur ses algorithmes de SLAM gardés secrets.

FIGURE 2.4 – Lidar de navigation

2.3.2 Laser rideau

Positionné également sur la partie haute du chariot, ce laser plan incliné d'environ 45° permet au chariot de détecter tout obstacles surélevés qui se trouve sur son chemin. En cas de détection cela déclenche un arrêt jusqu'à ce que l'objet détecté ne se trouve plus sur le trajet. En pratique cela permet de s'arrêter avec une bonne marge de sécurité devant un opérateur ou un autre chariot.

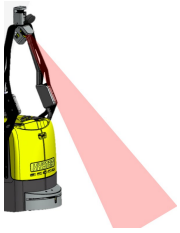

FIGURE 2.5 – Laser rideau

Cependant ce capteur pose un problème. En effet après la détection d'un objet par le laser rideau, le chariot s'arrête et a besoin d'être remis en ligne par un opérateur. Ceci dû au fait que le robot ne peut pas savoir si l'objet détecté est toujours présent entre le plan de détection et le chariot.

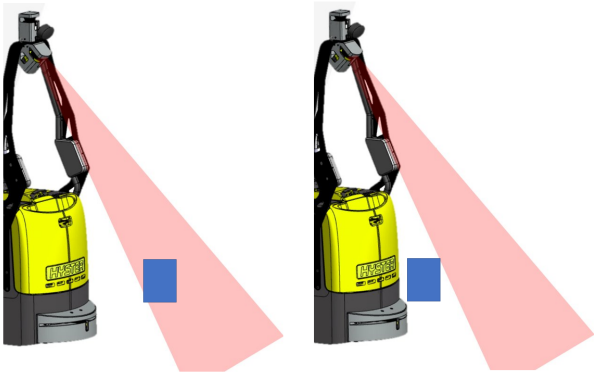

FIGURE 2.6 – Cas particulier obstacle

Dans la figure précédente le chariot ne voit plus l'obstacle avec le laser rideau et ne peut également pas le détecté avec le laser de sécurité du bas car l'obstacle est en hauteur. Balyo souhaite donc remplacer ce capteur par une caméra 3D.

2.3.3 Laser de sécurité

Ce dernier type de laser se trouve au niveau le plus bas du chariot, il permet de créer une zone de sécurité grâce à la présence de lasers avant, arrière et coté autour du chariot et lui permet de s'arrêter si un objet est posé sur le sol à côté de celui-ci. Il permet également de s'arrêter si des opérateurs circulent à proximité du chariot et de respecter la norme 1525 [3] qui régit les chariots sans conducteur.

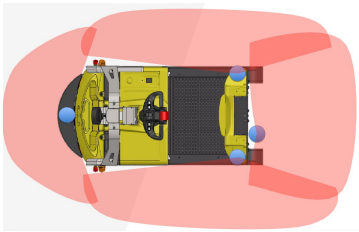

FIGURE 2.7 – Champs des lasers de sécurité

2.3.4 Caméra 3D

Si la caméra 3D n'est pas encore utilisée comme substitue au laser rideau, elle est utile en revanche sur certains chariots pour permettre à la fois la détection de palettes mais aussi de vérifier si un espace est bien vide avant de procéder à la dépose. La caméra 3D permet au chariot de corriger la position de ses fourches avant une prise de palette.

FIGURE 2.8 – Caméra 3D fourche

2.3.5 Télémètre laser

Fixé sur les fourches, il permet de mesurer la distance qui sépare les fourches de la palette que l'on souhaite récupérer. Cela permet en cas d'erreur dans le placement des fourches dans la palette de vérifier que le chariot ne pousse pas cette dernière et d'éviter ainsi de la faire tomber de son emplacement.

FIGURE 2.9 – Fourches mal positionnées

Dans le cas représenté, les fourches poussent la palette, au-lieu de voir la distance séparant les fourches de la palette diminuer, la valeur du télémètre reste inchangée. Ceci fait surgir une erreur qui empêche le chariot de continuer à pousser la palette.

2.4 Interfaces Homme Machine

Afin de pouvoir interagir et prévenir de ses mouvements les personnes se trouvant dans l'environnement d'évolution du chariot, Balyo équipe ses robots de plusieurs interfaces homme machine visuel et sonore. Ainsi chaque chariot est équipé d'un Blue Spot qui permet de signaler le trajet du chariot. Des beepers qui se mettent en route lors de la mise en marche du chariot et de ses changements de cap et finalement des clignotants qui signalent la direction que s'apprêtent à prendre le chariot. Aussi pour pouvoir lancer de manière aisée les missions du chariot, il dispose d'un écran tactile.

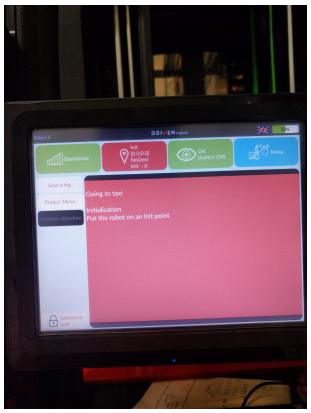

FIGURE 2.10 – Interface graphique du chariot

Chapitre 3

Camera 3D

FIGURE 3.1 – Caméra IFM OD303

3.1 Types

Malgré tous les progrès fait ces dernières années en calculs et développement de système intelligents, la problématique de vision reste encore un obstacle majeur qui entrave l'avènement de systèmes autonomes évoluant dans des environnements inconnus et sujet aux changements.

Il existe plusieurs types de caméra qui permettent de procurer aux robots une vision de leur espace d'évolution.

3.1.1 Caméra Time Of Flight

Solution choisie par Balyo, les caméras ToF permettent d'obtenir la dimension de profondeur d'une image en mesurant le temps écoulé entre l'émission et la réception d'une grille infrarouge émis par une multitude de LED.

L'atout de cette caméra réside dans la faible puissance de calcul nécessaire pour le traitement de ses données et de sa rapidité. En revanche ce type de caméra ne fonctionne pas en extérieur car le soleil perturbe ses données.

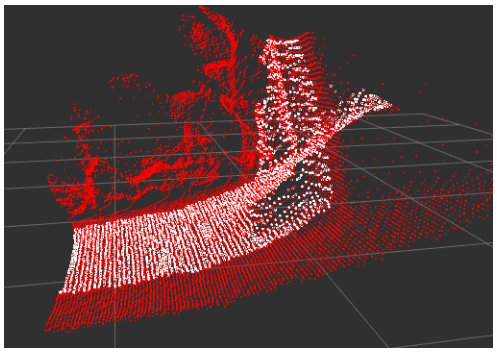

FIGURE 3.2 – Nuage de point en présence de soleil

3.1.2 Caméra stéréo

De la même manière que nous parvenons à apercevoir le monde en 3 dimensions grâce à nos deux yeux, les caméras stéréo possède deux capteurs photos permettant d'obtenir l'image d'une scène vue sous deux angles différents. Ce type de caméra n'est pas influencée par la présence de soleil et peut donc fonctionner en extérieur néanmoins le traitement est plus lourd et plus lent que les caméras ToF.

FIGURE 3.3 – Caméra stéréo ZED

3.2 Principe

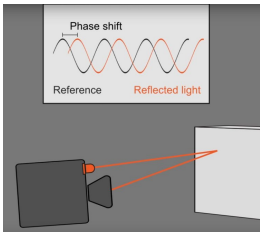

FIGURE 3.4 – Illustration différence de phases

Les chariots opérants en intérieur et dans un environnement peu exposé au soleil, le type de caméra ToF a été choisie par Balyo pour les équiper. La caméra IFM O3D303

avec laquelle j'ai travaillé, fonctionne en émettant à un intervalle régulier des pulsions d'ondes infrarouge et en calculant le temps de vol des faisceaux, elle en déduit la profondeur. Ce temps de vol est obtenu en calculant le déphasage entre l'onde émise et réfléchie[4].

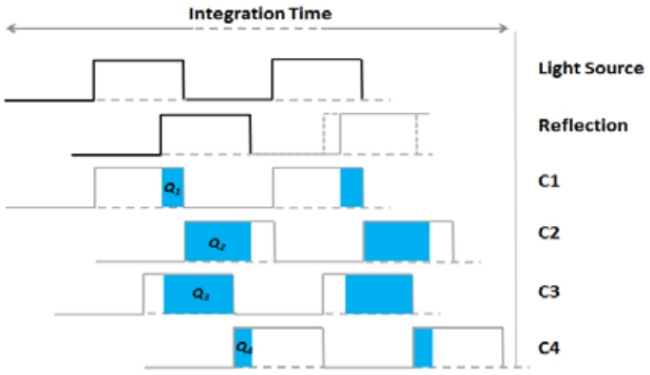

FIGURE 3.5 – Représentation méthode ToF

$$\phi = \arctan(\frac{Q_3 - Q_4}{Q_1 - Q_2}) \tag{3.1}$$

$$d = \frac{c}{4.\pi.f}\phi \tag{3.2}$$

Avec f la fréquence de l'onde choisie et c la vitesse de la lumière.

3.3 Problèmes rencontrés

Durant les utilisations de la caméra 3D j'ai relevé plusieurs problèmes de fonctionnement liés à l'utilisation d'ondes infrarouge.

3.3.1 Points fantôme

Ces points fictifs ont tendance à apparaître sous forme de lignes collées à la partie haute du champ de vision de la caméra dans le nuage de points. Ils sont causés par une confusion de deux salves successives des émetteurs infrarouges. En effet la caméra pense qu'il s'agit de réflexions d'onde émise à un instant t alors qu'il s'agit d'onde émise à t-1. L'estimation de la profondeur se trouve donc compromise et fausse. La solution adoptée pour prévenir cette erreur et de réduire le champ de vision de la caméra.

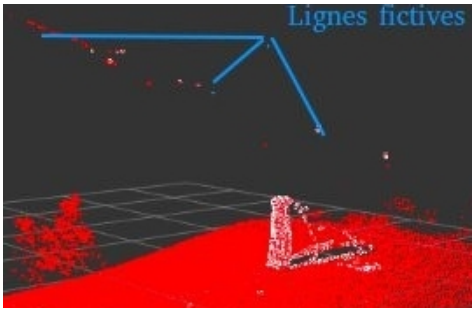

FIGURE 3.6 – Nuage de points brute

3.3.2 Chevauchement

La caméra IFM O3D303 fonctionnant par émission de salves d'ondes infrarouge à une fréquence donné, si plus d'une caméra se retrouve à consulter la même scène (croisement de deux chariots) les ondes se chevauchent et une erreur se produit. Le logiciel de réglages de la caméra 3D fournis par IFM permet de choisir entre 3 fréquence d'émissions, il suffit donc de s'assurer que deux caméras qui risquent de consulter la même scène à un même moment ne soit pas réglées sur la même fréquence.

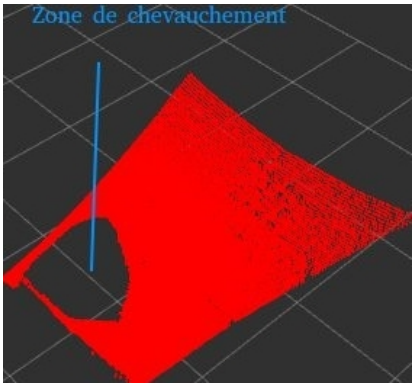

FIGURE 3.7 – Nuage de points d'une scène vue par deux caméras

3.3.3 Erreur due a des textures singulière

Une autre erreur constatée lors de la phase de test est des erreurs d'estimation de profondeur en consultant certains types de surfaces. Si l'erreur est seulement de quelques centimètres elle reste gênante parce que cela ce produit quand on observe des types de sol particuliers. Ainsi le robot détecte comme obstacle le même sol sur lequel il roule. Cependant cette erreur peut être corrigé en rehaussant le seuil au-delà duquel un objet est considéré comme étant un obstacle mais cette méthode ne peut pas être utilisée si la texture du sol change pendant le trajet du robot. Seule solution dans ce cas-là est de définir des zones où le robot sait qu'il doit rehausser son seuil de détection d'obstacle.

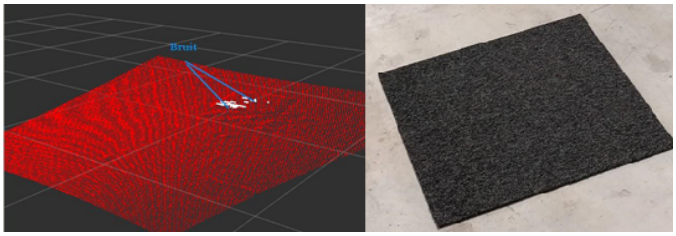

FIGURE 3.8 – Type de sol engendrant des erreurs de détection

3.3.4 Surchauffe

La caméra IFM O3D303 possède plusieurs paramètres réglables sur le pilote fournis. Il est possible de choisir le nombre d'images par seconde que l'on souhaite obtenir, la durée d'exposition, de réglé la portée maximale, de choisir le canal d'émission des ondes pour prévenir l'erreur de chevauchement et même d'appliquer des filtres spatiaux et temporels directement sur les nuages de points.

Tous ces réglages consomment beaucoup de puissance de calcul et si mal réglé, la caméra a du mal à maintenir une fréquence d'image acceptable (plus de 15 images par seconde) et parfois même s'éteint. Il est donc nécessaire de vérifier que les paramètres entrés ne surchargent pas la caméra en traitement ou d'avoir une fixation mécanique permettant de réaliser une bonne dissipation thermique.

3.3.5 Soleil

Le soleil émettant sur un large spectre de longueur d'ondes (de 250 à 2 500 nm), il perturbe les données relevées par la caméra 3D car cette dernière utilise des fréquences présente dans l'infrarouge aux alentours de 800nm. Il est malheureusement impossible de filtrer ou de corriger les erreurs introduites par la présence d'éclairage du soleil sur une scène. D'où l'obsolescence de ce type de caméra en extérieur.

3.3.6 Calibration

En fin la dernière erreur constatée n'est pas lié au principe du fonctionnement de la caméra mais à la manière dont celle-ci est fixée sur le chariot prototype avec lequel j'effectuais mes tests. En effet il a été décidé que la caméra serait fixée de la même manière que le laser rideau, c'est à dire en haut du chariot avec une inclinaison avoisinant les 45°.

L'une des premières tâches qui m'eut été donné par mon tuteur été justement de vérifier quel effet à cet angle sur la présence de bruit dans les données de la caméra. Il fallait également trouver un compromis entre cet angle et le champ de vision de la caméra. Vu que la fixation de la caméra était assurée par une paire d'aimant, l'angle de la caméra nécessaire pour pouvoir remettre le nuage de point dans le repère du chariot n'était pas connu précisément. Il fallait donc manuellement trouver cet angle ce qui n'était pas précis. En effet si cet angle est mal déterminé des objets présents dans le

nuage de points sur le sol peuvent être soit vu plus grand ou plus petit qu'ils le sont, ou, pire le sol peut alors être détecté comme obstacle.

La solution été donc d'utilisé PCL pour pouvoir faire une détection de plan et en déduire l'orientation de la caméra. Cette méthode est détaillée par la suite dans le chapitre traitant l'Autocalibration.

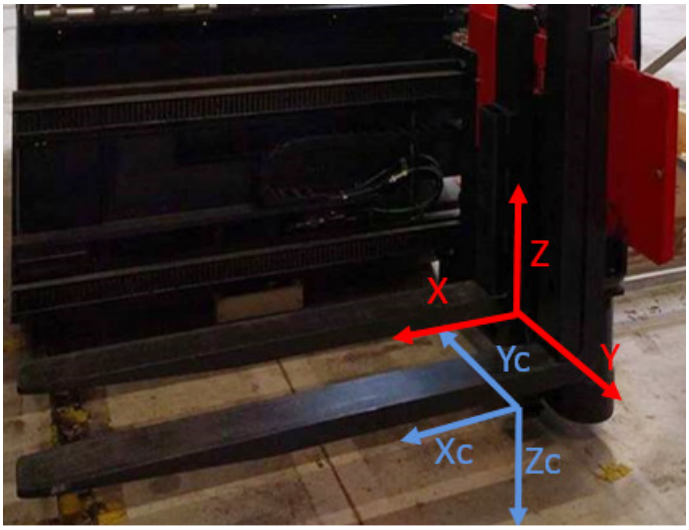

FIGURE 3.9 – Repère caméra(bleue) et chariot(rouge)

Chapitre 4

ROS et PCL

4.1 ROS

ROS étudié à l'ENSTA en 2-ème année et appliqué en 3-ème lors des projets Shepherd et Guérledan permet de gérer les données et d'interfacer plusieurs capteurs. Durant mon stage je l'ai principalement utilisé pour le prototypage. En effet grâce à la fonction Dynamic Reconfigure de RQT, j'ai pu de manière empirique trouver les bons réglages de filtres appliqués aux nuages de points.

FIGURE 4.1 – Logo de Robot Operating System

4.1.1 Introduction

Le Robot Operating System se veut, comme son nom l'indique, être un système d'exploitation pour robots. Créé par le laboratoire Willow Garage, son objectif principal est de faciliter la création d'application robotique en réduisant la quantité de code qui doit être écrit à chaque nouveau projet.

C'est pour cela que ROS promeut la réutilisation de code sous forme de nœuds [5]. Ainsi permettant de ne pas avoir à réinventer la roue à chaque fois mais aussi facilitant le travail avec une multitude capteur équipant un même robot.

4.1.2 Application

Durant mon stage j'ai utilisé ROS pendant plusieurs étapes de développement de mes projets. J'ai ainsi pu bénéficier de l'outil de visualisation RViz pour avoir un aperçu des modifications que j'effectuais sur mes nuages de points. Grâce à ROS également il m'était possible de voir les effets de mes paramètres sur la rapidité de l'algorithme de filtrage et les fréquences d'affichage de nuage développé par mon tuteur.

Par la suite pour la création de mon algorithme d'auto-calibration, ROS m'a permis

de publier et souscrire à plusieurs orientations et versions de nuages de points sur lesquelles j'ai effectué différentes opérations de transformations et de filtrage. D'autant plus que grâce à RQT je pouvais en ayant créé les structures et .cfg adéquat faire varié mes paramètres sur un log pris à partir d'un chariot et rejoué sur mon ordinateur depuis mon bureau.

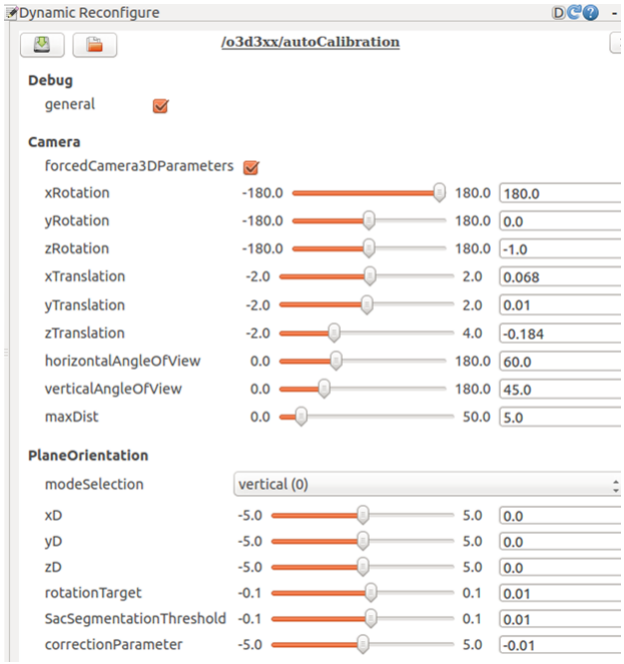

FIGURE 4.2 – Capture d'écran de l'outil Dynamic Reconfigure de RQT

4.2 PCL

Issue du même laboratoire à l'origine du Robot Operating System, la Point Cloud Library permet de fournir aux développeurs et roboticiens des outils facilitant le travail avec les nuages de points.

FIGURE 4.3 – Logo de la Point Cloud Library

4.2.1 Introduction

Un nuage de point est une structure de données utilisée pour représenter une collection de points multi-dimensions dans le traitement de données 3D. En plus de coordonnées spatiales (x,y,z) chaque points peut également contenir des informations sur la couleur(RGB, HSV...).L'acquisition de ses données se fait à l'aide de capteurs tels

qu'une caméra acoustique un scanner laser ou, comme pour mon stage, une caméra ToF.

4.2.2 Application

PCL [6] met à disposition une multitude de fonction allant de la simple copie et concaténation de nuages de points jusqu'à la segmentation. Et grâce à ROS il est facile et rapide d'obtenir un affichage en temps réel des nuages de points grâce à l'outil RQT. En effet mon tuteur avait écrit un programme regroupant plusieurs filtres spatiaux et leurs paramètres et pouvant aussi agir sur les coordonnées de la caméra 3D. Cela m'a permis d'avoir une première impression et une réelle compréhension de ses différents filtres, puisque je pouvais voir en temps réel l'effet qu'à chaque paramètre sur le nuage de points résultant.

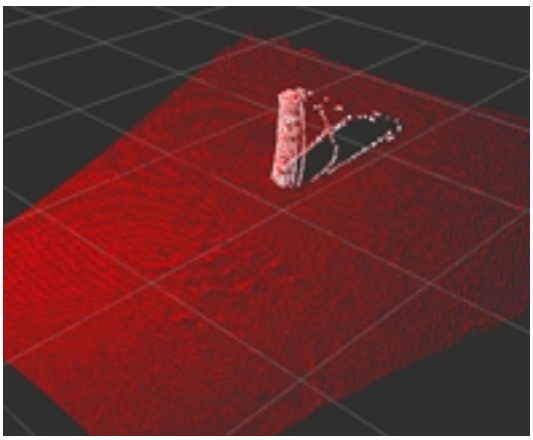

FIGURE 4.4 – Résultat du filtre médian sur la figure 3.6

Pour la suite de mon stage je me suis inspiré du code de mon tuteur et également des tutoriels de Point Cloud Library, pour créer une méthode permettant la détection du sol et la calibration de l'angle de tangage de la caméra. Cet algorithme est décrit dans le chapitre Algorithme d'auto-calibration.

PCL et ROS m'ont permis de pouvoir tester mes algorithmes de façons rapide, néanmoins la partie la plus longue pour moi était de créer une librairie en cpp et séparer ROS de mon code. Étant débutant dans ce langage et ne connaissant pas les règles d'usage de codage de la compagnie il m'a fallu beaucoup d'effort d'adaptation.

Chapitre 5

Outils de développement

5.1 GIT

Le département de Recherche et Développement de Balyo étant composé de plusieurs équipes qui travaillent toutes simultanément sur le développement du SDK. De la création d'une nouvelle fonctionnalité par l'équipe software robot à son implémentation dans l'éditeur de circuit, plusieurs étapes sont nécessaires. Il est ainsi impératif pour éviter les conflits et permettre un travail en parallèle efficaces d'utiliser GIT. GIT permet donc à toutes les équipes de Balyo de travailler sur différentes versions en parallèle et de pouvoir partager les changements réalisés par une personne au reste de l'équipe. Il permet aussi à chaque utilisateur de modifier un fichier et de pouvoir revenir à une version antérieur et assure la sauvegarde de l'évolution des programmes.

Pour ma part après avoir installer un outil de cryptage de fichier sur mon ordinateur -cette période de l'année était marquée par plusieurs attaques de type ransomware- j'ai pu obtenir de GIT la version du SDK sur laquelle j'allais travailler pendant la durée de mon stage. Sur les conseils de mon tuteur j'ai créé une branche où je pouvais modifier le code déjà existant en toute liberté et développer mon propre code.

5.2 RedMine

Après développement d'une nouvelle fonctionnalité par l'équipe Robot ou Tools et Supervision, l'équipe validation doit vérifier qu'aucun bug n'est rencontré lors de son implémentation finale à bords des chariots. Ils conduisent donc une série de tests dans le Paddock (zone de test de Balyo).

Lorsqu'un problème est détecté il est signalé au reste de l'équipe R et D grâce à RedMine, un outil web qui permet de répertorier les tickets selon leur priorité, la version du SDK concerné et également l'attribuer à une personne qui sera chargée de le résoudre.
Durant ma période de stage j'ai eu l'occasion de résoudre des tickets de traductions de descriptions de variables dans un fichier xml, en développant un script python qui le faisait automatiquement à partir de d'un fichier texte contenant la traduction des va-

leurs correspondantes. Mais aussi de résoudre des problèmes de détection qu'a rencontré l'équipe validation avec la caméra 3D en l'absence de mon tuteur.

Chapitre 6

Algorithme autocalibration

Afin de pouvoir se servir des données acquises par la caméra 3D, il est nécessaire de connaître ses coordonnées aussi bien en translation qu'en rotation. Ces coordonnées connues il est possible alors d'effectuer un changement de repère vers le référentiel du chariot et pouvoir réaliser des détections de palettes, d'espaces vide et d'obstacle.

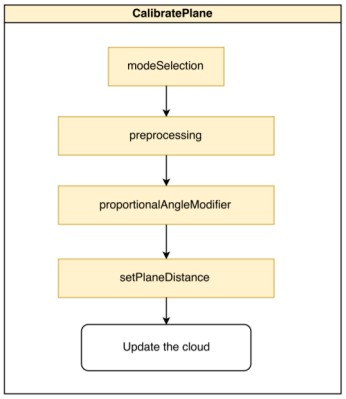

FIGURE 6.1 – Algorithme d'autocalibration

6.1 Introduction

La calibration est donc une étape cruciale pour le bon fonctionnement du chariot. Une mauvaise calibration cause une multitude de problèmes. J'ai fait des tests où en introduisant délibérément une erreur de 3 degré sur l'orientation de la caméra 3D équipée sur les fourches d'un chariot K-Matic de Linde, le chariot poussait les palettes et les faisait tomber si on l'arrêtait pas manuellement.

La méthode que mon tuteur et moi avant décidé d'utiliser est de déduire les orientations de la caméra à partir de la détection de plans normal à chaque axe. En retrouvant les équations d'un plan dans le repère de la caméra et connaissant son équation dans le repère du chariot, on retrouve ainsi les coordonnées de la caméra 3D.

Une solution plus simple aurait été de rajouter une centrale inertielle au support de la caméra mais l'ajout de ce capteur n'a pas été envisagé lors de la conception du kit Balyo. Aussi la précision des centrales inertielle à bas coût reste faible.

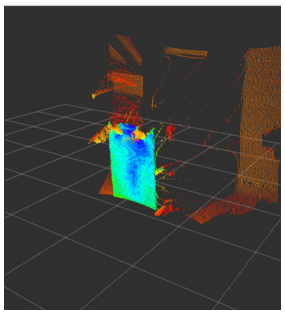

FIGURE 6.2 – Nuage de points brute lors de la calibration

6.2 Filtre Pass-Through

Après avoir positionné une surface plane normal à l'axe x du chariot, la première opération qu'effectue mon algorithme est un filtrage de l'espace. J'utilise un filtre pass-through pour ne garder que la zone où se trouve le plan d'intérêt. Ainsi l'algorithme a besoin de moins de puissance et est bien plus rapide puisque le nombres de points a traiter est beaucoup moins élevé.

Aussi cela me permet d'éliminer le risque de détection d'un faux plan dans le nuage de points. En effet la partie segmentation retourne le modèle de plan le plus grand dans le nuage de points au cas où il y'aurait plusieurs plans présents dans la scène. Il est possible ensuite de réduire encore plus le nombre de points en rajoutant un autre filtre spatial tel qu'un filtre à voxel. Ne voyant pas de net amélioration en vitesse d'exécution en ayant implémenté un filtre a voxel, je l'ai retiré de la version finale de mon algorithme.

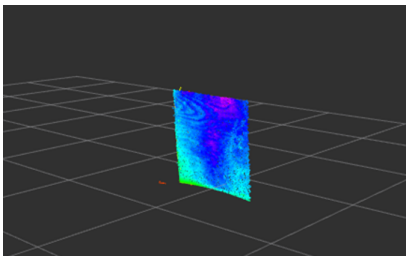

FIGURE 6.3 – Nuage de points après application du filtre passThrough

6.3 Segmentation

Une fois le nuage de points filtré vient l'étape de segmentation. Comme présenté ultérieurement, PCL permet à ses utilisateurs de bénéficier de nombres de fonctionnalités déjà implémentée par la bibliothèque et la communauté de développeur, l'une d'entre elle est la segmentation.

En créant un modèle de segmentation, PCL permet grâce à l'utilisation de la méthode itérative RANSAC (RANdom SAmple Consensus) de trouver les coordonnées du plan cible sous-forme de quaternions. On obtient ainsi une équation du plan observé.

$$ax + by + cz + d = 0 \tag{6.1}$$

où a, b, c et d sont les coordonnées du plan trouvé par la méthode RANSAC.

6.4 Quaternion

On accordera plus d'importance à l'application des quaternions par PCL qu'à leur définition, lourde, pour ce rapport. Un quaternion est un élément d'un espace vectoriels de dimension quatre. Il est représenté comme suit [7] :

$$q = m + ni + pj + rk \tag{6.2}$$

Avec m, n, p et r des réels et i, j et k des symboles satisfaisant la relation suivante.

$$i^2 = j^2 = k^2 = ijk = -1 \tag{6.3}$$

Les quaternions sont utilisé comme substitue aux matrices de rotations car ils permettent d'atténuer grandement l'utilisation de mémoire.

6.5 Proportionnel

Une fois le modèle de plan calculé, pour trouver les angles à appliquer en post-traitement aux données de la caméra pour les transférer au repère du chariot. Il faut minimiser les bons coefficients du plan. Ainsi pour trouver le plan qui a comme vecteur normal l'axe x du repère du chariot, nous recherchons a minimiser les paramètres b et c de du plan.

Pour se faire il suffit de trouver les bonnes rotations à appliquer autour de l'axe z et y. J'utilise un correcteur proportionnel qui réduit au fur et à mesure les valeurs adéquate du modèle. De manière empirique, mon problème se résume aux équations suivantes.

$$angle_z(t) = angle_z(t-1) + b.\alpha \tag{6.4}$$

$$angle_y(t) = angle_y(t-1) - c.\alpha \tag{6.5}$$

Avec alpha le coefficient du proportionnel choisi par l'utilisateur. Par défaut il est réglé à 0.01. L'algorithme estime avoir trouvé les bons angles quand b et c recalculé sont suffisamment proche de 0.

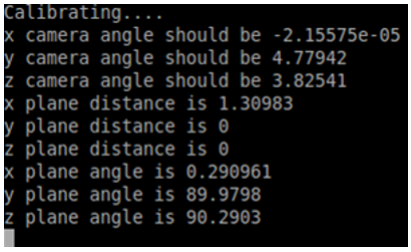

FIGURE 6.4 – Capture d'écran de la console

La raison pour laquelle j'utilise un proportionnel au lieu d'effectuer la transformation en un seule étape est que les données fluctuent, et que en utilisant un proportionnel, il est plus aisé de contrôler le résultat et de remplir les conditions fixées. L'algorithme se termine après avoir trouvé les orientations de la caméra qui place le plan à plus ou moins 0.3 degré du résultat optimal. Pour la détection de plan Horizontal et de profil, les angles sont trouvés comme suit.

Cas Horizontal

$$angle_y(t) = angle_y(t-1) + a.\alpha \tag{6.6}$$

$$angle_x(t) = angle_x(t-1) - b.\alpha \tag{6.7}$$

Cas profil

$$angle_z(t) = angle_z(t-1) + a.\alpha \tag{6.8}$$

$$angle_x(t) = angle_x(t-1) - c.\alpha \tag{6.9}$$

6.6 Calcul de distance

Une fois les orientations de la caméra trouvés et le plan réorienté, la dernière étape de l'algorithme consiste à trouver la distance séparant la caméra du nuage de points pour en déduire la position de la caméra dans le repère du chariot, suivant l'axe normal au plan observé. Ainsi une observation d'un plan vertical nous renseignera sur la

coordonnés x de la caméra. Cette coordonnée est calculée en effectuant la moyenne des points appartenant au modèle du plan.

$$d_x = \frac{\sum_{i=0}^{N} inliers(i)}{N} \tag{6.10}$$

Où N est le nombre de points appartenant au modèle de plan défini.(inliers)

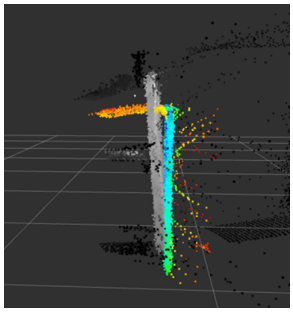

FIGURE 6.5 – Nuage de point après calibration (en couleur)

6.6.1 Bilan

Cet algorithme permet donc de retrouver les coordonnées de la caméra 3D de manière aisée. J'ai commencé par une version prototype qui dépendais de ROS pour finalement créé une librairie complètement séparée qu'il est possible d'utiliser hors de ROS.

Chapitre 7

Algorithme de détection d'obstacle.

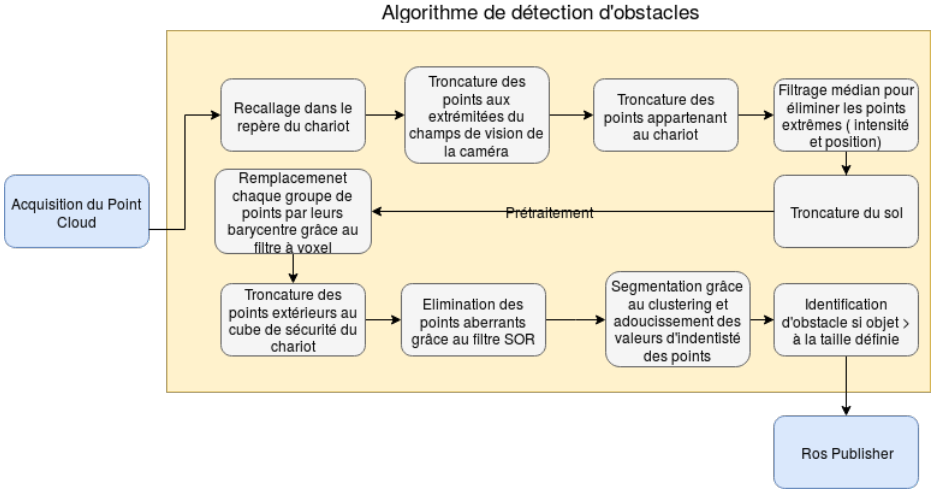

FIGURE 7.1 – Algorithme de détection d'obstacle

7.1 Introduction

Les chariots autonomes de Balyo disposent d'un laser de navigation qui leur permet de connaître leurs coordonnées dans leurs zones d'opération. Et même s'ils peuvent connaître la position des autres robots, il serait possible qu'ils effectuent leurs missions sans avoir à bord de capteurs de sécurité. Hélas il est trop dangereux de supposer qu'il n'y est aucun accident qui laisserait un objet sur le trajet du chariot d'autant plus que dans bon nombres de sites clients, les chariots autonomes de Balyo roule dans le même espace que des chariots classiques. C'est pour cela que Balyo équipe ses chariots d'une multitude de laser assurant une zone de sécurité autour d'eux, ce qui leur permet de manœuvrer qu'en l'absence d'obstacle sur leurs trajectoires.

Dans la continuité de l'innovation de l'entreprise et à fin d'améliorer la fiabilité des chariots, Balyo a décidé d'étudier l'utilisation d'une caméra 3D au lieu du laser rideau. En effet ce laser ne permet pas au chariot de savoir si l'obstacle qui a entraîné un arrêt est toujours présent devant lui.

7.2 Implémentation du capteur

Le principal défi quant à l'intégration de la caméra 3D sur les chariots réside dans sa position, son orientation et également le support qui doit la protéger et la fixée de manière stable. La communication avec la caméra 3D, elle, se fait par Ethernet.

Il a été choisi par l'équipe de recherche et développement de positionner la caméra 3D de la même manière que le laser rideau, c'est à dire en haut du robot, scrutant le sol à un angle avoisinant les 45°.

Durant mes essais sur le robot équipé d'une caméra fixée à l'aide d'un support à aimant, j'ai pu constater que l'orientation de la caméra pouvait augmenter le phénomène de points fantômes. La plupart d'entre eux appartenant au chariot, il est facile et sans risque de les supprimer en réduisant le champ de vision de la caméra. Un problème plus gênant est qu'en cas de présence d'un objet autour de la caméra (support de fixation), les données se retrouvent perturbés et inutilisables. La caméra doit donc être exposé et non protégé en cas de choc. C'est pour cela que ma proposition de mettre la caméra à un niveau plus bas du chariot et orientée horizontalement a été refusée, puisque trop fragile. Ce changement d'orientation aurait pu résoudre les problèmes engendrés par des sols irrégulier et qui engendre un faux positif en détection d'obstacles.

Un changement d'orientation de la caméra en cas de choc ou vibration si le capteur est mal fixé, entraîne également des erreurs. En effet une erreur de 1° suivant l'un des axes de rotations se traduit par une erreur de 1.7 cm tous les 1m. Plus la zone observée est loin et plus l'erreur est grande.

$$e_d = d * tan(e_a) \tag{7.1}$$

Où d est la distance entre la caméra et le point observé et e_a l'erreur angulaire.

7.3 Acquisition et traitement

Une fois le nuage de point acquis par le nœud de détection obstacle, il subit une série de filtrage et de transformation avant de conclure quant à la présence d'obstacles sur le trajet du chariot.

7.3.1 Changement de repère

La première étape consiste à reporter le plan de la caméra à celui du plan d'évolution du chariot. Cela se fait en renseignant les coordonnées adéquates de la caméra. Ces coordonnées dans un premier temps sont remplies manuellement mais par la suite il est envisagé d'utiliser un système d'autocalibration d'où le travail que j'ai réalisé auparavant.

7.3.2 Paramètres de sécurités

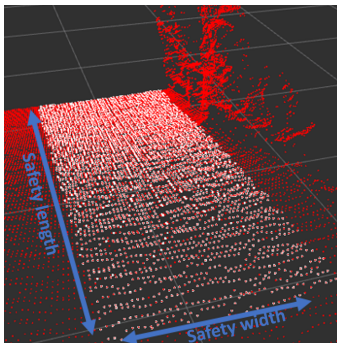

FIGURE 7.2 – Zone de sécurité

La rubrique Safety regroupe tous les paramètres relatifs à la taille de l'obstacle et à la zone où l'on considère qu'un objet entre dans l'espace du chariot.Les paramètres dans la rubrique Safety sont à régler selon le type de chariot et les exigences de sécurité (distance de détection et largeur par exemple).

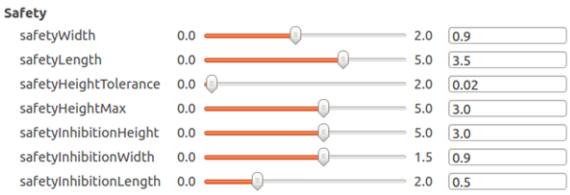

FIGURE 7.3 – Section safety dans RQT

SafetyWidth représente la largeur de la zone de sécurité (en blanc), elle correspond à la largeur du chariot. safetyLength représente la longueur de cette zone, c'est à dire à combien de mètre un obstacle peut être détecté. safetyHeightTolerance représente la hauteur minimale d'un élément du nuage de points pour qu'il soit considéré comme obstacle. safetyHeightMax la hauteur maximale d'un obstacle (le chariot peut passer en dessous au delà). Les 3 paramètres safety inhibitions constituent les dimensions du polyèdre qui entoure le chariot. Tous les points intérieur ne sont pas pris en compte dans la suite de l'algorithme.

7.3.3 Réduction du champs de vision

On procède ensuite à la partie externalRemoval qui permet de retirer les points appartenant au chariot et de tronquer les points fictifs que la caméra voit spécialement à la limite haute de son champs de vision.

La présence de ces points revient à une confusion de salves infrarouge émise par la caméra. En effet la caméra pense qu'il s'agit d'une salve émise à un temps plus récent et estime donc qu'un objet est présent devant elle.

7.3.4 Filtre médian

Vient ensuite le filtre médian qui va permettre de filtrer les points isolés et réduire le bruit. Il remplace un point dans un voisinage par la valeur médiane de ce voisinage. Une fenêtre de taille variant entre 3 et 5 constitue un bon compromis puissance de calcul/qualité du filtre. Une fenêtre trop grande entraîne un ralentissement conséquent sur l'ensemble du traitement.

7.3.5 Filtre Statistical Outlier Removal

Le filtre SOR permet de retirer tous les points qui sont considérés aberrant selon les paramètres définis (nombres de voisins et l'écart type). Pour se faire la distance moyenne entre chaque point du nuage et ses voisins est calculé et élimine les points dont la distance est supérieur à la somme de la moyenne et l'écart type. Il vient compléter le filtre médian. A noter que ce filtre peut être ignoré car il consomme beaucoup de puissance de calcul comparé à l'utilité qu'il apporte. Il fait passer le nombre d'images par seconde de l'algorithme de 75 environ à 50.

7.3.6 Filtre à Voxel

Un voxel est l'équivalent d'un pixel en 3D. Le filtre à voxel permet de subdiviser l'espace en voxel, c'est à dire en cubes de dimensions choisies. Tous les points contenus dans un cube sont remplacer par leur barycentre. Cela permet de réduire grandement le nombres de points sans pour autant perdre en précision et en informations en fonction de la taille de cubes choisie. Plus la taille est grande moins on obtient de points en sortie et plus on perd d'informations.

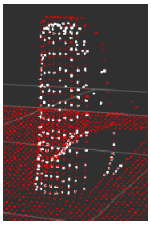

FIGURE 7.4 – Résultat du filtre à Voxel

7.3.7 Segmentation

Enfin vient l'étape de segmentation. La méthode d'arbre k-d est utilisé pour pouvoir regroupé les points proches en objet.

Un arbre k-d est une structure de donnée utilisé pour le traitement de points appartenant à un espace à k dimensions. Dans le cas de mon stage et des données que je traitent il s'agit de points appartenant à un espace de dimension trois. Un arbre k-d est un arbre binaire. Il est utile dans la recherche des voisins les plus proche d'un point spécifique.[8]

Un groupe de points est considéré comme objet si dans un voisinage donné en trouve un nombres de points supérieurs au seuil fixé. Ainsi un objet est signalé comme obstacle si il est sur le chemin du chariot et qu'il dépasse une certaine hauteur fixée par l'utilisateur.

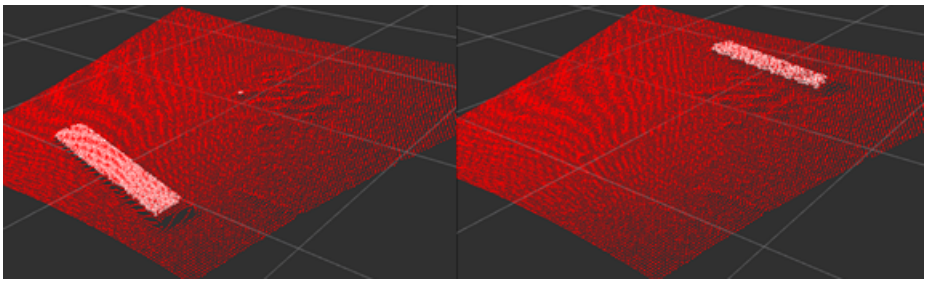

FIGURE 7.5 – Obstacle de 7cm de hauteur vu à 1m puis 3m du chariot

Conclusion

Ce stage m'a permis de mettre en application les connaissances acquises au cours de mon cursus à l'ENSTA Bretagne. En effet j'ai eu l'occasion d'utiliser le Robot Operating System pendant plusieurs de mes projets à l'école. J'ai également beaucoup progressé en algorithmique en général et appris les bonnes règles d'usage de codage en cpp et en groupe avec GIT.

Aussi en ayant fait parti de l'équipe Balyo durant cette période important de la société, j'ai pu voir de prés le fonctionnement d'une entreprise et son passage de start-up à l'introduction en bourse.

Les perspectives d'améliorations consistent principalement à l'implémentation de l'algorithme d'auto-calibration dans une version du SDK que le client pourra utiliser via l'interface graphique du robot. Quant à l'algorithme de détection d'obstacle, je pense qu'un changement de capteur est nécessaire pour éviter les défauts de la caméra ToF.

Bibliographie

[1] Communiqué de presse de l'introduction en bourse de Balyo disponible à l'adresse suivante :
https ://www.balyo.fr/Actualites/Presse/Large-success-of-the-Initial-Public-Offering-of-Balyo-on-Euronext

[2] Localisation et cartographie simultanées pour un robot mobile équipé d'un laser à balayage : CoreSLAM, Oussama El Hamzaoui.

[3] Appareils de levage et chariots de manutention, chapitre 6, Christian LE GALL.

[4] Time-of-Flight Camera – An Introduction, Larry Li.

[5] Lien vers la page d'aide de ROS :
http ://wiki.ros.org/ROS/Tutorials

[6] Lien vers la page d'aide de PCL :
http ://pointclouds.org/documentation/tutorials

[7] Quaternions et Rotations, Michel Llibre

[8] An intoductory tutorial on kd-trees, Andrew W. Moore.

Table des figures