Implémentation d'un SLAM Monoculaire pour un robot d'intérieure

Tuteur : Fabian LAPOTRE
Encadrant : Benoit ZERR
Addresse : Station F, 55 Boulevard Vincent Auriol, Paris 13

28 août 2018

Remerciement

En termes de reconnaissance et de considération, j'adresse mes innombrables remerciements à la Direction et au corps professoral de l'Ecole Nationale Supérieure des Techniques Avancées Bretagne (ENSTA Bretagne) et plus particulièrement à Monsieur le professeur encadrant Benoit ZERR qui m'a accompagné tout au long de mon cursus à l'ENSTA Bretagne.

Mes remerciements sont aussi présentés à tous les membres du jury pour l'intérêt qu'ils vont porter à ce travail afin de le passer en revue et émettre leurs précieux conseils.

Je tiens également à exprimer ma gratitude et ma haute considération à Monsieur Fabian LAPOTRE qui m'a permis d'effectuer mon stage chez Camtoy et qui n'a pas hésité à me faire part de l'ensemble des connaissances techniques que j'ai pu acquérir tout au long de cette expérience professionnelle.

Enfin, je ne peux clore cette page sans remercier tous ceux qui m'ont aidé, quelle que soit l'ampleur de la tâche, dans l'accomplissement de ce travail ; qu'ils soient des membres de la famille, des amis, des collaborateurs de mon entreprise d'accueil ou d'autres.

Table des matières

1 Contexte **13**
1.1 Presentation de l'entreprise . . . 13
1.2 Présentation de Laika . . . 14

2 SLAM **17**
2.1 Définition . . . 17
2.2 Différents SLAM . . . 17
2.3 Choix . . . 19

3 Théorie **21**
3.1 ORB . . . 21
3.1.1 Zones d'intéret, points d'intérêt . . . 21
3.1.2 Détecteurs . . . 22
3.1.2.1 Détecteur FAST . . . 22
3.1.2.2 Détecteur BRIEF . . . 23
3.1.2.3 Detecteur ORB (Oriented FAST and Rotated BRIEF) . . . 24
3.2 ORB-SLAM . . . 25
3.2.1 Présentation de l'ORB-Slam . . . 25
3.2.2 Présentation des diverses tâches . . . 26
3.2.2.1 La tâche de suivi . . . 26
3.2.2.2 Cartographie locale . . . 27
3.2.2.3 Fermeture de boucle . . . 27

4 Implémentation **29**
4.1 Outils . . . 29
4.1.1 GIT . . . 29
4.1.2 ROS . . . 30
4.1.2.1 Definition . . . 30
4.1.2.2 Fonctionnement . . . 30
4.1.2.3 Simulation . . . 32
4.2 Performances . . . 33
4.2.1 Test sur système AMD . . . 33

4.2.2 Test sur système ARM 36

Table des figures

1.1 Equipe Camtoy 13
1.2 Robot Laika avec base de rechargement et Tracker 14
1.3 Diagramme de fonctionemment du robot Laika 15

2.1 Graphe de comparaison des différentes méthodes SLAM. 20

3.1 Détection de coins par FAST 23
3.2 Divers résultats du détecteur BRIEF selon la variation par rotation 24
3.3 Résultat d'une détection avec ORB 25
3.4 Tâches réalisées par l'ORB-SLAM 28

4.1 Logo de ROS 30
4.2 Schèma de communcation avec de le Master 31
4.3 Communication des noeuds par topic 31
4.4 Phase de matching les points avec la simulation du robot Laika 32
4.5 Détéction d'amers avec la simulation du robot Laika 33
4.6 Phase de matching de points dans l'algorithme d'ORB-Slam 34
4.7 Amers obtenus grâce au détecteur ORB dans l'ORB-SLAM 35
4.8 Résultat de l'algorithme d'ORB-SLAM2 35

Résumé

Dans le cadre de notre cursus d'élèves-ingénieur à l'École Nationale Supérieure des Techniques Avancées Bretagne (ENSTA Bretagne), nous devons effectuer un stage de fin d'étude. Celui-ci doit nous permettre de prendre conscience du réel travail de l'ingénieur et nous confronter aux problèmes techniques et managériales que rencontre quotidiennement ce dernier. J'ai choisi d'effectuer mon stage dans l'entreprise Camtoy où nous avons participé au développement de leur produit Laika qui est un robot compagnon de chiens.

Ce rapport présente le travail que j'ai réalisé, et où je suis principalement intérvenu sur le mode autonome de Laika et particulièrement dans la localisation de celui-ci tout au long de sa mission. En navigant, le robot doit être capable de se localiser en intérieur. Nous devons donc utiliser un algorithme qui va permettre au robot de se localiser et ceci en n'utilisant que les capteurs présents dans le robot, c'est à dire la caméra, l'odomérie et des capteurs infrarouges. L'algorithme choisi est un algorithme de SLAM (Simultaneous localization and mapping) appelé ORB-Slam de type monoculaire c'est à dire qu'il n'utilise qu'une seule caméra. Après le choix de ce Slam il a fallu l'implémenter dans différentes architectures de type AMD et ARM, chacun utilisant un système d'exploitation différent et proposant différentes ressources.

Abstract

At the end of our university course at ENSTA Bretagne (Ecole Nationale Supérieure des Techniques Avancées), we must perform an internship end of study in order to become aware of the real work of the engineer and to confront us with the daily problems of the field, both technical and managerial. I have chosen to do my internship in Camtoy company where I participated in the development of their principal product Laika which is a dog companion robot.

This report presents the work that I realized and the subjects on which I mainly intervened namely the autonomous mode of Laika and its localization throughout its mission. In fact, while navigating, the robot must be able to locate itself inside. We must therefore use an algorithm that will allow the robot to locate and this by using only the present sensors in the robot : I.e : the camera, odometry and infrared sensors.

The algorithm chosen is a Simultaneous localization and mapping (SLAM) algorithm called ORB-Slam of the monocular type which uses only one camera. After choosing this Slam, it was necessary to implement it in different architectures of AMD and ARM type, each using a

different operating system and offering different resources.

Introduction

Dans le cadre de la validation de la dernière année à l'ENSTA Bretagne, nous sommes amenés à réaliser un stage de fin d'études dont le but est de réaliser le travail d'ingénieur. Cette expérience va donc nous permettre de valider les connaissances théoriques acquises tout au long de l'année scolaire. J'ai ainsi réalisé ce stage dans l'entreprise Camtoy ou j'ai dû réaliser plusieurs tâches pour mener à bien le projet qui m'a été confié.

Dans le cadre de ce projet, nous devions implémenter un algorithme dans le robot de l'entreprise qui doit lui permettre de se localiser à l'intérieur d'une maison et cela en n'utilisant que les capteurs présents dans celui-ci. La localisation étant une problématique majeure dans le domaine de la robotique, il a fallu choisir un algorithme robuste et qui permet au robot de connaitre sa position avec une petite marge d'erreur. L'algorithme que nous avons implémenté est un algorithme de SLAM (Simultaneous localization and mapping). Il sera développé plus en détails dans les parties suivantes.

Dans ce rapport, nous allons commencer par présenter l'entreprise Camtoy et le contexte dans lequel s'inscrit notre sujet. Nous allons ensuite présenter la technique de SLAM, différents algorithmes de SLAM et le choix du SLAM retenu. Ensuite nous introduirons les outils utilisés par ce dernier et son fonctionnement. Et pour finir nous parlerons de l'implémentation de celui-ci dans un système AMD puis dans deux cartes embarquées différentes.

Chapitre 1

Contexte

1.1 Presentation de l'entreprise

Camtoy est une start-up française basé à la Station F, le plus grand incubateur de start-up d'Europe avec plus de 1000 start-up. Elle a été fondée par quatre personnes, Thomas, Marvin, Samy et Fabian passionné pas les chiens. L'idée de leur robot leur est venue lors d'un cours de leur école. Ils ont réalisé des statistiques qui leur ont permis de voir que plus de 30% des chiens restent seuls à la maison et beaucoup de ces chiens ont des comportements négatifs. Cette idée consiste à créer un robot compagnon de chien qui occupe les chiens lorsque leur maître est absent. Après leur victoire au concours stratégie Océan Bleu, ils ont décidé de créer la start-up.

L'équipe de Camtoy est composée de quatre fondateurs en plus de quelques stagiaires qui s'occupe de la partie ingénierie

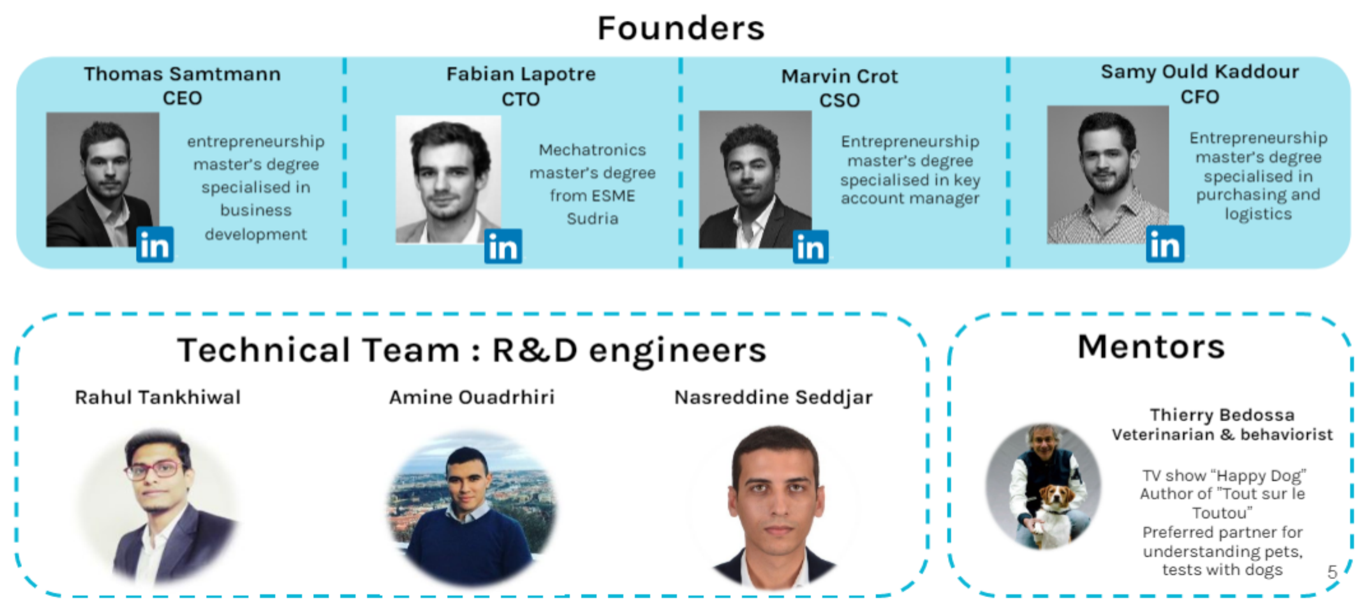

Figure 1.1 – Equipe Camtoy

1.2 Présentation de Laika

Laika est un robot mobile dont l'architecture est faite de façon à ce que les différents chiens n'arrivent jamais à le soulever ni à le prendre dans leur bouche. Son nom a

été choisi en référence au premier chien à être monté dans l'espace.

Il peut être utilisé suivant deux modes. Un premier mode ou c'est l'utilisateur qui le commande via un téléphone portable Smart Phone et un deuxième mode autonome. Dans les deux cas, la navigation du robot se fait grâce à une caméra ainsi que des capteurs infrarouges. On trouve également dans le robot un microphone qui va permettre à l'utilisateur de parler à son chien ainsi qu'un distributeur de friandises qui va lui permettre de le récompenser. L'utilisateur pourra ainsi garder un œil sur son chien.

Le robot sera accompagné d'une base de rechargement ainsi que d'un tracker. Le tracker doit être installé dans le collier du chien et va permettre au robot de détecter les mouvements du chien grâce aux capteurs intégrés dans celui-ci. Il va également permettre au robot de connaître la position du chien dans un périmètre lors de sa navigation autonome.

Figure 1.2 – Robot Laika avec base de rechargement et Tracker

Lors de la navigation autonome, le robot doit sortir de sa base de rechargement, et doit se diriger vers le chien où il a détecté sa présence dans un périmètre. Il doit ensuite l'identifier grâce à un algorithme de Machine Learning pour jouer avec lui et enfin il doit être capable de revenir à sa base de rechargement.

Mon stage intervient dans les parties de réalisation d'une carte fiable par le robot et le chargement de celle-ci pour qu'elle soit utilisée lors de la phase de navigation vers le chien et le retour vers la base de rechargement.

L'organigramme suivant explique le fonctionnement du robot Laika :

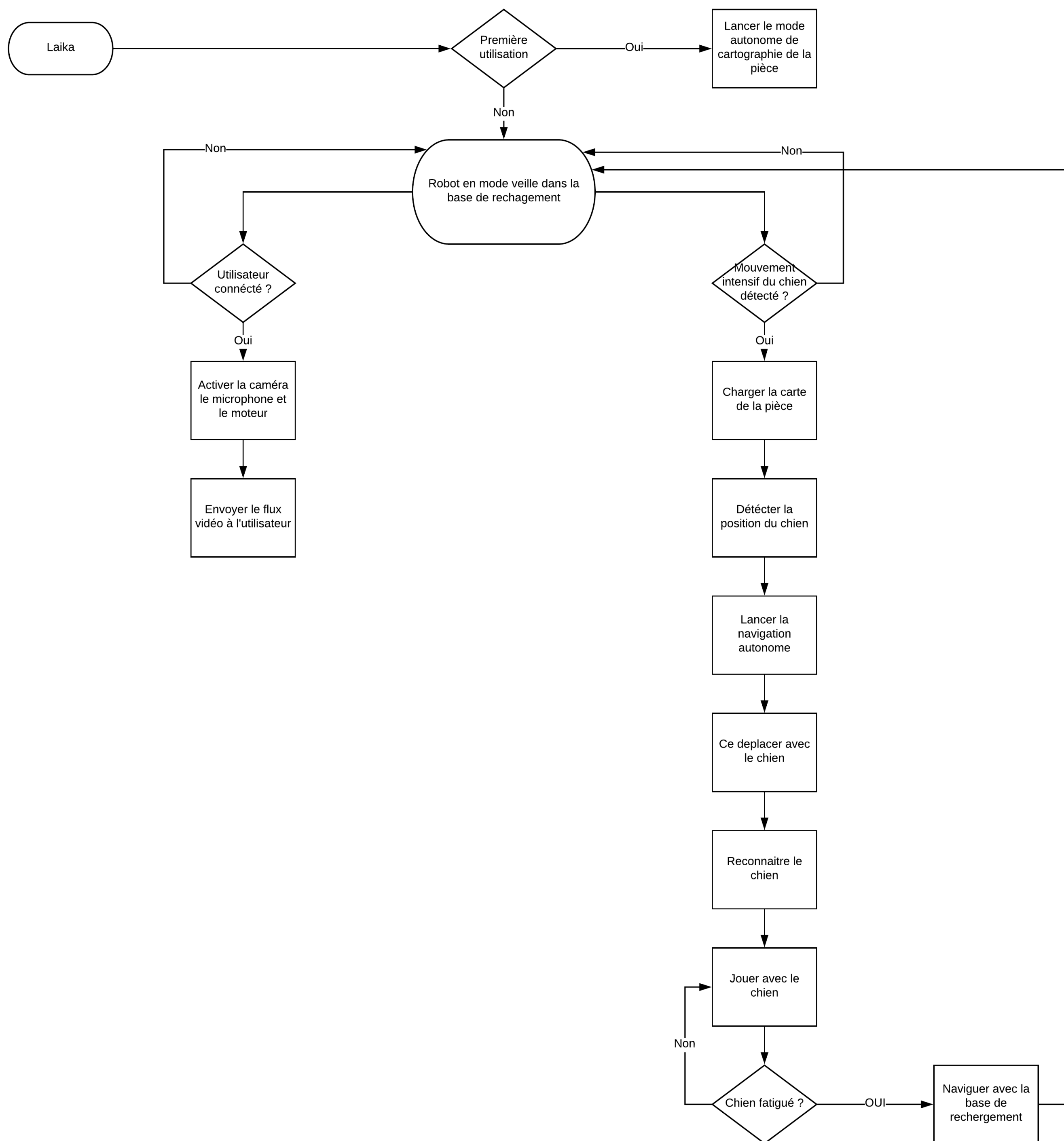

FIGURE 1.3 – Diagramme de fonctionemment du robot Laika

Le modèle d'évolution de notre robot est décrit par 6 paramètres : ses coordonées sur la carte (x, y, z) et ses trois angles d'Euler $(roll, pitch, yaw)$ par rapport à un repère externe. Notre robot va se déplacer dans un plan à deux dimensions. Les variables cinématiques utilisées donc dans notre cas sont (x, y, θ) où x et y représentent ses cordonnées cartésiennes et ϑ

son orientation angulaire. Son vecteur d'état est donc $\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$ (1.1). Le modèle probabiliste de l'évolution du robot utilise la probabilité conditionnelle $p(s_t|u_t, s_{t-1})$(1.2) où $s_t$ets_{t-1} sont les vecteurs de position du robot à l'instant t et t-1 et u_t le vecteur de la commande de contrôle. Lors du déplacement du robot, il accumule des erreurs de translation et une dérive. Ce modèle probaliste nous permet ainsi de représenter une zone où le robot a une grande probalité de se trouver. Il existe deux modèles d'évolution : un modèle de vitesse et un modèle d'odomètrie. Notre robot utilisant l'odomètrie pour se déplacer, nous allons décrire le modèle par odométrie. Ce modèle décrit la position du robot à partir d'une position initiale et des distances parcourues par les roues droite et gauche $d_r$et d_l. Il est donné par les équations suivantes : $f(x', y', \theta') = \begin{pmatrix} x' = x + d_c cos(\theta + \frac{\delta_\theta}{2}) \\ y' = y + d_c sin(\theta + \frac{\delta_\theta}{2}) \\ \theta' = \theta + \delta_\theta \end{pmatrix}$ (1.3) avec $\left\{ \begin{array}{l} d_c = \frac{d_r + d_l}{2} \\ \delta_\theta = \frac{d_r - d_l}{2} \end{array} \right\}$(1.4)

Chapitre 2

SLAM

2.1 Définition

La localisation est une problématique majeure dans le domaine de la robotique puisqu'en mode autonome, tout robot doit connaitre sa position. En extérieur, ce problème se pose moins du fait de l'utilisation des GPS, en revanche en intérieur et dans les milieux sous-marins par exemple, le problème est encore posé. Une des approches permettant de se localiser est d'utiliser un système de marqueurs artificiels ou des dispositifs actifs comme des balises qui permettront au robot de se localiser par rapport à elles. Et une autre approche est d'utiliser des algorithmes qui permettent de réaliser cette tâche. Dans notre cas, puisque la première approche nécessite des configurations chez le client, nous allons utiliser la deuxième approche. Parmi les algorithmes de localisation qui ne nécessitent pas d'informations préalables sur l'environnement on trouve l'algorithme de SLAM.

Le SLAM (Simultaneous Localization And Mapping) consiste à créer une carte précise d'un endroit et se localiser à l'intérieur de cette carte. Son but est de rendre le robot complétement autonome. C'est à dire qu'en plaçant un robot dans un endroit inconnu, celui-ci doit être capable de connaître sa position dans la carte qu'il est en train de construire grâce aux données qu'il acquière avec les capteurs qu'il intègre. En prenant les problématiques de mapping et de localisation séparés, elles sont facilement résolvables. Mais le défi du SLAM est de pouvoir réaliser les deux tâches en parallèle, puisque dans le SLAM, ces tâches dépendent l'une de l'autre. Le SLAM est principalement basé sur les relations statistiques des éléments mesurés par des capteurs comme des capteurs laser, ultrason, ou encore une caméra.

2.2 Différents SLAM

Il existe différentes approches pour réaliser un SLAM, celles-ci diffèrent selon le type de capteur utilisé et selon le type de carte à produire.

En ce qui concerne le type des capteurs, les plus utilisés sont des capteurs de distance tel que des télémètres laser, ultrason ou sonar. L'utilisation de ces capteurs est dûe au fait que

leurs algorithmes n'ont besoin que de faibles ressources mémoire, ce qui les rend facilement implémentables dans des systèmes embarqués. Mais à partir des années 2000 et avec le développement des caméras numériques et des ordinateurs, le SLAM à partir de caméra appelé SLAM visuel ou SLAM par vision a fait son apparition. Cette apparition est aussi dûe au fait que les caméras numériques sont des dispositifs simples à mettre en oeuvre et les images de la caméra offrent plus d'informations que les capteurs télémétriques surtout en terme d'apparence. Le principal inconvénient de ce SLAM visuel est le fait qu'il soit gourmand en terme de mémoire et nécessite plus de temps de calcul.

SLAM	
	Capteurs
ORB-SLAM	Caméra monoculaire Caméra stéréo Caméra de profondeur
PTAM	Caméra monoculaire
LSD-SLAM	1
Mrpt_ekf_slam_2d (EKF)	Laser Odométrie
Fast-SLAM	Odométrie
Pose_ekf_SLAM	Caméra
Robot_pose_ekf	Odométrie IMU sensor Caméra (Combinés)
vslam	Caméra stéréo Caméra monoculaire

Table 2.1 – Tableau de différents algorithmes de SLAM avec capteurs utilisés

En ce qui concerne les cartes à produire, nous distinguons deux approches : une approche métrique et une approche topologique [7].Dans le cas de l'approche métrique on trouve deux solutions. La première est une solution par grille d'occupation, où l'environnement est modélisé sous forme de grille et chaque grille a un état, soit libre soit occupé. La deuxième solution représente l'environnement sous forme d'amer. Chaque amer est un point 3D facilement repérable et dont on connait les coordonnées.

L'approche topologique quant à elle, représente l'environnement sous forme d'un graphe de lieux dont les arrêtes représentent les relations entre les lieux. Dans cette approche, nous utilisons le plus souvent une caméra.

2.3 Choix

Le choix de l'algorithme de SLAM que nous comptons utiliser pour la suite du projet est principalement basé sur les capteurs intégrés dans le robot qui sont une caméra et des capteurs infrarouges. C'est pour cela que le choix naturel est un SLAM Visuel Monoculaire, c'est à dire qu'il utilise une seule caméra.

Le SLAM Visuel consiste à faire une représentation des images du flux vidéo pour en tirer des informations pertinentes appelées primitives comme des couleurs, des textures, des points d'intérêts, etc. Ces primitives peuvent ainsi être comparées entre elles et en ce qui concerne les points d'intérêts, nous pouvons également comparer les distances entre elles. En suivant ces primitives dans plusieurs images successives, Nous arrivons à calculer les positions géométriques de points 3D (cas de l'approche métrique) ou bien à représenter l'apparence d'un lieu grâce à ces points (cas de l'approche topologique).

Dans le cas de l'approche métrique, le principe de fonctionnement de la localisation se base sur le fait d'observer un même point 3D à des instants différents, cela permet d'en déduire sa position 3D par triangulation. Nous construisons ainsi une carte avec les différents points au fur et à mesure et on se localise par rapport à elle. La limite de cette approche est le fait que plus l'endroit est grand plus on a de la dérive.

Dans le cas de l'approche topologique, cela revient à résoudre un problème de reconnaissance d'images car deux images similaires peuvent être prises à partir d'un même endroit. Ainsi la connaissance de l'endroit permet la localisation du robot. Cette approche permet de gagner en robustesse [7].

Il existe également des travaux qui combinent les deux approches. Dans ces travaux-là, des amers sont créés à partir du flux vidéo et on enregistre les images précédentes. Ainsi le processus de localisation est fait à partir des amers et des images.

Pour notre choix de SLAM, nous avons choisi d'utiliser une méthode qui combine les deux approches car celle-ci est plus robuste et nous obtenons une dérive plus faible qu'avec l'utilisation d'une seule approche. Nous trouvons plusieurs SLAM qui combinent les deux approches. Je dois ainsi choisir parmi ces SLAM celui le mieux adapter à notre cas et avec lequel on obtient la meilleure trajectoire.

Le Graphe ci-dessous montre une comparaison entre plusieurs algorithmes de SLAM et leur capacité à suivre une trajectoire définie[8].

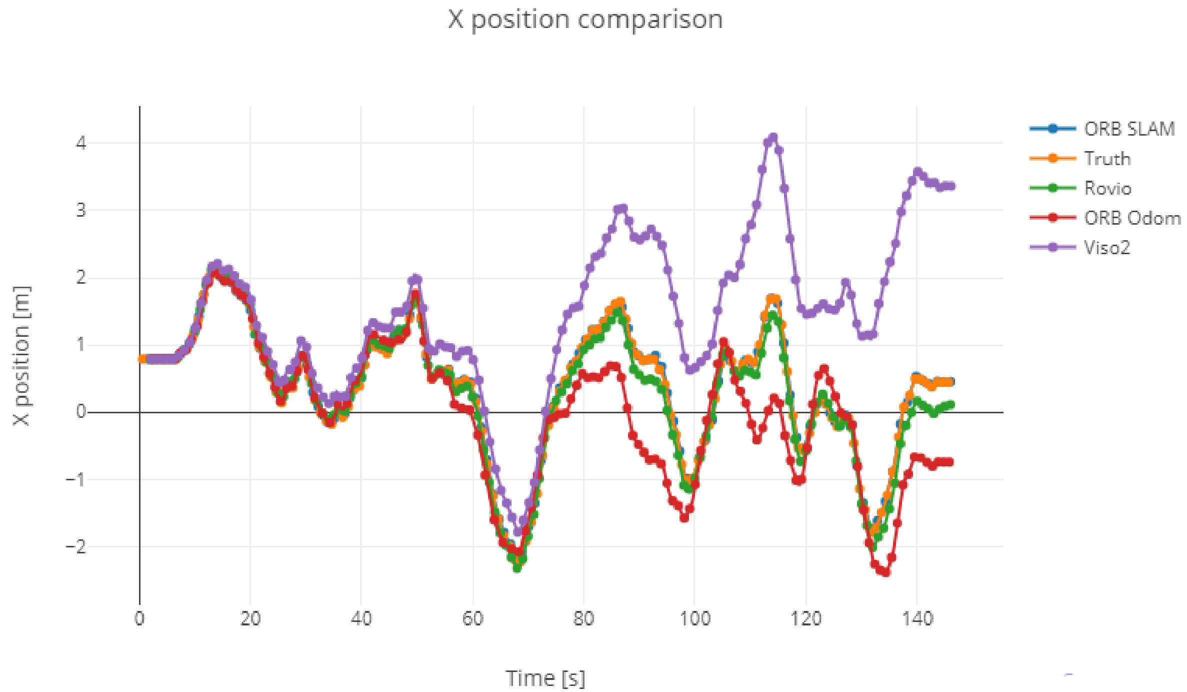

FIGURE 2.1 – Graphe de comparaison des différentes méthodes SLAM.

La couleur orange représente la vraie trajectoire et l'algorithme qui la suit le mieux est l'ORB-SLAM. L'ORB-SLAM suit tellement bien cette trajectoire que ces points sont presque confondue avec celle-ci.

Notre choix a été donc d'utiliser l'algorithme de l'ORB-SLAM même s'il prend un peu plus de temps d'execution.

Dans la suite, nous allons nous intéressé lu décteur ORB qu'utilise l'ORB-SLAM au fonctionnement de ce SLAM.

Chapitre 3

Théorie

Dans cette partie, nous allons d'abord commencer par une définition des zones d'intérêts ainsi que des points d'intérêt, ensuite nous allons présenter les détecteurs FAST et BRIEF sur lesquels est basé le détecteur ORB et finir par une présentation de l'ORB-SLAM.

3.1 ORB

3.1.1 Zones d'intéret, points d'intérêt

Zones d'intéret

Les zones d'intérêts en traitement d'images sont les zones jugées "intéressantes" dans une image. Elles sont utilisées lors de l'analyse de l'image. On représente ces zones sous forme de courbes continues, de points ou bien de rectangles selon d'algorithme de détection utilisé. Les zones d'intérêt peuvent être soit des contours, soit des points d'intérêt ou encore des régions d'intérêt.

Points d'intérêt

En ce qui concerne les points d'intérêt, ils représentent des points particuliers des contours selon des caractères précis. Ces points correspondent à des doubles discontinuités de la fonction d'intensité. La plupart des algorithmes pour détecter ces points d'intérêts utilisent une analyse locale de l'image, mais ils diffèrent en fonction de l'opérateur de dérivation utilisé, qui s'intéresse soit à la discontinuité de la fonction de réflectance, aux discontinuités de profondeur ou aux variations de texture comme c'est le cas des coins. Les avantages des points d'intérêt par rapport aux contours sont [14] :

1. Sources d'informations plus fiables que les contours car plus de contraintes sur la fonction d'intensité
2. Robuste aux occultations (soit occulté complètement, soit visible).
3. Pas d'opérations de chainage.
4. Présents dans une grande majorité d'images.

3.1.2 Détecteurs

Dans cette partie, je vais m'intéresser à la détection des points d'intérêt. Cette approche permet d'avoir une représentation 2D de l'image. Il nous est donc indispensable d'utiliser des détecteurs fiables et suffisamment robustes aux changements de perspective et ainsi détecter les mêmes points dans des images avec différents points de vues. Dans le cadre de ce rapport, nous allons parler des détecteurs FAST, BRIEF et ORB qui n'est que la combinaison des deux derniers.

Les propriétés d'un bon détecteur sont : [1]

- Répétabilité : le point doit apparaître aux mêmes endroits quelque soit la déformation.
- Représentativité : les points doivent être le plus nombreux possible.
- Efficacité : le détecteur doit être rapide à calculer (cf SURF, FAST)

Et ceci malgré le fait que les propriétés de répétabilité et de représentativité ne sont pas indépendants en eux.

Dans ce tableau, voici une liste des detecteurs les plus utilisés [2] :

	Invariances	
Détecteurs	rotation	Chg d'échelle
Harris [5]	X	
Harris-Affine [8]	X	X
DoG [6]	X	X
LoG [11]	X	X
Fast [9]	X	
Fast-hessian [1]	X	X

Table 3.1 – Détecteurs de points d'intérêt

3.1.2.1 Détecteur FAST

Le détecteur FAST est souvent utilisé dans la détection de points d'intérêt pour ces propriétés de calcul. Ce détecteur prend en paramètre le seuil d'intensité d'un pixel central avec celui des pixels qui l'entourent et ceci de façon circulaire mais sans prendre en compte l'orientation de ces derniers. Ceci revient à dire que celui-ci sélectionne les points dont le voisinage circulaire présente des plages contiguës assez longues de points significativement plus clairs (resp. plus sombres). On dit qu'un pixel est clair si son intensité est plus grande que celle du pixel central avec un certain seuil et sombre si son intensité est plus faible du pixel central avec ce même seuil.

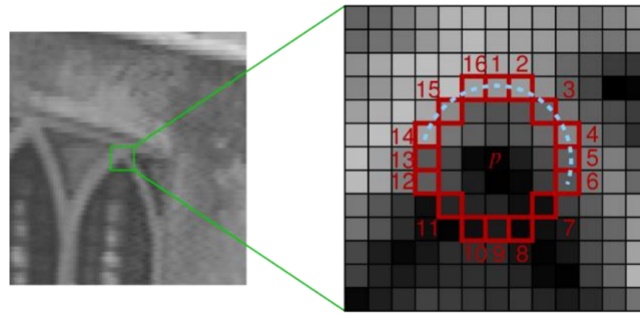

FIGURE 3.1 – Détection de coins par FAST

Le détecteur FAST est utilisé dans la détection de coins. Nous considèrons p comme étant le pixel central, ainsi si un ensemble de n voisins consécutifs sont considérés comme 'clairs' ou 'sombres' , alors on dit que p est un coin.

A cette couche de détection est ajouté un apprentissage qui permet de contrôler la comparaison des pixels et d'en réduire le nombre et ainsi diminuer le temps de calcul du détecteur [4].

3.1.2.2 Détecteur BRIEF

Le détecteur BRIEF est basé sur la recherche du proche voisin, avec une taille de voisinage et une langueur de descripteur indépendante. Dans cette méthode, nous sélectionnons de façon aléatoire des paires de pixels dans un large voisinage et on calcule le descripteur en fusionnant les résultats des différentes comparaisons binaires de chacune des paires. Cet algorithme est conçu pour être invariant en échelle et en orientation. Il est également robuste au changement d'éclairage et au flou [5].]. Le détecteur BRIEF est également connu pour ne pas être trop gourmand en terme de ressources mémoire et de vitesse de calcul. Ce qui rend cet algorithme capable d'être intégré dans un système embarqué.

Il existe de nombreuses versions de ce détecteur, mais son fonctionnement général est comme décrit ci-dessous :

Nous séléctionnons n_d pairs de pixels aléatoires et nous comparons leur intensité avec la méthode suivante : si nous considèrons p et q deux pixels choisis de façon aléatoire, le résultat de la comparaison de l'insenté des pixels est 1 si $I(p) < I(q)$ et 0 sinon, avec I représentant la fonction d'intensité du pixel. Le résultat de ce descripteur est la combinaison des n_d comparaisons.

Ce détecteur ne renvoie pas de points d'intérêt, il doit donc être combiné avec un autre détecteur qui va détecter les points d'intérêt. Son rôle est de réaliser une correspondance plus rapide des points d'intérêt, sauf dans le cas d'une forte invariance par rotation dans le plan.

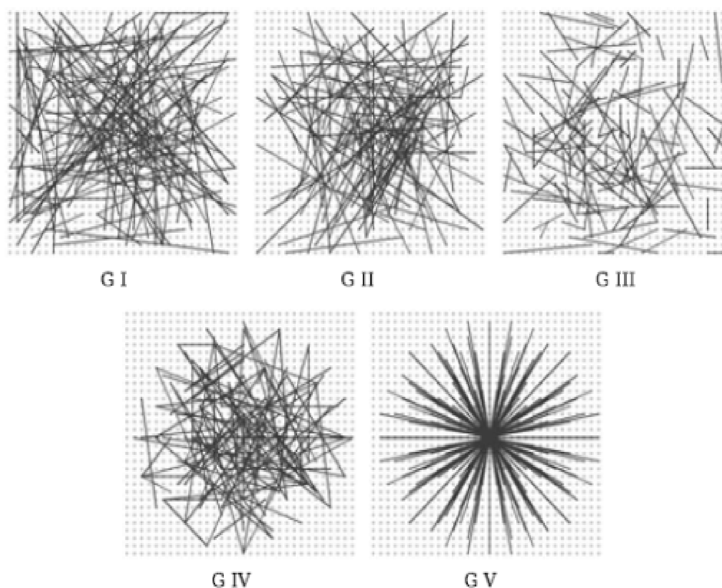

FIGURE 3.2 – Divers résultats du détecteur BRIEF selon la variation par rotation

3.1.2.3 Detecteur ORB (Oriented FAST and Rotated BRIEF)

L'algorithme ORB (Oriented FAST and Rotated BRIEF) a été développé par le laboratoire d'OpenCV et a l'avantage d'être Open Source. Il est donné comme alternative au détecteur SIFT qui est plus gourmand en temps de calcul et en ressources, en plus d'être sensible aux bruits dans l'image. Son but principal est de permettre de faire des correspondances d'images avec de faibles ressources comme c'est le cas des systèmes embarqués et de réduire le temps de détection dans les ordinateurs ordinaires. L'ORB est basé sur la combinaison de deux détecteurs, le détecteur FAST et le détecteur BRIEF mais en y ajoutant quelques modifications. Les principales modifications ajoutées sont les suivantes[6] :

- L'ajout d'un composant d'orientation rapide et précis à FAST
- Une méthode de calcul efficace de la détéction par BRIEF orienté
- Analyse de la variance et de la corrélation de la détection par BRIEF orientées
- Une méthode d'apprentissage pour décorréler les détections par BRIEF orientées même en cas d'invariance par rotation

L'ORB utilise un oFAST, c'est à dire un détecteur FAST orienté. Cette composante d'orientation dans l'ORB est ajoutée grâce à un centroïde d'intensité qui suppose que dans la détection de coins, l'intensité de chaque coin est différente que celle du centre. Il utilise donc le vecteur entre le centre et le coin comme orientation. Il découpe ainsi l'image en parcelle et définit le moment

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \quad (3.1)$$

Avec ces moment nous trouvons le centroïde

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \quad (3.2)$$

Et le vecteur $\overrightarrow{OC}$ du centre du coin vers le centroide qui représente l'orientation :

$$\Theta = atan2(m_{01,}m_{10})(3.3)$$

L'ORB combine avec le oFast une variante du détecteur BRIEF appelé rBRIEF (Rotation-Aware Brief). Dans rBRIEF, nous ajoutons une orientation qui a pour but d'obtenir une invariance par rotation sur le plan puisque les performences du BRIEF chute rapidement pour une rotation dans le plan. Nous dirigeons ainsi le détécteur BRIEF en fonction des coins à détécter.

Le rôle de la méthode d'apprentissage ajoutée est de remédier à une perte de variance du détecteur rBRIEF.

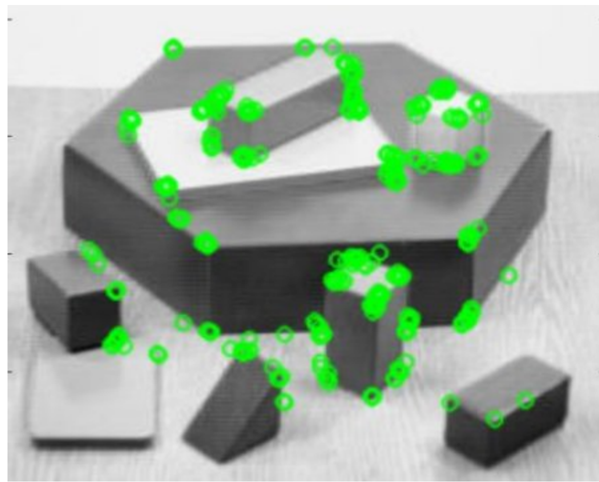

FIGURE 3.3 – Résultat d'une détection avec ORB

3.2 ORB-SLAM

3.2.1 Présentation de l'ORB-Slam

Comme nous l'avons cité précédemment, l'ORB-Slam combine l'approche métrique et topologique, ainsi dans son fonctionnement on va trouver de la détection par rapport à des amers et de la reconstitution d'endroits. Le fonctionnement de l'ORB-Slam repose sur trois tâches principales : la tâche est le suivi, la tâche la cartographie locale et la tâche de la fermeture de boucle. Toutes ces tâches reposent sur un choix strict d'images et d'amers. L'ORB-Slam intègre ainsi des mécanismes pour détecter les images dupliquées pout les supprimer et détecter les mauvais appariements des amers.

La structure des amers dans le système ORB SLAM est composée de [12] :

— $X_{w,i}$ la position 3D des amers dans le repère du monde
— L'angle de vue, il s'agit d'un vecteur moyen de tous les angles de vues.
— Un descripteur représentative type ORB noté D_i , ce descripteur est choisi de telle sorte que la distance Hamming soit minimale.

— dmax , dmin sont respectivement la distance maximale et minimale avec lesquelles l'amer peut être observé.

Celle d'une image notée K_i est composée de :

— La position de la caméra notée T_{iw} , (transformation du repère du monde vers le repère de la caméra).

— Les paramètres intrinsèques de la caméra.

— Tous les amers avec leurs descripteurs ORB détectés dans cette image.

3.2.2 Présentation des diverses tâches

3.2.2.1 La tâche de suivi

Cette tâche s'occupe de la localisation de la caméra lors de la réception de chaque image en l'ajoutant ou non au système. Pour cela elle utilise un modèle d'évolution pour connaître la position de la caméra. Cette tâche s'occupe également de la relocalisation en cas de perte la position si le robot tourne trop rapidement. Et enfin elle sauvegarde l'image ou l'abandonne selon sa pertinence. La tâche de suivi est décomposée en sous-tâches qui sont :

— L'extraction par détecteur ORB

Dans cette étape, on utilise le détecteur ORB cité dans la partie (3.1.2.3) pour repérer les points d'intérêt c'est à dire les différents coins présents dans l'image. Pour chaque amers détecté, nous calculons l'orientation qui va nous servir dans la phase de l'appariement.

— Estimation de la position de la caméra et relocalisation

Cette étape repose sur l'étape précédemment décrite. Ainsi si on détecte assez de coins c'est à dire nos amers et que le nombre dépasse un seuil fixé, l'ORB-Slam estime la position de la caméra en utilise un modèle de mouvement à vitesse constante. Au cas où le nombre d'amers n'est pas suffisant, un processus de relocalisation est lancé. Ce processus tente alors une relocalisation globale c'est à dire une recherche plus large des points pour essayer de trouver les points de la dernière image enregistrée.

— Suivi de la carte locale

Cette étape utilise un graphe de covisibilté qui est un graphe qui connecte toutes les images sélectionnées par l'ORB-Slam. Chaque nœud de ce graphe est une image et le lien entre ces nœuds est le nombre n'amers partagés et qui doit dépasser 15 amers. Grâce à ce graphe, nous distinguons les amers qui constituent la Map locale et à chaque nouvelle image reçue à partir du flux vidéo on réalise une projection avec les amers de l'image précédente. Entre ces deux images, nous trouvons plusieurs amers en commun.

— Décision pour ajouter ou pas une image

Le but de cette étape est de rendre l'algorithme plus robuste vis à vis des rotations. Pour que l'image soit rajoutée dans le système et soit considérée comme image clé elle doit remplir les conditions suivantes :

- Plus de 20 images doivent être prises après une relocalisation globale

- Le mappage local doit être en mode inactif ou bien plus de 20 images sont passées depuis l' insertion la dernière image clé.

- Cette image doit contenir au moins 50 points détéctés.

- Cette image suit moins de 90% de points que l'image clé précédente, ce qui permet d'éviter la redondance.

3.2.2.2 Cartographie locale

C'est dans cette étape que notre Map est créée à partir des différents amers qui représentent les différents points de la Map. Ainsi cette étape traite les nouvelles images acquises et exécute une compensation par faisceaux local pour obtenir une Map optimale. Cette Map doit également être tenue à jour grâce à une Map locale qu'on obtient instantanément lors de l'analyse de chaque image. Pour cela on utilise les processus suivants :

— Insertion d'une nouvelle image

L'insertion d'une nouvelle image permet de tenir le graphe de covisibilité à jour et cela en lui ajoutant de nouvelles images qui représentent ses nœuds et en reliant celle-ci avec les anciennes images qui partagent un grand nombre d'amers en commun.

— Sélection des amers

Pour obtenir une carte fiable, les amers doivent satisfaire un certain nombre de conditions. Ils doivent être observés et suivis dans au moins 25 images et ils doivent également apparaitre dans au moins 3 images consécutives. Ainsi si ces conditions sont satisfaites, l'amers est ajouté dans la carte.

— Création de la map à partir d'amers

Pour créer un nouvel point de la carte c'est à dire un amer, nous avons besoin de deux images. Cela n'empêche pas que ce point apparaisse dans d'autres images. Celles-ci doivent être présentes dans le graphe de covisibilité, ensuite grâce à une triangulation on crée le lien entre elles. Pour qu'un amer soit accepté, nous vérifions sa profondeur qui doit être positive.

— Compensation par faisceaux

Dans cette étape, nous optimisons les images traitées (la dernière image prise) et toutes les images qui lui sont associées dans le graphe de covisibilité grâce à une compensation par faisceaux. Dans ce processus d'optimisation, nous représentons toutes les images qui contiennent les mêmes points que l'image traitée et avec lesquelles celle-ci n'est pas liée comme fixe, ce qui nous permet de les tenir également en compte. Après optimisation, le résultat permet d'ignorer les observations aberrantes.

— Suppression des images

Dans cette étape, on supprime les images sur lesquelles on a une forte redondance des images. Ces images sont celles dont 90% des amers se trouvent déjà dans les trois dernières images observées.

3.2.2.3 Fermeture de boucle

Cette étape cherche une fermeture de boucle à chaque nouvelle image prise. A chaque boucle détectée, nous calculons le glissement cumulé et fusionnons les amers dupliqués et ensuite nous

optimisons le graphe de covisibilité. Pour trouver cette fermeture en boucle, nous procèdons ainsi :

— Détection de fermeture de boucle

Ici nous fixons un seuil de ressemblance minimum et nous comparons la similitude de deux images par rapport à ce seuil. Dans ce processus, toute les images liées à notre image sont directement éliminées. Pour valider cette fermeture de boucle nous devons détecter au moins trois fermetures de boucle d'images successives liées en elles. Cela sert à éviter de déclarer deux images qui se ressemblent directement comme fermeture puisqu'on peut facilement avoir deux scènes presque pareilles.

— Calcul de la transformation de similarité

Cette étape intervient pour calculer la dérive accumulée lorsqu'on repasse par le même endroit. Cette dérive est plus remarquée dans le SLAM monoculaire. Ainsi, pour réaliser cette fermeture de boucle, nous devons réaliser une transformation entre la dernière image et l'image avec laquelle nous supposons qu'il y'a fermeture de boucle.

— Fusion de boucle

C'est dans cette étape là que nous réalisons une fusion des amers dupliqués et nous mettons à jour le graphe de covisibilité en y rajoutant les nouvelles transitions. Cette étape est réalisée après correction de la position des amers de l'image avec le facteur de la transformation de similarité et une projection des amers sur l'image avec laquelle nous avons potentiellement détecté une fermeture de boucle.

Le diagramme ci-dessous récapitule toutes les étapes de l'ORB-Slam avec leur enchaînement.

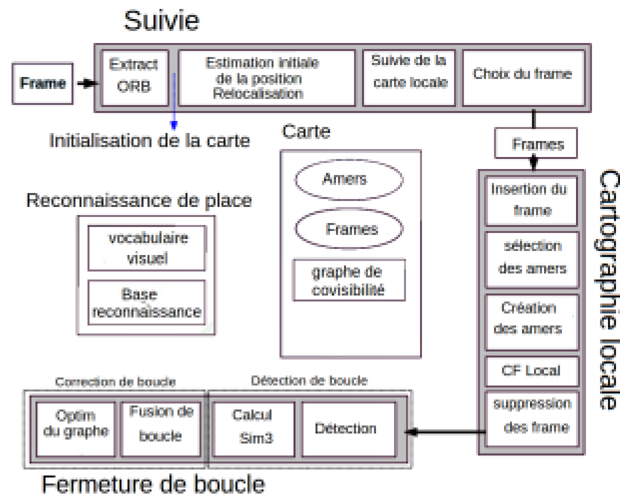

Figure 3.4 – Tâches réalisées par l'ORB-SLAM

Chapitre 4

Implémentation

Après avoir expliqué le fonctionnement de l'ORB-Slam, nous allons voir les outils que nous avons utilisé dans le cadre de notre projet pour l'implémenter puis tester ces performances dans différents systèmes. Nous étions ammené à travailler en équipe donc nous avons décidé d'utiliser l'outil Git qui va être expliquer par la suite. Pour gerer les différents capteurs intégrés dans le robot et pour le commander nous avons décidé d'utiliser le Middleware ROS qui propose divers outils pour faciliter cela. Ensuite nous avons implémenté notre algorithme de SLAM dans un système AMD et ARM pour tester ces performances.

4.1 Outils

4.1.1 GIT

Git est un outil de gestion des différentes versions d'un projet. Il a été conçu principalement pour être utilisé dans les projets informatiques pour gérer les code sources mais peut également être utilisé pour gérer les documents. Il est utilisé pour des divers avantages qui sont

- Permet de travailler en local tout en ayant accès au depôt central
- Permet un travail collaboratif sur le même fichier
- Fusionne le travail des différentes personnes qui ont travailler sur le même fichier sans rien écraser
- Conserve l'historique des modifications des fichiers avec le nom de celui qui les a modifier et la date de modification
- Permet de réaliser un retour en arrire
- Permet de switcher rapidement d'une version ‘a une autre et les comparer

Nous avons donc utilisé cette outil pour gerer les différentes versions de notre code que nous avons déposé dans une répertoire dans Gilab.

4.1.2 ROS

4.1.2.1 Definition

Figure 4.1 – Logo de ROS

Robot Operating System (ROS) est ensemble de logiciels en libre-service et qui, comme son nom l'indique, est un système d'exploitation pour les robots. Nous pouvons également le représenter comme un Framework pour écrire des logiciels robotiques. Son but est de créer une standardisation de la programmation dans la robotique.

ROS offre un ensemble de programmes qui permettent l'utilisation de divers capteurs, des logiciels de visualisation, des connexions inter-machines ainsi que des logiciels de simulation. Il peut être utilisé avec plusieurs langages de programmation tels que Python et C++.

4.1.2.2 Fonctionnement

Le fonctionnement de ROS est similaire à celui d'un client-serveur. Le master représente le serveur et les différents nœuds le client. Un nœud en ROS est un exécutable qui peut être, par exemple, les données d'un capteur. Le master, quant à lui, est le serveur où doivent s'abonner tous les nœuds avant de pouvoir discuter entre eux. Une fois abonnés au master, les nœuds discutent avec des topics qui sont des services de transports de l'information. Les nœuds peuvent soit publier des informations dans ces topics soit lire les informations publiées dans ces derniers ou bien les publier et les lire en même temps.

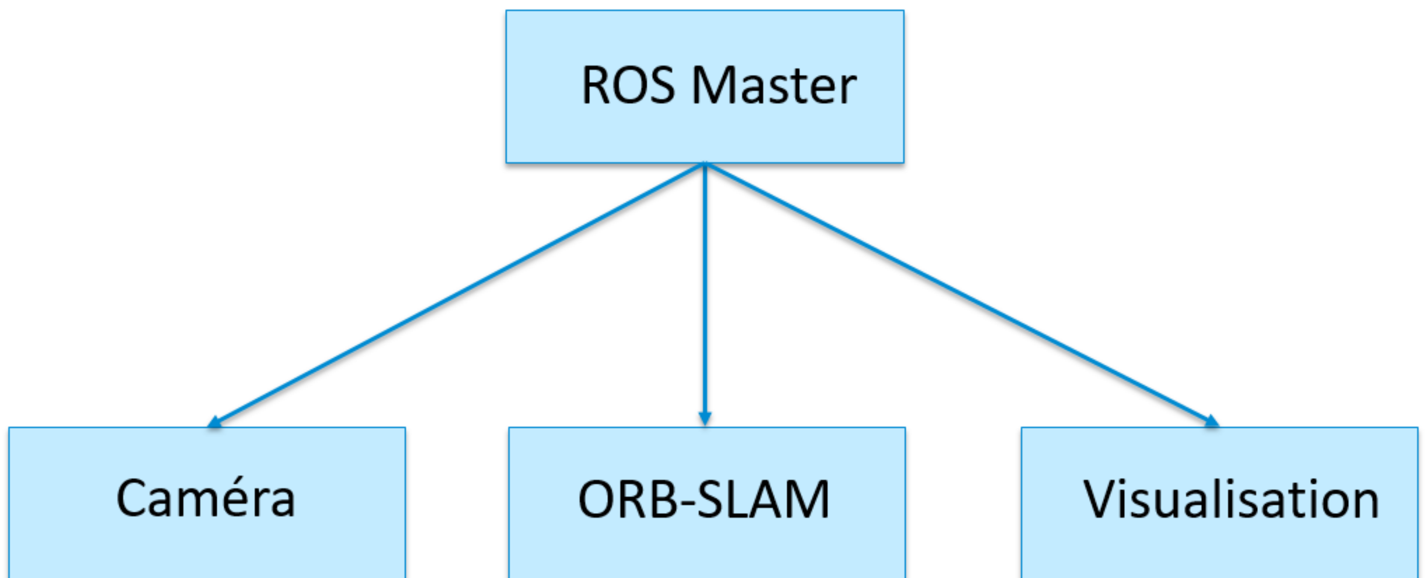

Figure 4.2 – Schèma de communcation avec de le Master

Dans le cas de l'ORB-SLAM, nous avons trois nœuds : celui de la caméra, celui de l'ORB-SLAM et celui de la visualisation. Les trois nœuds demandent au master l'autorisation pour pouvoir communiquer leurs informations. Une fois cette autorisation donnée, les nœuds peuvent communiquer entre eux grâce à des topics.

Figure 4.3 – Communication des noeuds par topic

Comme représenté dans la figure 5.1, le nœud usb_cam lit les données de la caméra et

envoie le flux vidéo au nœud ORB_Slam grâce au topic image_raw. ORB-Slam réalise ainsi son traitement sur ce flux vidéo en y détectant les différents coins. Il renvoie par la suite le flux traité avec le topic Frame où s'abonne le nœud image_raw pour l'afficher.

4.1.2.3 Simulation

ROS propose un outil de simulation nommé Gazebo. Ce dernier permet de simuler une multitude de robots et d'environnements. Son avantage est le fait qu'on peut y tester nos algorithmes rapidement pour vérifier leur viabilité avant de les implémenter dans de vrais robots.

Nous avons donc réalisé une simulation de notre robot Laika dans un environnement intérieur. Nous y avons intégré les capteurs nécessaires, c'est à dire une caméra, un accéléromètre, un gyroscope et un encodeur pour l'odométrie. Nous avons ensuite décidé de tester l'algorithme de l'ORB-Slam dessus.

Ensuite, nous avons paramétré l'ORB-Slam pour fonctionner avec cette simulation et l'avons testé. Les résultats étaient satisfaisants quant à la vitesse d'exécution de l'algorithme et le nombre d'amers détectés. Nous sommes parvenus à construire une Map de la pièce même si celle-ci présente quelques défauts. L'origine de ces défauts est l'estimation de la profondeur des amers qui étaient mal calculés par le modèle de la caméra. Dans cette simulation l'ORB-Slam n'utilise que les données de la caméra. Une amélioration à ajouter plus tard est de combiner les données de la caméra avec ceux de l'odométrie pour avoir une Map plus appréciée.

Les photos ci-dessous représentent l'ORB-Slam lors de son processus de Matching des points et lors de la création de la Map.

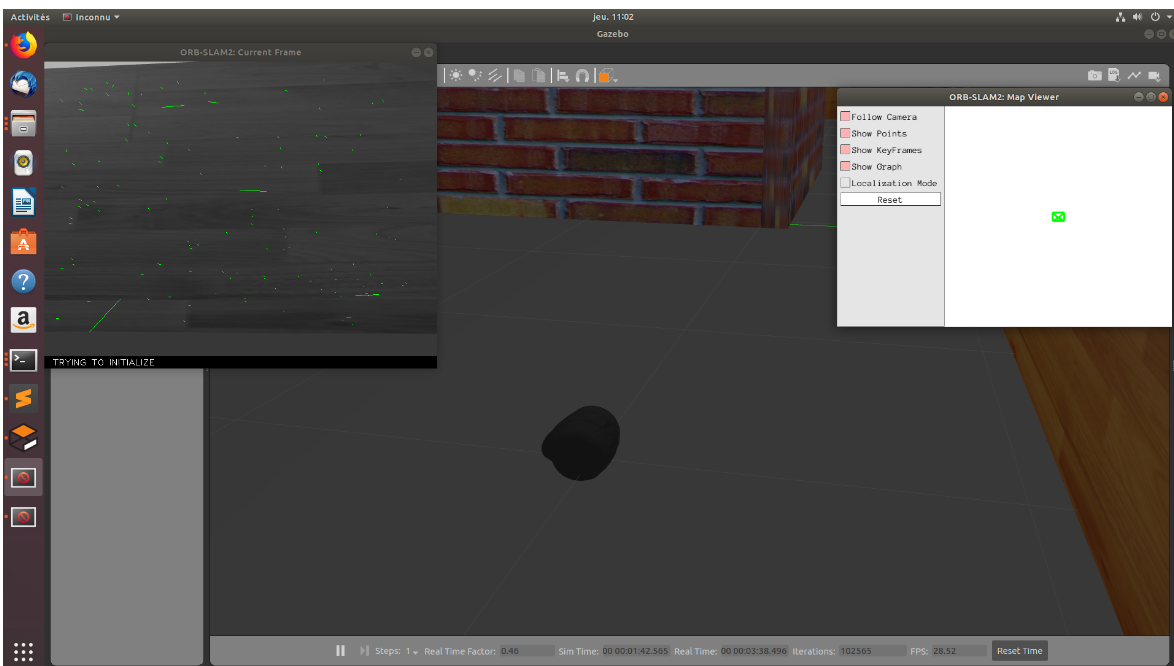

FIGURE 4.4 – Phase de matching les points avec la simulation du robot Laika

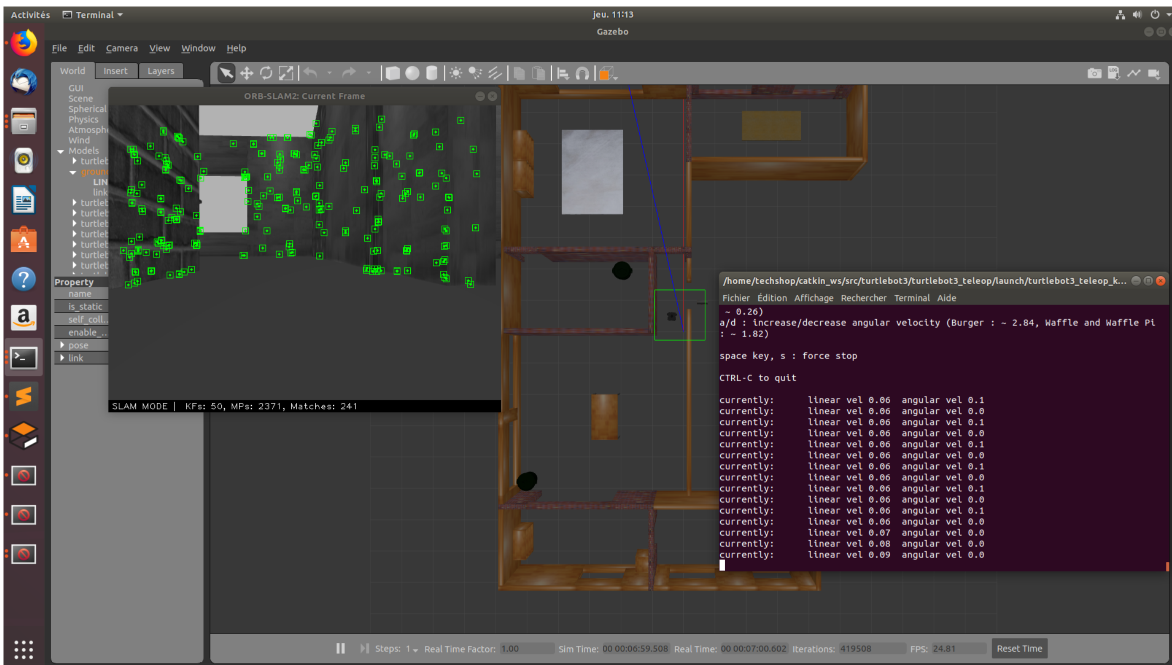

FIGURE 4.5 – Détéction d'amers avec la simulation du robot Laika

4.2 Performances

Les derniers tests étant réalisés avec un serveur d'une très grande puissance de calcul, nous avons décidé de tester nos algorithmes sur des machines moins puissantes afin d'évaluer les performances de notre système. Nous avons commencé par le tester sur une machine AMD avec un processeur Pentium de 4 cœurs. Ensuite nous l'avons implémenté dans un système ARM sur une carte Raspberry et une carte DART SD-410.

4.2.1 Test sur système AMD

Pour évaluer l'algorithme d'ORB-Slam, nous avons commencé par l'implémenter dans un système de type AMD, qui utilise Ubuntu 16.04 comme système d'exploitation et la version ROS Lunar. Nous l'avons implémenté dans cette architecture plus puissante qu'une carte électronique afin de tester sa réactivité et son temps d'exécution. Ainsi, si l'ORB-Slam ne fonctionne pas en temps réel sur notre ordinateur, il ne le sera pas sur la carte également.

Cet algorithme a été initialement développé sur une version Ubuntu 12.04 avec un ROS Fuerte et a été testé sur un Ubuntu 14.04 avec un ROS Indigo. Nous n'avions donc ni la bonne version Ubuntu, ni la bonne version ROS. Ainsi, mon premier travail sur cet algorithme a été un travail sur les dépendances et le débug de celui-ci, car il utilisait des bibliothèques présentes sur les anciennes versions Ubuntu mais qui n'existent plus sur les nouvelles. Ceci a nécessité dans certains cas de compiler les codes source de ces programmes.

Après cette étape de débugage, nous avons testé l'ORB-Slam avec des flux vidéo de scènes

que nous avons téléchargé. Les résultats de ces tests étaient satisfaisants : nous avons obtenu une Map claire de l'environnement. Nous l'avons testé par la suite avec un flux vidéo de la caméra. Pour cela, nous avons écrit des nœuds ROS de lecture de la caméra et de publication de son flux où doit s'abonner l'ORB-Slam. En plus du flux vidéo, l'ORB-Slam prend également les données de calibration de la caméra. Nous avons donc écrit un programme qui permet de les obtenir.

Après paramétrage de ce SLAM pour fonctionner avec la caméra et avec nos données de calibration, nous pu le tester avec celle-ci. Ainsi, nous sommes parvenus à obtenir le SLAM Visuel avec une bonne précision de la carte en temps réel.

Il existe deux versions de l'ORB-SLAM. Nous avons commencé par effectuer les premiers tests avec la première version de l'ORB-SLAM et nous sommes passés par la suite à la version ORB-SLAM. La différence entre ces deux versions est que la version 2 propose un mode de localisation. Dans ce mode, le thread s'occupant de la cartographie et celui s'occupant de la fermeture de boucle sont désactivés et nous n'avons plus que le thread de localisation qui fonctionne. Lorsque cet algorithme sera implémenté dans le robot, nous aurons besoin de se mode là.

Pour la suite du projet, elle a été réalisée à l'aide de la version 2 de l'ORB-SLAM.

En pratique, l'algorithme fonctionne ainsi :

1. Il faut bouger la caméra de façon à réaliser le Matching de points comme décrit dans la partie ORB-SLAM afin d'obtenir des amers sûrs. Le temps de cette tâche varie selon le nombre de points détectés et de leur continuité dans les images successives.

Figure 4.6 – Phase de matching de points dans l'algorithme d'ORB-Slam

2. Il suffit ensuite de bouger la caméra en suivant une trajectoire définit. Lors de ce mouvement, nous obtenons une mise à jour continue de nos amers ainsi que notre Map locale. La Map globale est la fusion de la Map locale et des points détectés précédemment.

FIGURE 4.7 – Amers obtenus grâce au détecteur ORB dans l'ORB-SLAM

Ainsi nous obtenons une carte de notre environnement et la trajectoire de la caméra

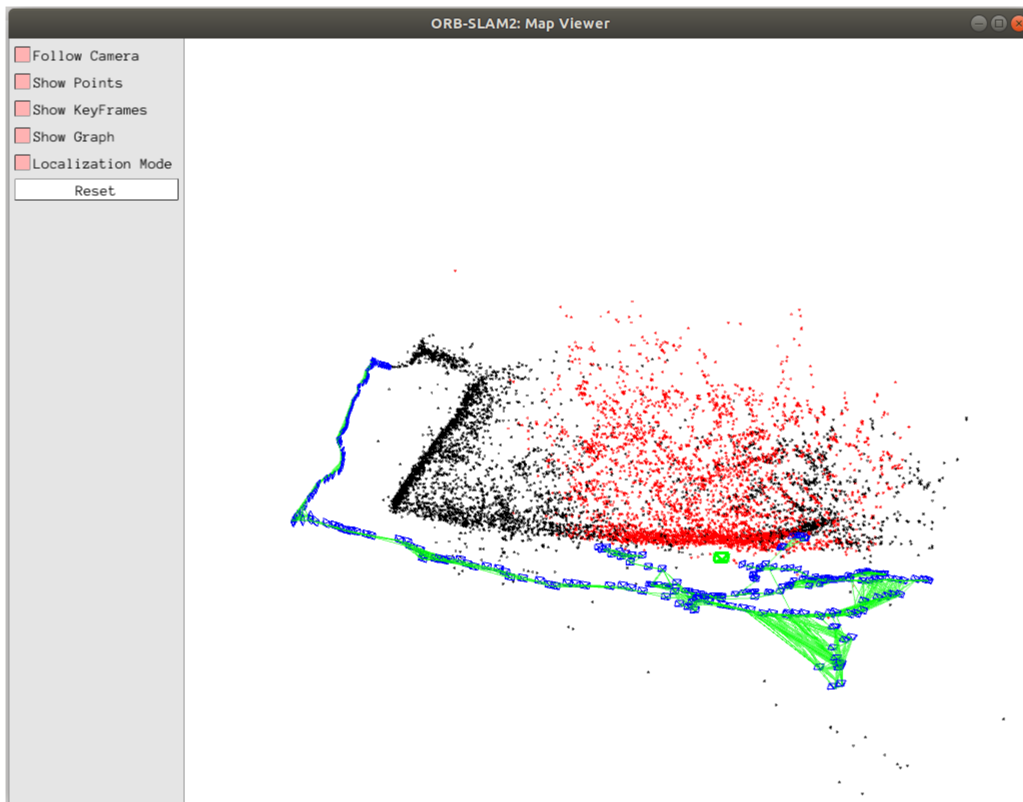

FIGURE 4.8 – Résultat de l'algorithme d'ORB-SLAM2

Dans l'image ci-dessus, les carrés bleus représentent les anciennes positions de la caméra, le carré vert la position actuelle de la caméra, les points rouges les points de la Map locale, les

points noirs les points enregistrés et les traits verts la trajectoire de la caméra.

Après avoir testé cet algorithme, il a fallu lui ajouter les fonctionnalités dont nous aurons besoin et qui n'y sont pas implémentées. La première fonctionnalité que nous avons développé est l'ajout d'un composant qui permet d'enregistrer la Map et de la recharger par la suite car après l'arrêt du système actuel, toutes les données de la Map sont perdues. Cette modification a donc permis d'enregistrer la Map, de la recharger et de se localiser à l'intérieur de celle-ci.

La Map que renvoie l'ORB-SLAM est une Map 3D. Le but de cette dernière est d'être utilisée avec un nœud de navigation déjà implémenté sur ROS appelé Navigation Stack. Ce nœud prend en entrée une Map 2D avec un format particulier de type *.bpm*. Le travail suivant est donc de convertir notre Map 3D en Map 2D au format *.bpm*.

4.2.2 Test sur système ARM

Après implémentation dans un système de type AMD et développement des fonctionnalités manquantes, il a fallu tester cet algorithme sur un système embarqué.

Le premier problème qui s'est posé pendant cette implémentation est le problème de dépendance. Notre carte Raspberry était au début utilisée avec un système Rasbian et nous l'avons changé par la suite par un système d'exploitation Ubuntu Mate. Ce dernier présente moins de problèmes de dépendance des bibliothèques. La carte qui va être utilisée par la suite dans notre robot est une carte DART SD-410. Elle a comme système d'exploitation un Debian Stretch.

Après résolution de tous les problèmes de dépendance, le deuxième problème rencontré était celui de la compilation. Les ressources mémoire des deux cartes étaient faibles et ne permettaient pas de compiler l'ORB-SLAM. Sur la Raspberry, en effectuant une bonne gestion de la mémoire, nous sommes parvenus à le compiler. En ce qui concerne la DART, nous avons dû acheter une carte avec plus de ressources mémoire.

Après compilation de ce SLAM sur les deux cartes, nous sommes passés à la phase de test. Par rapport aux tests réalisés sur la Raspberry, nous avons eu une certaine latence lors du traitement des images de la caméra après exécution de l'algorithme, en particulier lors de la détection des amers par l'algorithme ORB. Malgré cette latence, nous avons pu obtenir une carte. En revanche, son inconvénient est que le robot devrait bouger à faible vitesse, ce qui augmentera le temps de création de la Map.

Et en ce qui concerne l'implémentation sur la DART, puisque celle-ci n'a pas d'interface graphique, nous avons dû afficher la Map ainsi que l'image de la caméra dans un ordinateur connecté sur le même réseau local. Lors du test de l'algorithme avec un flux vidéo téléchargé, la phase de Maching et la création de la carte se faisaient de façon normale avec une certaine latence. Mais en effectuant le test avec notre propre caméra, l'algorithme prenait plus de temps dans la phase de Matching des points. Ce temps est dû soit à une mauvaise manière de bouger la caméra de notre part, soit à l'algorithme qui prend plus de temps à s'exécuter sur cette carte.

Conclusion

Ce stage fut une agréable découverte du travail au sein d'une Stat-up car il diffère largement des grandes entreprises en termes d'ambiance de travail, de relationnel et de charge de travail. A travers ces multiples facettes, ce stage fut une expérience extrêmement enrichissante. En effet, j'ai participé au quotidien de Camtoy en essayant de résoudre les problèmes rencontrés dans le cadre de mon projet. J'ai ainsi pu mettre en pratique les diverses connaissances théoriques que j'ai acquises lors de mon cursus à l'ENSTA Bretagne. J'ai également eu l'occasion de développer de nouvelles compétences et de découvrir de nouveaux outils de travail.

En ce qui concerne le projet, nous avons pu réaliser une grande partie de l'implémentation sur la carte embarquée qui sera intégré dans le robot. Cet algorithme a pu être testé sur un simulateur et il ne reste plus qu'à le tester sur le vrai robot. Cette tâche est impossible pour le moment caque nous ne possédons pas de robot opérationnel.

La tâche qu'il reste à faire mis-à-part l'implémentation sur le robot, est de trouver la source de la lenteur de l'ORB-SLAM sur la DART. Si ce problème n'est pas résolu, l'une des solutions possibles est de réaliser les calculs de la première carte à l'allumage du robot dans un serveur qui peut communiquer avec le robot, et qui dispose de plus de ressources mémoire.

Bibliographie

[1] Antoine Manzanera, « Cours :Traitement d'Images et Vision par Ordinateur », ENSTA-ParisTech / U2ISf

[2] Manuel Grand-brochie, Christophe Tilmant, Michel Dhome « Descripteur local d'image invariant aux transformations affines », LAboratoire des Sciences et Matériaux pour l'Electronique et d'Automatique (LASMEA)

[3] Hélène THOMAS, « Cours : Vision par ordinateur Détection de points d'intérêt, description et appariement », ENSTA Bretagne, dec 2017

[4] Youssef El Rhabi, « Alignement de données 2D, 3D et applications en réalité augmentée. »,Université de Caen Normandie, juin 2017

[5] Michael Calonder, Vincent Lepetit, Mustafa Özuysal, Tomasz Trzcinski, Christoph Strecha,Pascal Fua « BRIEF : Computing a local binary descriptor very fast », Ecole Polytechnique Fédérale de Lausanne (EPFL) Computer Vision Laboratory

[6] Ethan Rublee,Vincent Rabaud, Kurt Konolige, Gary Bradski « ORB : an efficient alternative to SIFT or SURF »,Laboratoire OpenCV

[7] Marion Decrouez, « Localisation et cartographie visuelles simultanées en milieu intérieur et en temps réel », Laboratoire Informatique de Grenoble à l'INRIA Grenoble Rhône-Alpes, août 2006

[8] Nicolò Valigi, « Open source Visual SLAM evaluation », Cruise Automation, sep 2016

[9] Aurélien Gonzalez, Michel Devy, Joan Solà, « SLAM visuel monoculaire par caméra infrarouge »,Université de Toulouse, juil 2011

[10] Émilie Wirbel , « Localisation et navigation d'un robot humanoïde en environnement domestique », l'École Nationale Supérieure des Mines de Paris, oct 2014

[11] Brian Williams, Georg Klein,Ian Reid, « Real-Time SLAM Relocalisation », University of Oxford

[12] Raul Mur-Artal, J. M. M. Montiel, Juan D. Tardos, « ORB-SLAM : a Versatile and Accurate Monocular SLAM System », IEEE Transactions on Roborics, 2015

[13] F. Servant, « Localisation et cartographie simultanées en vision monoculaire et en temps réel basé sur les structures planes », Université de Rennes 1, juin 2009

[14] Frédéric Devernay,« Détection de points d'intérêts »,UFRIMA