



# Représentation et positionnement relatif à partir d'informations visuelles

Rapport de stage M1

Lucie SCHWARZ

# Table des matières

# 1 Cadre du projet

## 1.1 Objectifs

Les robots ont des difficultés à se reperer juste a partir d'une caméra, ils sont souvent aidés d'un laser pour mesurer les distances. Celà vient en partie du fait que le traitement d'image nécessaire est coûteux en ressources et pas toujours fiable.

L'objectif ici est de permettre à un raspberry doté uniquement d'une caméra (ou deux dans le cas d'une vision stéréoscopique) de modéliser grossièrement son environnement au sein d'un moteur java 3D (JME3). Les outils utilisés seront un raspberry doté d'une camera. On s'aidera des algorithmes déjà implémentés sur OpenCv ainsi qu'OpenSLAM, qui sont tous deux des bibliothèques spécialisées dans le traitement d'images.

L'hypothèse est que la représentation d'une entité dans l'espace ne se restreint pas à la capture d'une information externe, mais que le système projète également un modèle de ce qu'il s'attend à percevoir comme grille de lecture. Cela afin d'accélérer le traitement et ne plus analyser "bêtement" chaque pixel de l'image captée comme en SLAM.

Ainsi, si l'entité s'attend à entrer dans une pièce, elle peut construire, à priori, un parallélépipède rectangle dans sa représentation virtuelle, puis tenter d'ajuster les dimension de ce parallélépipède de façon à coller avec ce qui est perçu. Si l'on se restreint aux informations visuelles, il ne serait alors plus nécessaire de réaliser de coûteux traitements d'image pour identifier les arêtes et murs de l'image issue de la caméra. Les informations attendues a priori permettent d'accélérer le traitement. Bien sûr, le problème devient plus complexe lorsque l'on s'intéresse à des éléments plus petits ou plus riches.

Dans le cadre de ce stage, et comme évoqué dans l'exemple précédent, l'objectif serait de s'intéresser dans un premier temps à la reconnaissance de la structure générale des environnements fermés. En statique puis en dynamique, en comparant les performances avec des algorithmes de SLAM ou de reconnaissance de la littérature, via openCv ou openSLAM.

Un certain nombre de travaux commencent aujourd'hui à s'intéresser à cette approche, mais ils ne l'utilisent qu'en post-traitement. On fait l'hypothèse que structurer la représentation du monde par une projection simultanée des connaissances et des capteurs sur un même espace permettrait d'accroître significativement la performance des algorithmes de reconnaissance, et les capacités de représentation et d'interaction d'un agent autonome.

Le but concret de mon projet est de proposer un système de repérage d'un raspberry dans un environnement clos juste à partir d'une caméra.

## 1.2 Hypothèses simplificatrices

Le sujet étant très étendu, j'ai décidé de poser des hypothèses simplificatrices afin de me limiter à certaines configurations particulières. Les hypothèses sont les suivantes :
- le raspberry se trouve dans une pièce close, un parallèlépipède, dont il faudra déterminer les dimensions
- sur les photos prises par le raspberry on voit le sol ainsi que le plafond
- l'image est prise face à une face de la pièce de manière droite
- le raspberry est posé sur une surface plane
- la face du parallèlépipède face au raspberry est visible entièrement sur l'image

La pièce dans laquelle on se trouve sera donc visible en perspective frontale.

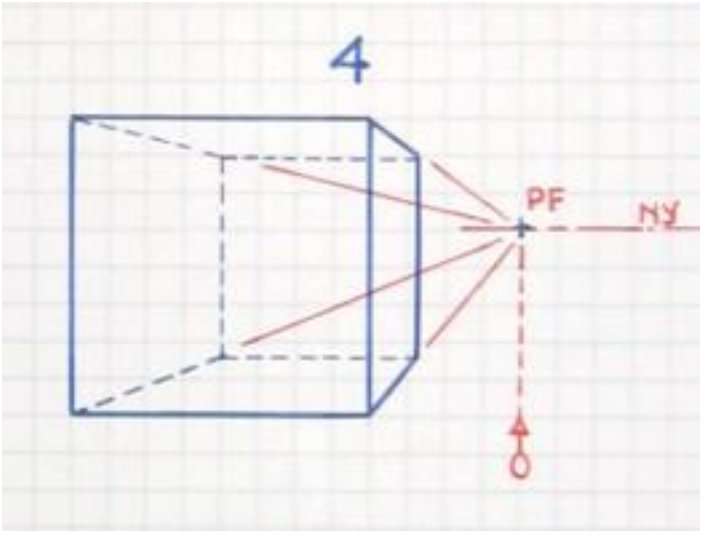


Figure 1 – Parallèlépipède observé en perspective frontale

### 1.3 Méthodologie

Afin de réaliser notre objectif, je vais présenter une méthode pour identifier le mur sur la photo et un rectangle le contenant. A partir de cette donnée, il sera possible de déterminer l'angle apparent du mur. Enfin, à partir de deux clichés successifs, on se servira des angles apparents et de la distance parcourue pour en déduire des dimensions réelles.

## 2 Identification du mur sur le cliché

A partir d'une image brute, il faut effectuer plusieurs transformations sur l'image afin de pouvoir identifier le mur frontal. Je vais vous présenter dans la partie qui suit les différentes étapes que j'ai effectuées afin d'y parvenir.

### 2.1 Segmentation d'images

Dans un premier temps j'ai cherché à partager l'image en plusieurs parties pour en retirer des informations. La segmentation d'images est la division d'une image en zones homogènes afin de séparer les divers composants visibles et de les identifier. Il existe trois principales approches :
- la segmentation fondée sur les régions
- la segmentation fondée sur les contours
- La segmentation fondée sur la classification ou le seuillage des pixels en fonction de leur intensité

Je me suis concentrée sur l'approche par détection de frontières qui exploite le fait qu'il existe une transition détectable entre deux régions connexes

### 2.2 Détection des contours

La première étape que j'ai réalisée pour la segmentation de l'images était la détection de contours. J'ai pour cela testé deux méthodes différentes : l'application du filtre de Canny sur l'image et la transformée de Laplace.
On peut constater que les contours sont plus nets avec le filtre de Canny qu'avec la transformée de Laplace. J'ai donc choisi pour la suite d'utiliser le filtre de Canny.

Figure 2 – Image d'origine

Figure 3 – Application Canny

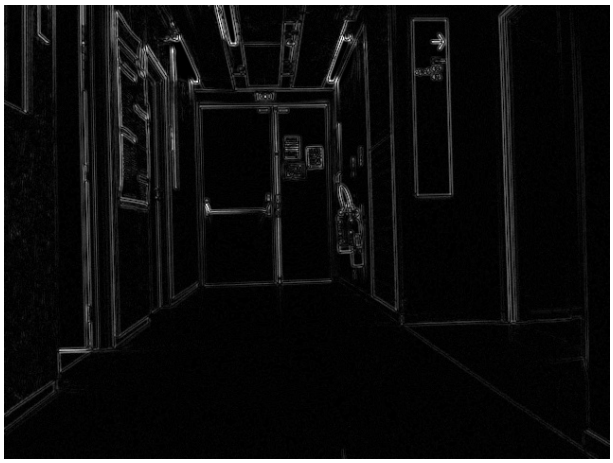

Figure 4 – Application Laplace

## 2.3 Détection de lignes

A partir des contours détectés grâce au filtre de Canny, j'ai ensuite cherché à detecter des lignes sur ces contours. J'ai pour cela utilisé la transformée de Hough

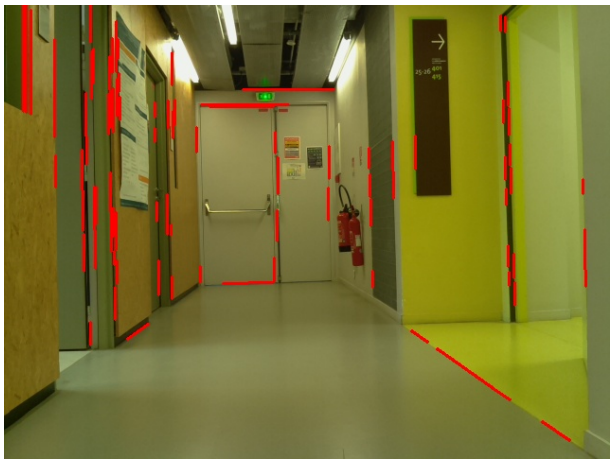

Figure 5 – Application de Hough

## 2.4 Détection de régions d'intérêts

J'ai ensuite voulu détecter les différentes régions de l'image afin de d'identifier l'aire qui délimite le sol ainsi que celle délimitant le plafond. J'ai pour cela voulu utiliser les algorithmes SURF(Speeded Up Robust Feature) / SIFT(Scale Invariant Feature Transform) utilisés pour la détection d'objets.
Ces algorithmes aurait pû être utilisés afin d'identifier des points caractéristiques du parallèlépipède, vu sous différentes orientations même. Mais pour l'utiliser il aurait fallu créer une base de données de pièces afin de pouvoir trouver des similitudes avec notre image. J'ai donc abandonné cette approche.

## 2.5 Détection de rectangle

J'ai ensuite voulu détecter le rectangle correspondant à une des faces du parallèlépipède, afin de par la suite pouvoir en déduire les dimensions de notre pièce.
Pour détecter les rectangles dans l'image j'ai d'abord effectué un seuillage, puis j'ai utilisé l'algorithme de détection de contours qui est déjà implémenté dans OpenCV, ensuite j'ai seulement gardé les 10 plus grand contours pour éviter de détecter de tout petits rectangles issus du bruit. Enfin à partir de ces contours j'ai détecté des rectangles.
On se retrouve donc avec une dizaine de rectangles, le plus grand faisant le tour de l'image.

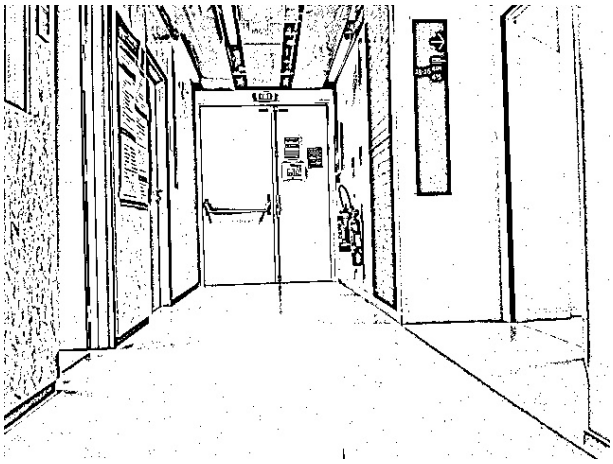

Figure 6 – Seuillage

Figure 7 – Tous les contours

Figure 8 – Les 10 plus grand contours

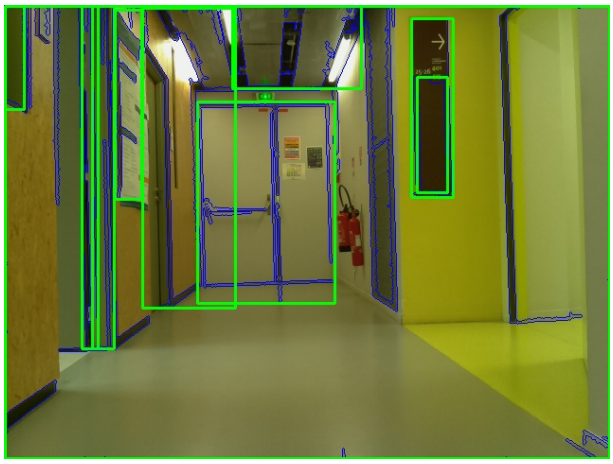

Figure 9 – Rectangles détecter

### 2.6 Identification du "'bon"' rectangle grâce au point de fuite

D'après nos hypothèses, on se place dans le cas où l'on observe la pièce depuis la perspective frontale, que l'on peut voir sur le schéma ci-dessous.
Dans cette perspective, deux points de fuite sont à l'infini, l'un pour les segments verticaux qui sont parallèle entre eux donc ne se croiseront jamais (d'où le nom de point de fuite à l'infini) et l'autre pour les segments horizontaux qui sont aussi parallèles. En revanche le troisième point de fuite est lui réel et se situe dans le rectangle.
Ainsi, grâce à la detection de point de fuite on peut en déduire lequel de nos 10 rectangles détectés précedemment est le bon.

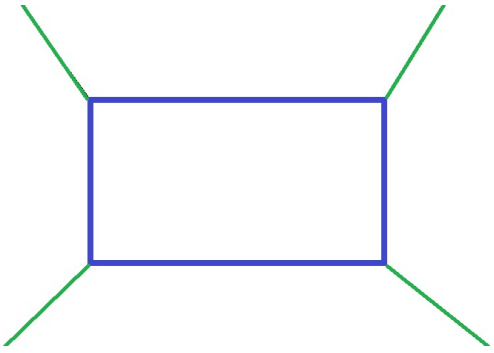

Figure 10 – Schéma de la pièce observée

Pour celà, on commence tout d'abord par détecter les segments de l'image, comme précedemment avec la transformée de Hough, correspondant aux lignes de fuites.
On élimine ensuite les segments verticaux et horizontaux (avec une petite marge) pour ne garder que ceux correspondant à notre point de fuite recherché.
A partir des segments restant on prend les deux plus longs qui ne sont pas parallèles et on calcule leur point d'intersection, puis on fait celà pour les autres segments. Le point d'intersection des plus longs segments est le plus probable d'être le point de fuite.
On prend le premier point d'intersection et on regarde si l'un de nos rectangle le contient : si oui, ce rectangle correspond à la face recherchée. Sinon, ce point était dû à du bruit lors de la détection de segments et on passe donc au point suivant.

# 3 Reconstitution d'un modèle de la pièce

## 3.1 Récupération des mesures

Nous sommes parvenus dans la partie précédente à identifier le rectangle du mur frontal sur chaque cliché. Nous allons pouvoir maintenant en déduire les mesures de la pièce à partir de deux clichés pris en avancant le robot d'une distance d. Nous allons obtenir ce résultat en deux étapes :

*Etape1* : calcul des angles apparents du mur. On peut déterminer ces angles en calculant la proportion de l'image qu'occupe le rectangle du mur. A partir de l'angle d'ouverture $\theta$, qui dépend de la focale, on en déduit l'angle apparent.

*Etape2* : calcul des dimensions réelles à partir de la distance d entre les deux clichés, et des angles apparents sur les deux clichés. Un détail des calculs mis en jeu est visible ci-dessous.

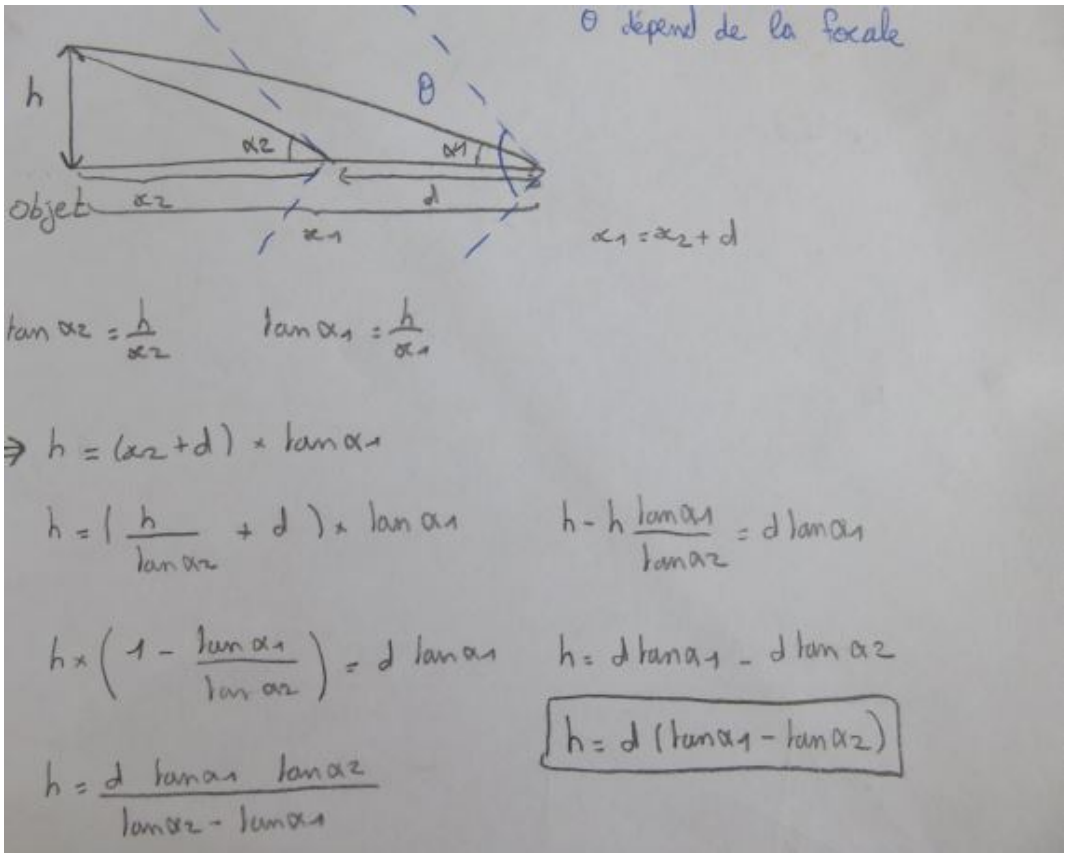


Figure 11 – Calcul d'une dimension à partir des angles apparents

## 3.2 Résultats obtenus

Pour tester cet algorithme de calcul de mesures de la pièce, j'ai effectué l'expérience dans le couloir du laboratoire. J'ai pris deux photos grâce au raspberry, en avancant le robot d'1m50 entre chacun des clichés. J'ai obtenu des résultats par l'algorithme ci-dessous avec une erreur inférieure à 10 % pour chacune des dimensions mesurée : dans un exemple où la distance du raspberry jusqu'au bout de la pièce était de 13m, j'ai obtenu 14m grâce au calcul. Les résultats finaux ont été arrondis au mètre près, pour les courtes distances il vaudrait sans doute mieux faire des arrondis au centimètre.

En revanche, lors d'une autre expérience, l'algorithme n'a pas réussi à détecter correctement les contours du mur frontal, ca n'a pas abouti à un résultat cohérent.
Il y a donc encore du travail à faire pour rendre cela plus fiable, par exemple en prenant plus que deux clichés ou en amélioration la détection des contours dans l'image.

### 3.3 Reconstruction en 3D

Tout ce que nous avons vu jusqu'à présent, du traitement d'images au calcul des mesures, a été implémenté en python : en effet, de nombreuses fonctionnalités de traitement d'image ainsi que de fonctions mathématiques y sont déjà implémentées.

En ce qui concerne la reconstruction en 3D de la pièce, j'ai utilisé l'outil JME3 sous Eclipse, qui est un outil de référence pour la visualisation 3D. J'appelle donc depuis mon programme Java mes scripts python qui écrivent les résultats, c'est-à-dire les 3 dimensions de la pièce dans un fichier. Depuis mon programme Java, je récupère ces informations et j'en génère un modèle 3D de la pièce grâce à JME3.

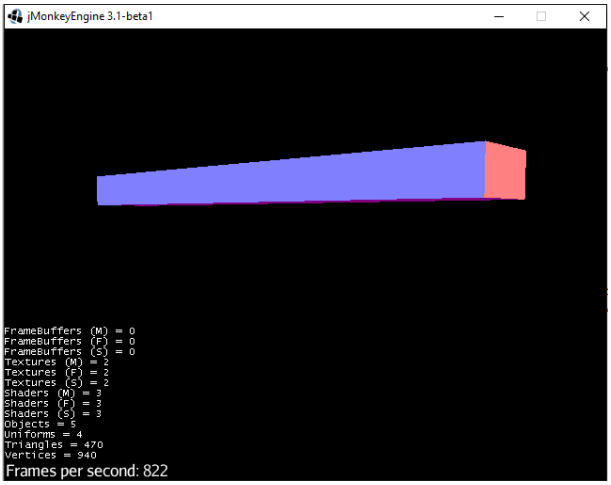


Figure 12 – JMonkey

## 4 Conclusion

J'ai au cours de ce rapport, proposé une approche pour mesurer les dimensions de la pièce entourant un robot, à partir de deux clichés pris par le robot. J'ai obtenu des résultats cohérents avec une précision raisonnable. En revanche, l'algorithme n'a pas fonctionné dans toutes les situations puisque les contours des murs n'étaient pas forcément bien identifiés. Il y a donc encore du travail pour rendre cette approche plus robuste.

Par ailleurs, je n'ai pas eu le temps de tester cette approche en condition de temps réel avec le raspberry, et je n'ai pas pu mesurer si notre approche était réellement plus économe en ressources qu'un algorithme classique, ni si elle serait viable dans le cas d'environnements ou pièces plus complexes. Ces considérations pourraient constituer des idées à creuser pour démontrer une véritable viabilité de cette approche.

## Références

[1] Kai Xu, Hanlin Zheng, Hao Zhang, Daniel Cohen-Or, Ligang Liu, and Yueshan Xiong. Photo-inspired model-driven 3d object modeling. *ACM Transactions on Graphics*, 30, 2011.

[2] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots : a survey. *bla*, 0000.

[3] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep : Extracting editable objects form a single photo. *bla*, 0000.