Localisation d'un robot à partir de la mesure des carreaux d'un carrelage

Bertrand Turck

Maitre de stage : Luc Jaulin
Tuteur : Fabrice Le Bars
Adresse : ENSTA Bretagne, Brest, France

13 octobre 2020

Remerciements

Tout d'abord, je tiens à remercier Monsieur Luc Jaulin qui m'a permis de faire un stage au sein du labratoire de robotique de l'ENSTA Bretagne et qui a rédigé la partie théorique de ce rapport.

Le travail sur le robot Saturne n'aurait pas non plus été possible sans les nombreux conseils et l'expertise de Monsieur Fabrice Le Bars pour toute l'électronique du robot, et bien plus.

Enfin, parce qu'il était mon camarade de stage et qu'il a énormément travailler sur notre robot, merci à Robin Sanchez pour toutes ces heures que l'on a passé à construire et améliorer ce robot.

Table des matières

1 Introduction **7**

2 Modélisation **9**
2.1 Introduction 9
2.2 Calibration de la caméra 9
2.3 Représentation d'une ligne 10
2.4 Paramétrisation d'un carrelage 10
2.5 Méthode pour trouver les paramètres d'un carrelage 11
2.5.1 Orientation 11
2.5.2 Translations 12
2.6 Utilisation des intervalles pour la localisation 12
2.6.1 Equivalence entre les carrelages 12
2.6.2 Contracteur 13

3 Implémentation sur le robot Saturne **15**
3.1 Présentation du robot 15
3.2 Caméra et OpenCV 16
3.2.1 Calibration de la caméra 16
3.2.2 Détection des lignes du carrelage avec OpenCV 17
3.2.3 Conversion d'une ligne de la forme cartésienne à normale 19
3.3 Intervalles et Ibex 20
3.3.1 Intervalles à contracter 20
3.3.2 Contracteur 20

4 Simulation **23**
4.1 Modélisation sous V-Rep 23
4.2 Architecture ROS 24
4.3 Résultats Simulés 25

5 Conclusion **27**

Introduction

Localiser un robot autonome est souvent compliqué à mettre en œuvre à cause du coût des capteurs (souvent plusieurs milliers d'euros pour un LiDAR ou une IMU) et de la puissance de calcul nécessaire pour traiter les informations reçues. Ce stage vise à tester l'utilisation d'une simple caméra pour repérer les lignes d'un carrelage et à utiliser des intervalles pour en déduire une estimation de la position du robot.

Ainsi, on pourrait alors aisément localiser un robot dans un hôpital ou encore dans un entrepôt dans lesquels du carrelage est déjà présent.

Modélisation

2.1 Introduction

On considère un robot équipé d'une caméra pour voir les carreaux du carrelage sur le sol, et se déplacant uniquement dans le plan horizontal. On considère notre robot comme étant un modèle char. On ne mesure pas la vitesse de notre robot, on ne la régule donc pas. Sa dynamique est alors décrite par les équations d'états suivantes :

$$\begin{cases} \dot{\mathbf{x}}(t) &= (cos(\theta), sin(\theta), u) \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \end{cases}$$

où x est l'état du robot correspondant à sa position (x_1, x_2) et son orientation (x_3).

L'entrée u sera calculée par un controleur en fonction de la position estimée par le traitement du retour vidéo de la caméra, transformation que l'on va détailler par la suite.

2.2 Calibration de la caméra

La première étape consiste à prendre en compte les caractéristiques de la caméra (distance focale, distortion de la lentille, position sur le robot, etc.) et les mesurer pour appliquer une transformation inverse pour obtenir une image exploitable. On observe qu'avant transformation, des lignes droites ne le paraissent pas (2.1).

FIGURE 2.1 – image de la caméra avant traitement (à gauche) et après traitement (à droite)

2.3 Représentation d'une ligne

Une ligne sur un plan peut être représentée par l'équation suivante :

$$x\,cos(\alpha) + y\,sin(\alpha) \;=\; d \tag{2.1}$$

qui est une forme normale faisant apparaitre les coordonnées polaire (α,d). Cette forme est associée à la transformée de Hough qui permet de détecter des lignes sur une image. Une ligne peut alors être représentée dans l'espace des lignes (α,d).

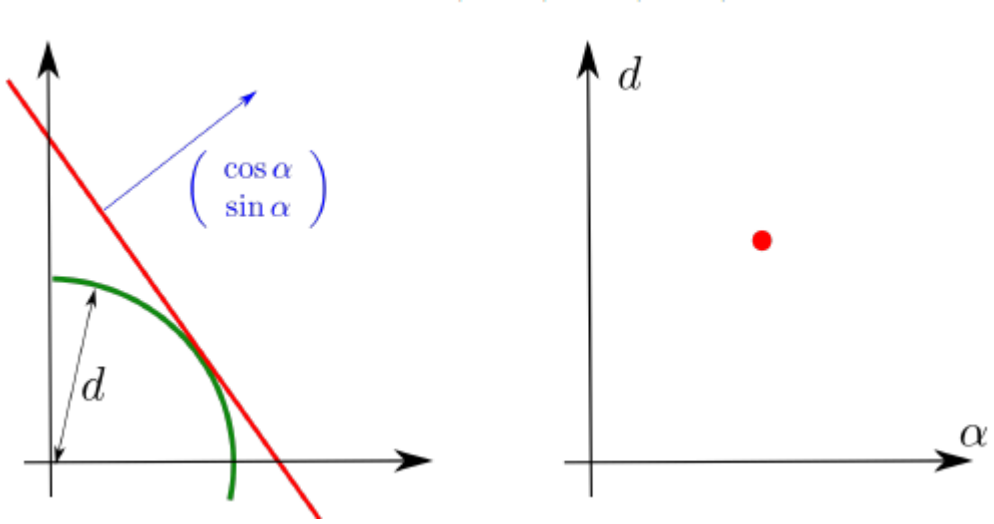

FIGURE 2.2 – représentation d'une ligne dans sa forme normale (crédit image : Luc Jaulin)

Une ligne est maintenant représentée par 4 paramètres (x, y, α,d), représentant successivement un déplacement horizontal, vertical et une orientation.

2.4 Paramétrisation d'un carrelage

Un carrelage est composé de deux ensembles perpendiculaires : un ensemble de lignes horizontales et un autre ensembe de lignes verticales. Au sein de ces 2 ensembles, toutes les lignes sont parallèles et équidistantes. On se place dans le cas unitaire, c'est-à-dire que la distance entre les lignes est de 1 mètre.

Un carrelage est parametrisé par un vecteur y de dimension 3 de la forme (y_1, y_2, y_3) $\in[-\frac{1}{2}, \frac{1}{2}]^2 x[-\frac{\pi}{4}, \frac{\pi}{4}]$ tel que le robot est translaté horizontallement de y_1, verticallement de y_2 et tourné de y_3.

2.5 Méthode pour trouver les paramètres d'un carrelage

On considère un ensemble L de lignes issues d'un traitement d'image. Ces lignes sont représentées sous une forme normale (α_i,$\mathrm{d_i}$). Parmis ces lignes se trouvent des lignes parasites qu'il faudra filtrer (outliers).

FIGURE 2.3 – des lignes en trop sont détectées en bas à gauche

2.5.1 Orientation

Le premier paramètre que l'on peut trouver est l'orientation du carrelage. En ramenant les angles de toutes les lignes à des angles compris entre -$\frac{\pi}{4}$ et $\frac{\pi}{4}$, grâce à un modulo par exemple, et en appliquant un filtre médian sur ces valeurs, on obtient l'orientation globale du carrelage que l'on notera $\bar{\alpha}$.

Maintenant, on peut découper notre ensemble L en 2 sous-ensembles $\mathrm{L_{hor}}$ et $\mathrm{L_{ver}}$ qui correspondent aux ensembles des lignes horizontales et verticales. On peut utiliser la propriété suivante :

$$\begin{cases} (\alpha_i, d_i) \in L_{ver} \; si \; cos(\alpha_i - \bar{\alpha}) = 0 \\ (\alpha_i, d_i) \in L_{hor} \; si \; sin(\alpha_i - \bar{\alpha}) = 0 \end{cases}$$

Aussi, on en profite pour enlever toutes les lignes qui ne sont ni parallèles ni perpendiculaires à l'orientation globale.

2.5.2 Translations

Tout d'abord, on applique une rotation d'angle -$\bar{\alpha}$ à nos lignes afin de n'avoir que des lignes horizontales et verticales.

On notera qu'après rotation, la valeur de d change.

Pour chaque ensemble L_{hor} et L_{ver}, on ramène les valeurs de d_i entre -$\frac{1}{2}$ et $\frac{1}{2}$, en utilisant un modulo encore une fois, et on applique un filtre médian sur ces valeurs.

On obtient alors nos 3 paramètres du vecteur y qui vont nous permettre de localiser notre robot.

2.6 Utilisation des intervalles pour la localisation

On remarque que pour une image d'un carrelage récupérée à un instant donné, on ne peut en déduire directement la position d'un robot. Cependant, connaissant la position a priori de notre robot à l'instant précédent, on peut en déduire une bonne approximation grâce à la théorie des intervalles.

2.6.1 Equivalence entre les carrelages

Le robot magenta et le robot bleu (figure 2.4) ne sont pas au même endroit et pourtant ils voient la même chose. Il y a ambiguité sur leur position. La relation entre les 2 vues des robots est une relation d'équivalence (réflexive, symétrique et transitive). Cette transformation est une combinaison de 2 translations (horizontale et verticale) d'un nombre de carreaux entier, et d'une rotation proportionnelle à $\frac{\pi}{2}$, de part la géométrie du carrelage.

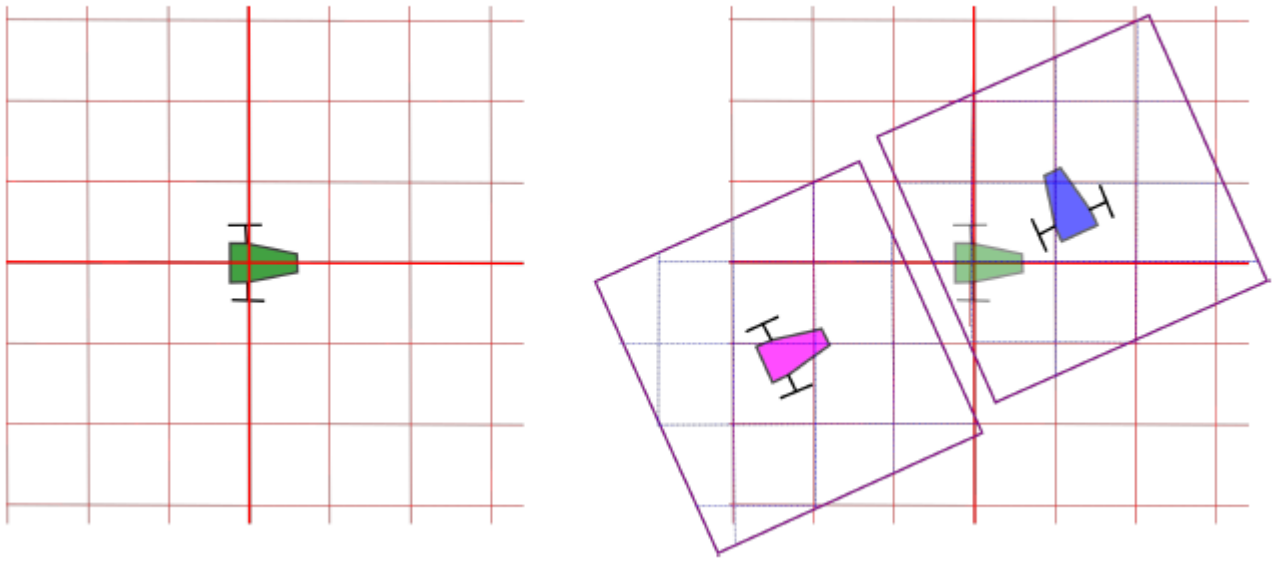

FIGURE 2.4 – 2 robots qui ont une position différente voient la même chose (crédit image : Luc Jaulin)

Notons y et z les paramètres de carrelage de 2 robots.
Alors y et z sont équivalents si et seulement si :

$$\begin{cases} y_1 - z_1 \in \mathbb{N} \\ y_2 - z_2 \in \mathbb{N} \\ \frac{y_3 - z_3}{\pi} \in \mathbb{N} \end{cases} \quad ou \quad \begin{cases} y_1 - z_2 \in \mathbb{N} \\ y_2 - z_1 \in \mathbb{N} \\ \frac{1}{2} + \frac{y_3 - z_3}{\pi} \in \mathbb{N} \end{cases} \tag{2.2}$$

2.6.2 Contracteur

L'état de notre robot est un vecteur de 3 intervalles. En pratique, on utilisera de l'odométrie pour avoir une première estimation de l'état de notre robot et nous pourrons ensuite contracter ces intervalles pour obtenir une meilleure estimation de l'état du robot.

Pour le contracteur, on utilise l'équation 2.2 et on la réécrit sous une autre forme en utilisant les propriétées des fonctions sinus et cosinus :

$$y \sim z \iff \begin{cases} sin\,\pi(y_1 - z_1) = 0 \\ sin\,\pi(y_2 - z_2) = 0 \\ sin\,(y_3 - z_3) = 0 \end{cases} \quad ou \quad \begin{cases} sin\,\pi(y_1 - z_2) = 0 \\ sin\,\pi(y_2 - z_1) = 0 \\ cos\,(y_3 - z_3) = 0 \end{cases} \tag{2.3}$$

avec y l'état du robot et z le paramétrage du carrelage vu par le robot.

Implémentation sur le robot Saturne

3.1 Présentation du robot

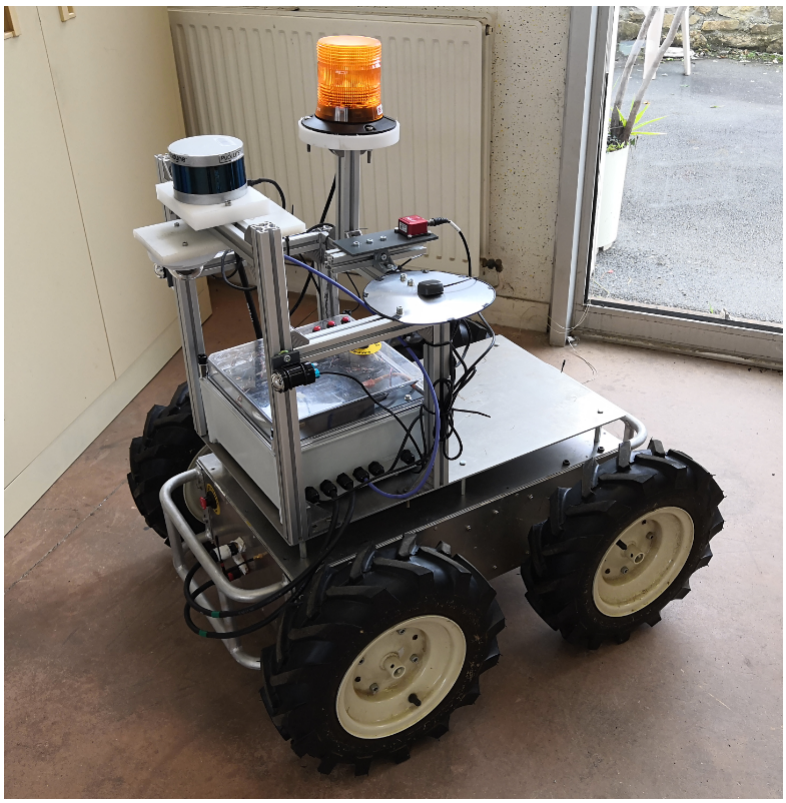

FIGURE 3.1 – Le robot Saturne et ses capteurs

Le robot Saturne, développé avec mon camarade Robin Sanchez, est un robot de type char dont la base mécanique a été créée par le pôle mécanique de l'ENSTA Bretagne. Une grande partie du stage a consisté à élaborer et construire l'architecture électronique du robot, à y installer différents capteurs (GPS, IMU, LiDAR, Caméra), à faire une interface ROS et enfin à développer des contrôleurs pour réaliser des missions simples. Pour les détails concernant Saturne, je vous renvoie vers le rapport de Robin Sanchez.

3.2 Caméra et OpenCV

Le robot Saturne embarque une caméra IP et peut donc récupérer un flux vidéo. Il est à noté que malgré un changement de réglages de la caméra (résolution, conversion du flux, etc.) il y a une forte latence d'environ 1 seconde, ce qui retarde le traitement et donc l'action du controleur. La caméra est installée avec un certain angle afin de voir le plus de lignes possible.

FIGURE 3.2 – La caméra du robot Saturne, avec son angle

3.2.1 Calibration de la caméra

La caméra est une caméra champ large (environ 120° d'après le fabriquant) et a donc une forte distortion. Des lignes droites se retrouvent alors tordues (figure 2 .1). L a bibliothèque de traitement d'image OpenCV propose une fonction pour appliquer un traitement inverse et retrouver une image non déformée.

$undistort(src,src, intrinsic, distCoeffs)$;

qui prend en paramètres une image à traiter, une image traitée, la matrice de la caméra et les coefficients de distortions. La matrice de la caméra fait directement intervenir la distance focale et les coordonnées du centre optique.

Pour trouver ces coefficients, OpenCV propose un protocole impliquant un échiquier (chessboard). Ce protocole est expliqué sur le site officiel d'OpenCV.

La caméra n'étant pas verticale dirigée vers le sol, il nous faut aussi trouver la transformation qui permet de tourner l'image. Nous utiliserons donc à nouveau notre échiquier. En effet, il est

facile de repérer les coins de ce dernier car ils sont à l'intersection du blanc et du noir, donc dans une zone de fort contraste où la dérivée de l'image est grande. Ainsi on peut en déduire les coordonnées de ces coins dans l'espace. A partir de 2 images, une avec l'échiquier posé sur le sol et une autre avec l'échiquier tenu dans l'axe de la caméra (figure 3.3), on obtient 2 nuages de points et on peut en déduire la transformation pour passer de l'un à l'autre.

FIGURE 3.3 – Calibration de la caméra

3.2.2 Détection des lignes du carrelage avec OpenCV

OpenCV est une bibliothèque trés complète de traitement d'image. De nombreuses fonctions sont implémentées, ainsi la solution présentée n'est peut être pas la plus optimale. La figure 3.4 représente la suite des traitements appliqués à nos images issues de la caméra.

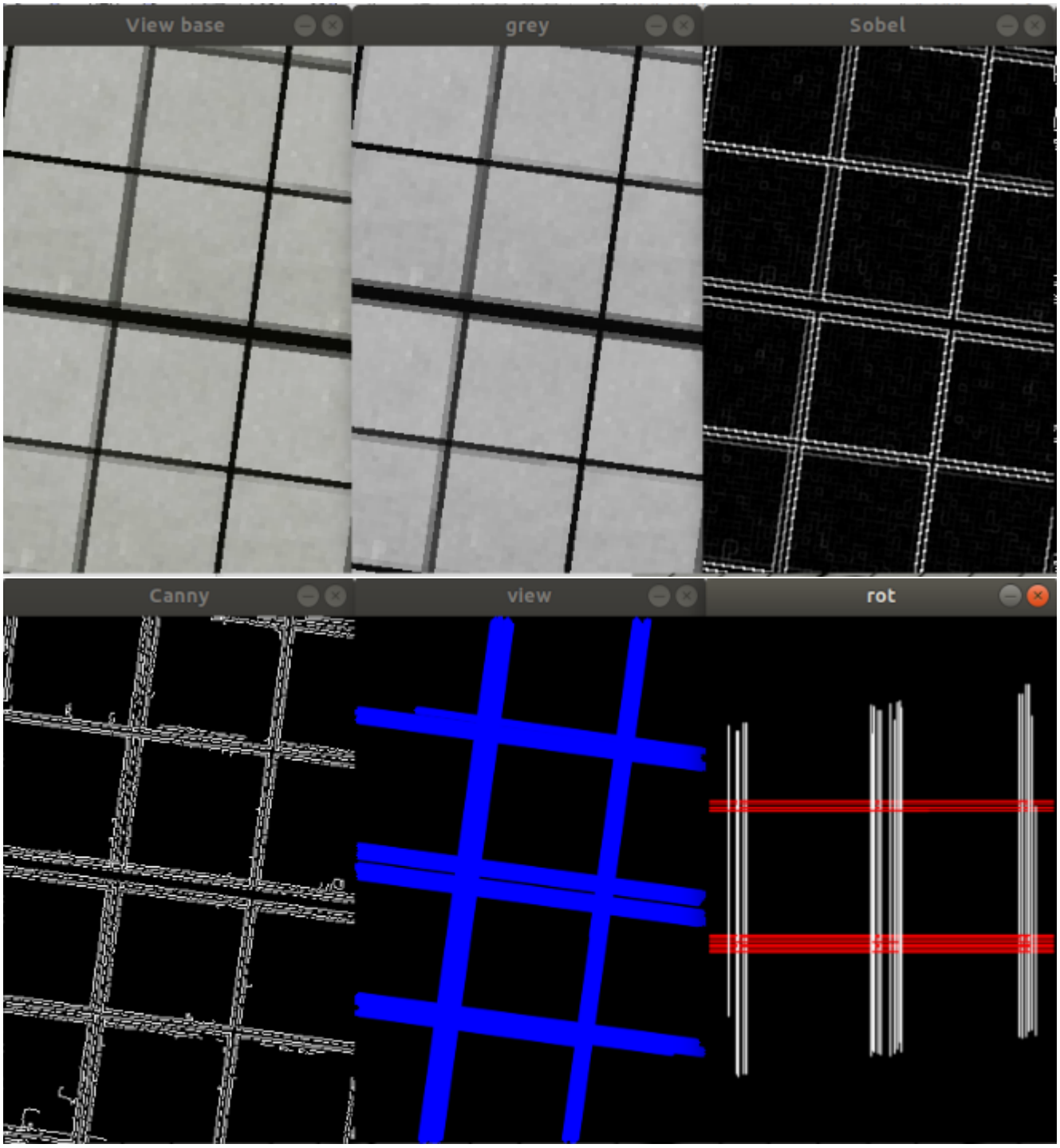

FIGURE 3.4 – Représentation des traitements successifs appliqués à nos images

La première étape consiste à reformer l'image (voir 3.2.1) et à la convertir en niveau de gris. Les lignes d'un carrelage étant un endroit de l'image où le niveau de gris change fortement, il est normal de penser à dériver l'image. Cette étape est réalisée grâce à un filtre de Sobel.

Ensuite, on applique un détecteur de contour de type Canny afin d'obtenir une image avec

des contours fins, et surtout obtenir une image binaire (où 1 correspond à un contour et 0 une absence de coutour).

Enfin, OpenCV implémente directement la transformée de Hough qui à un nuage de points associe un ensemble de segments qui passent par un maximum de points. Ces segments sont représentés par 2 points extrèmes, il faut donc les convertir dans la forme normale (équation 2.1) pour appliquer la méthode (section 2.5) et trouver les paramètres du carrelage.

3.2.3 Conversion d'une ligne de la forme cartésienne à normale

Soit A = (x_1, x_2) et B = (y_1, y_2) 2 points d'une droite à convertir dans la forme normale. En observant la figure 2.2, on en déduit que l'angle α est la direction de la droite :

$$\alpha = atan2(y_2 - y_1, x_2 - x_1)$$

Le paramètre d correspond à la distance entre une droite D et le point O origine du repère de coordonnée (0, 0). On pose M projeté orthogonale de O sur la droite D. Le paramètre d est donc la norme de OM.

L'aire A_{OAB}du triangle OAB est donné par 2 expressions :

$$\begin{cases} A_{OAB} = \frac{1}{2}||\overrightarrow{AO} \wedge \overrightarrow{AB}|| \\ A_{OAB} = \frac{1}{2} \times AB \times OM \end{cases} \iff OM = \frac{||\overrightarrow{AO} \wedge \overrightarrow{AB}||}{AB} \iff d = \frac{(x_2 - x_1) \times y1 - x_1 \times (y_2 - y_1)}{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

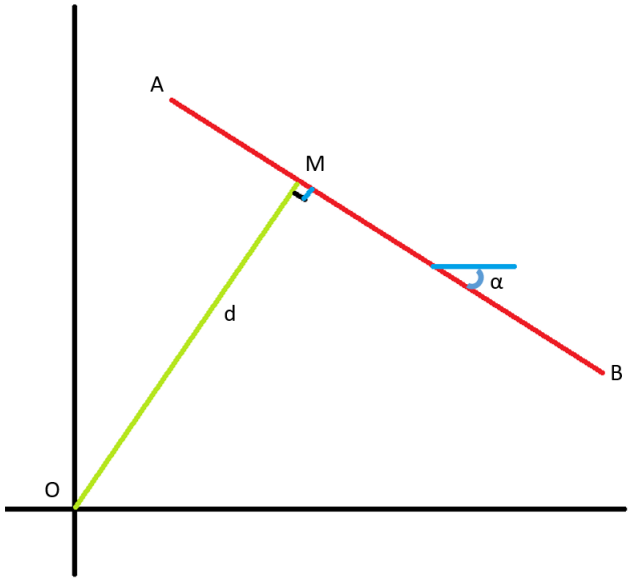

FIGURE 3.5 – Représentation d'une ligne

3.3 Intervalles et Ibex

Pour estimer la position du robot, la théorie des intervalles (section 2.5) est utilisée et implémentée en utilisant la bibliothèque Ibex.

3.3.1 Intervalles à contracter

La position du robot est estimée à chaque nouvelle image reçue. On utilise seulement la mesure des paramètres du carrelage pour se repérer (pas de fusion avec une mesure de vitesse, d'odométrie ou encore de GPS). Aussi, on se doute que le robot s'est déplacé après une mesure mais nous n'avons aucune idée de l'intensité du mouvement. Aussi, on considère que le robot fait des petits mouvements, plus petits que la taille d'un carreau entre chaque mesure. On en déduit ainsi que notre robot n'a pas bougé de plus d'une taille de carreau et on peut faire une estimation grossière de la position pour la contracter ensuite.

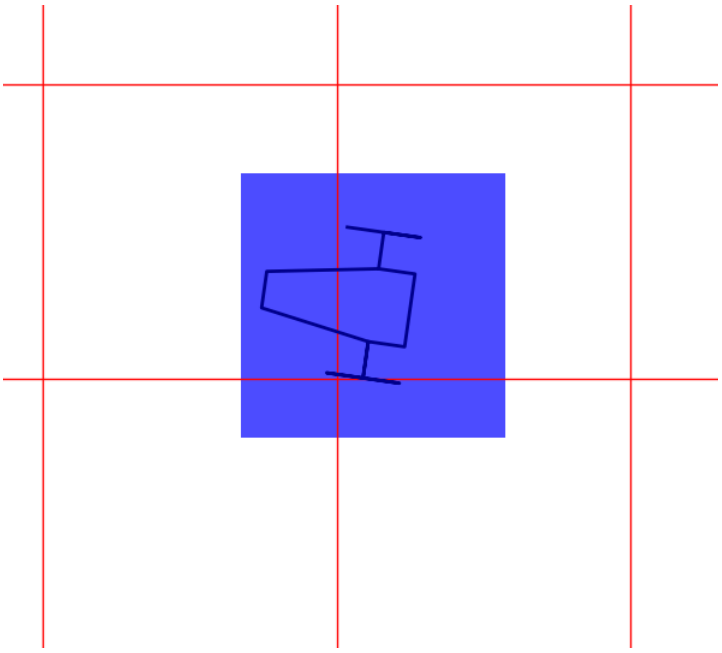

FIGURE 3.6 – Connaissant la position du robot, sa position à l'instant suivant sera dans le carré bleu

3.3.2 Contracteur

Pour créer notre contracteur, on reprend directement les équations trouvées précedement (équation 2.3).

On notera que ces dernières équations ont été obtenues dans le cas normalisé (carreaux de 1 mètre par 1 mètre), il faut donc faire intervenir la taille d'un carreau pour normaliser la position avant de contracter.

Algorithme 3.1 code pour créer nos contracteurs

```
ibex : :Function f1("x[3]", "y[3]", "(sin(pi*(x[0]-y[0])) ; sin(pi*(x[1]-y[1])) ; sin(x[2]-y[2]))") ;
ibex : :Function f2("x[3]", "y[3]", "(sin(pi*(x[0]-y[1])) ; sin(pi*(x[1]-y[0])) ; cos(x[2]-y[2]))") ;
ibex : :CtcFwdBwd c1(f1) ;
ibex : :CtcFwdBwd c2(f2) ;
```

Simulation

4.1 Modélisation sous V-Rep

Notre robot a été modélisé avec le logiciel V-Rep comme un simple modèle char. Il est équipé d'une caméra à l'avant, verticale orientée vers le bas. Comme sur le robot Saturne, on contrôle indépendamment les moteurs gauches et droits afin de réguler le cap du robot. Le robot est placé sur un carrelage infini pour tester les algorithmes.

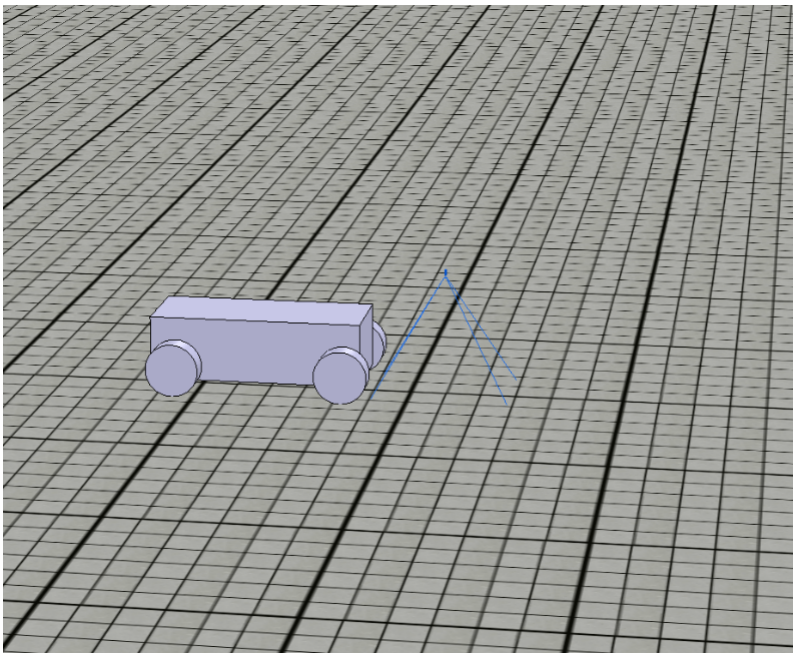

FIGURE 4.1 – Le robot modélisé sous V-Rep

4.2 Architecture ROS

L'utilisation de ROS nous permet de faire dialoguer facilement les différentes parties du programme. En outre, ROS permet aussi de faciliter la réutilisabilité du code (il suffit en effet de modifer quelques coefficients dans le controleur pour passer d'un robot simulé à un robot réel).

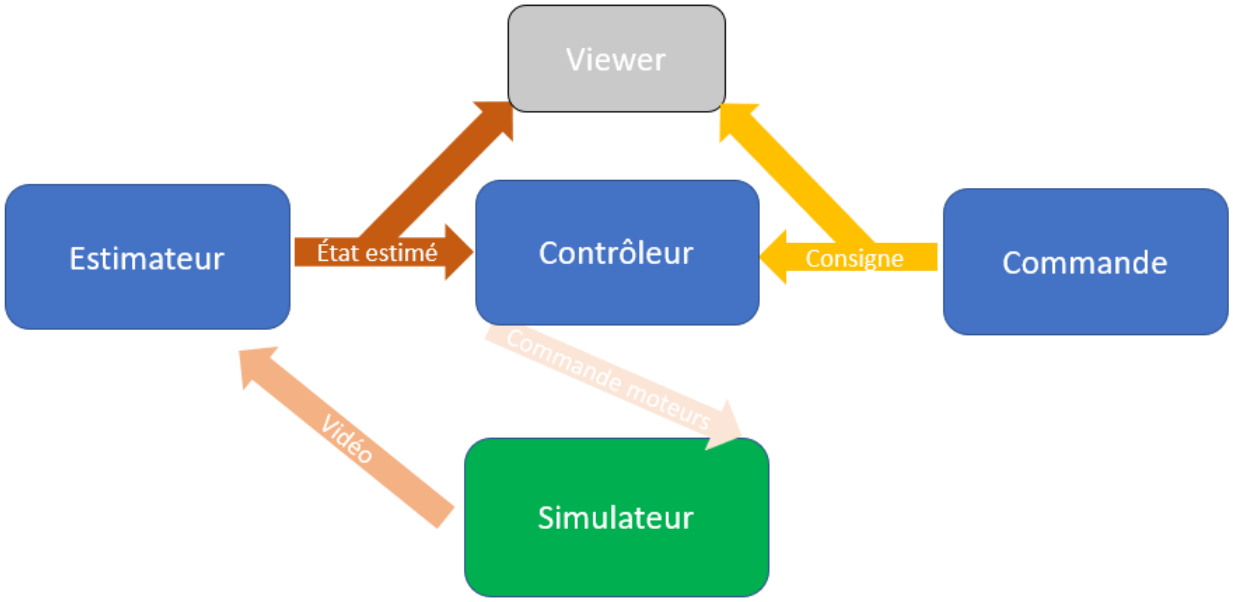

FIGURE 4.2 – Graphique de l'architecture ROS

L'architecture est centrée autour du contrôleur qui reçoit d'une part une position consigne (par exemple, les coordonnées consécutives d'une courbe de lissajous). D'autre part, il reçoit l'état estimé du robot et envoie une consigne aux moteurs en conséquence. Le noeud viewer permet de visualiser sur une même figure la position estimée, la position réelle et la consigne.

4.3 Résultats Simulés

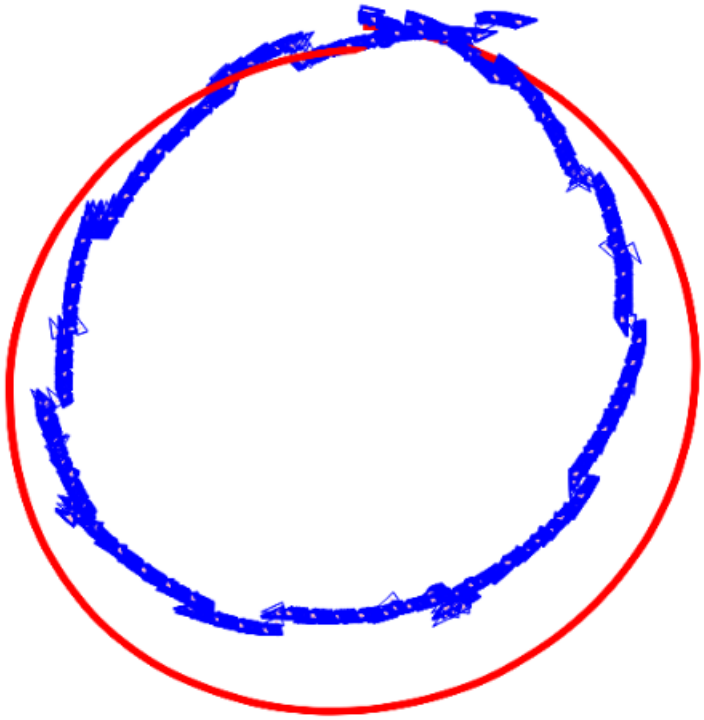

FIGURE 4.3 – Test de l'estimateur (position réelle en rouge, estimée en bleu)

Lorsque le robot fait un cercle, on observe que la position estimée (en bleu) suit globalement la forme attendue (en rouge) mais il y a toujours un décalage. On remarquera que la position estimée fait des sauts de temps en temps (en translation et en angle). Ce problème est dû au traitement d'image, parfois les lignes sont mal repérées et donc les paramètres du carrelage sont biaisés.

Le décalage observé est une conjonction de plusieurs facteurs. Tout d'abord, on considère que la ligne entre les carreaux est négligeable alors qu'en pratique elle ne l'est souvent pas. Aussi, lors du traitement d'image, on renseigne la taille d'un carreau en pixel sur l'image (pour faire une échelle entre la longueur en pixel d'un carreau sur l'image et une longueur réelle). Cette mesure est plutôt approximative et source d'erreur.

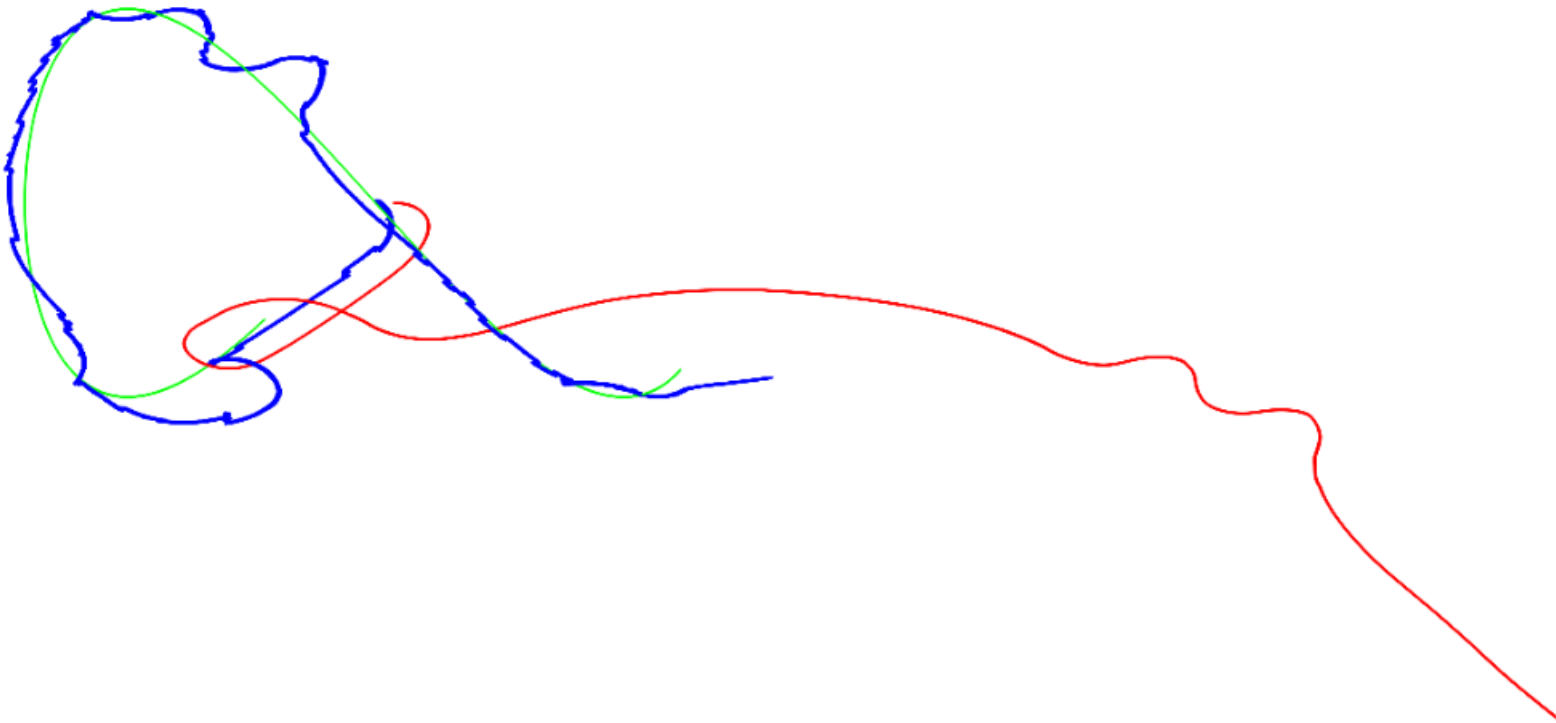

FIGURE 4.4 – Test de l'estimateur et du contrôleur (état estimée en bleu, réelle en rouge, consigne en vert)

Lorsque le robot doit faire une courbe de lissajou, on observe que la trajectoire réelle (en rouge) n'est pas du tout correct. En effet, à plusieurs moment il y a un saut dans l'angle estimé. Lorsque le robot arrive sur des angles proportionnels à $\frac{\pi}{4}$, il y a un saut car l'angle observé est compris entre $-\frac{\pi}{4}$et $\frac{\pi}{4}$. Cependant, en pratique, on observe que l'angle saute plusieurs fois. Même en ayant filtré, d'autres sauts apparaissent quand même.

Conclusion

Lors de ce stage, il a été montré que l'utilisation des intervallesest possible dans un problème de localisation. En outre, l'utilisation d'une caméra seule, bien que théoriquement acceptable, est beaucoup trop soumise aux imperfections des images. Rajouter d'autres capteurs (IMU pour avoir un cap, odométrie pour approcher plus précisement la position) pourrait permettre d'améliorer grandement les résultats et de rendre le système plus robuste.

Aussi, du travail est encore à faire pour réaliser un traitement d'image satisfaisant. En effet, lors des quelques essais avec le robot réel, le traitement d'image s'est révélé trés sensible aux variations de luminosité, parfois aucune ligne ne sont détectée alors que l'oeil humain les perçoit correctement.

Ce stage fut pour moi très enrichissant. En effet, j'ai pu toucher à de nombreux domaines de la robotique : fabriquation d'un robot à partir d'une base roulante, implémentation d'algorithmes, théorie pour la localisation, simulation, etc. Cela m'a permis d'apprendre beaucoup et de réappliquer toutes les notions vues en cours d'année. Ce stage m'a vraiment prouvé que j'ai choisi la bonne voie pour moi : la robotique.