RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

ÉCOLE NATIONALE POLYTECHNIQUE
DÉPARTEMENT D'AUTOMATIQUE

CDTA

PROJET DE FIN D'ÉTUDES EN VUE DE L'OBTENTION DU DIPLÔME
D'INGÉNIEUR D'ÉTAT EN AUTOMATIQUE

THÈME :

Navigation référencée vision et contrôle d'un Robot mobile de type voiture

Réalisé Par :
LOUNES SELMAN AHMED AMIN
BOUSRI ABDERRAOUF

Encadreur :
Mr.Omar STIHI.
A.KHELLOUFI

Juin 2015

Ecole Nationale Polytechnique – Rue des frères Oudek, Hacen Badi, El Harrach, ALGER 16200.

العنوان: التحكم في رؤية والسيطرة على تحرك آلي من نوع سيارة

تهدف مذكرة التخرج هذه الى التحكم في روبوت من نوع سيارة آلية Robucar (قاعدة التجارب بمركز تنمية التكنولوجيات المتقدمة **CDTA**). وذلك لكي تتمكن من متابعة مسارها وتفادي العراقيل في محيط ثابت عن طريق استخدام نظام تشغيل الروبوت كأداة **ROS**

الحل الذي اعتمدناه هو تطبيق المنطق الغامض للتمكن من الاستفادة من القوة والفعالية التي يتمتع بها. طوال فترة البحث انصب اهتمامنا على إمكانية تجميع منظم غامض من أجل تزويد السيارة الآلية به **المفاتيح** : آلي متحرك **,**تحكم مرئي **,**المنطق الغامض **,** نظام تشغيل الروبوت **ROS .**

Titre : Navigation référencée vision et contrôle d'un Robot mobile de type voiture

Résumé :

Ce travail a pour objectif, la synthèse d'une commande référencée vision pour un robot mobile de type voiture Robucar (Plateforme expérimentale du CDTA), pour qu'il puisse accomplir les tâches de suivi de trajectoire et d'évitement d'obstacles dans un environnement statique avec l'utilisation de ROS (Robot Operating System) comme outil.

La solution retenue consiste à faire appliquer une commande floue pour pouvoir bénéficier de la robustesse et des performances qu'elle présente.

Tout au long de ce travail notre préoccupation été de pouvoir synthétiser un régulateur floue pour ensuite l'implémenter sur le Robucar.

Mots Clés : robot mobile, asservissement visuel, logique floue, ROS (Robot Operating System), évitement d'obstacles, suivie de trajectoire.

Title : Image based control of a mobil robot car type.

Abstract :

This work aims, the synthesis of an image based control for a mobile robot car type (Robucar). This robot has to perform the tasks of a path followed with static obstacles avoidance. Using robotics operating ROS. The solution adopted is to enforce a fuzzy control to ensure the robustness and performance of the system. Throughout this work our our main objective to synthesize a fuzzy to synthesize a fuzzy controller to implement it on the robucar.

Key words : fuzzy logic, ROS (Robot Operating System), obstacle avoidance, path followed, Image based control.

Remerciements

Au terme de ce travail,

Nous adressons nos remerciements à l'ensemble des enseignants de l'École Nationale Polytechnique spécialement ceux du département du Génie Automatique, pour leur encadrement, appuie scientifique, disponibilité ainsi que pour tout le savoir qu'ils nous ont transmis durant tout au long de notre formation. ,

Nous remercions aussi Monsieur Rachid ILLOUL, enseignant à l'École Nationale Polytechnique, d'avoir accepté d'examiner et évaluer ce travail.

Nous tenons également à remercier Monsieur A.ACHOUR, enseignant à l'École Nationale Polytechnique, de nous avoir fait l'honneur de présider ce jury.

Nous remercions également tous ceux qui nous ont soutenus et ont contribués de loin ou de près à la réalisation de ce travail.

A mes Parents

A mes sœurs

A toute ma famille

A Abderraouf

A tous mes amis

Selman Ahmed Amin

A Maman.
A ma famille.
A Selman.

Abderraouf Bousri.

Table des matières

Table des figures

Liste des tableaux

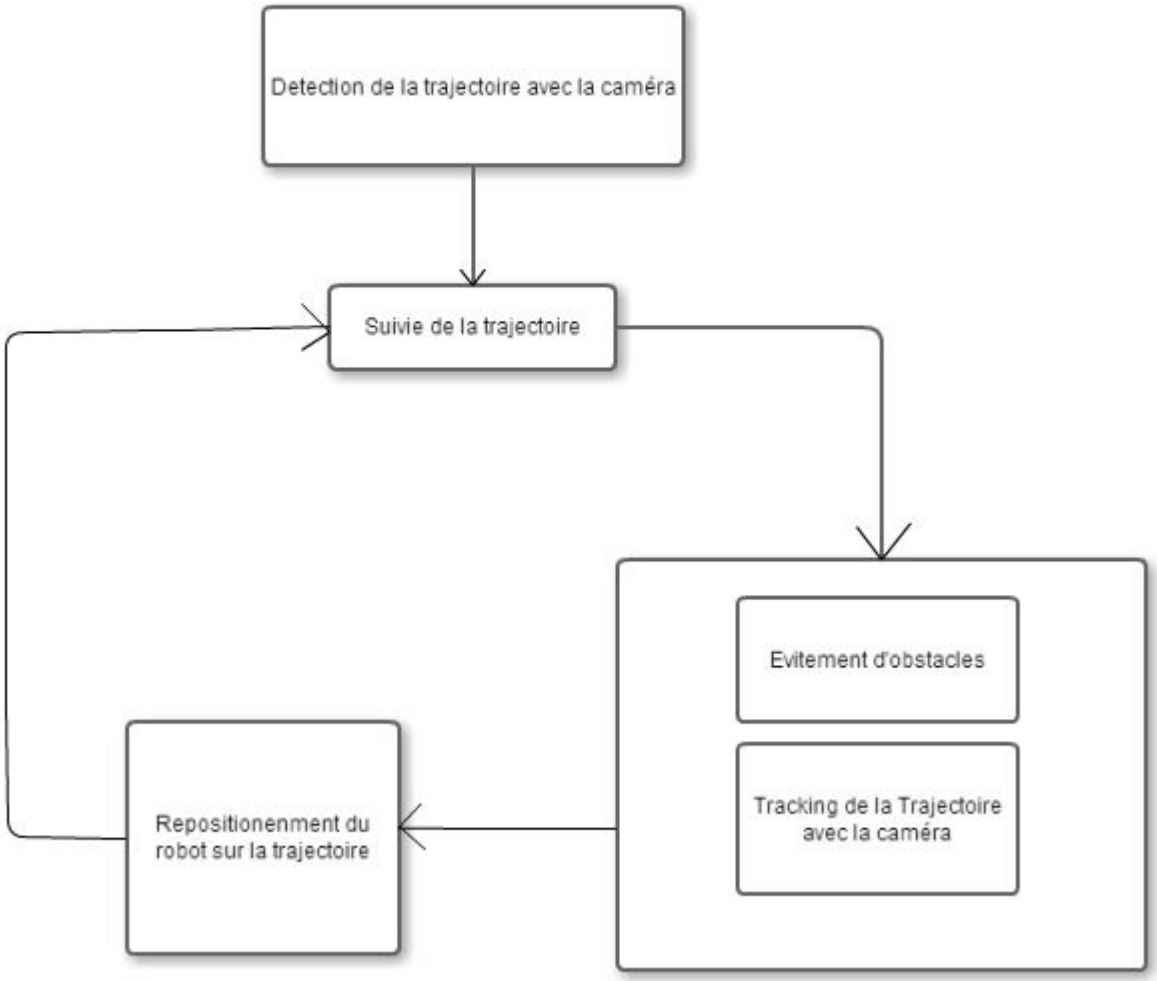
Depuis la nuit des temps l'homme a toujours essayé de réduire ses efforts en exploitant autrui, les circonstances et les faits historiques ainsi que sa curiosité lui ont permis d'inventer, de développer et ainsi exploiter des machines.

Le but de ce travail est de réduire les efforts tant mentales (décisions) que physiques (conduite) d'un conducteur, pour cela nous nous sommes orientés à la commande par logique floue car elle nous permet d'implémenter le raisonnement et ainsi les bons réflexes qu'un être humain aurait eu lors d'une situation déterminée.

Nous commencerons par présenter le robot de type voiture, sur lequel nous allons implanter la commande, ainsi que les capteurs embarqués sur celui-ci, nous présenterons aussi ROS (Robot Operating System) qui facilitera l'intercommunication entre le niveau haut (algorithme de commande) et le niveau bas (robot et capteurs).

Nous présenterons par la suite une méthode de détection de trajectoire avec une caméra pour pouvoir ensuite réaliser un asservissement visuelle 2D avec une commande par logique floue.

Nous entamerons par la suite la partie évitement d'obstacle où nous présenterons une méthode simplifié que nous avons mise au point cette partie sera liée avec la commande du support de la caméra afin de pouvoir garder la trajectoire dans le champs visuel lors de l'évitement d'obstacle. Nous finirons, enfin, par présenter une méthode de repositionnent du robot sur la trajectoire.


Detection de la trajectoire avec la caméra
Suivie de la trajectoire
Evitement d'obstacles
Tracking de la Trajectoire avec la caméra
Repositionenment du robot sur la trajectoire

Chapitre 1

Présentation du matériel

1.1 Présentation du Robucar

Le Robucar est un prototype de petit véhicule électrique construit sur la base d'un châssis tubulaire des petites voitures utilisées dans les terrains de Golf. Il est équipé d'un laser et d'une caméra qui lui permettent de suivre le marquage au sol et d'éviter les obstacles. Il comporte une architecture parallèle permettant le développement d'applications temps réel. Ses caractéristiques générales sont résumées dans le tableau suivant (Tableau : 1.1).

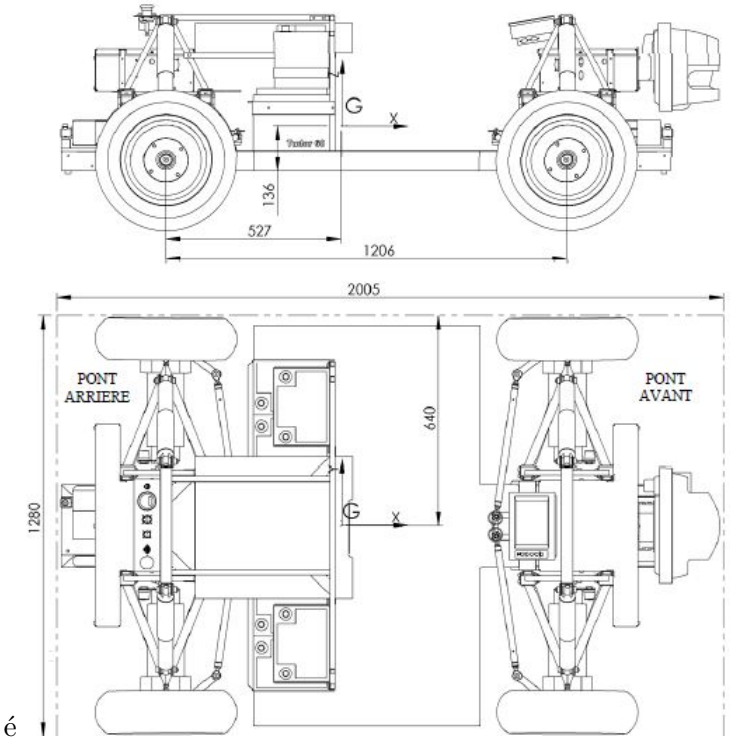


FIGURE 1.1: Plan d'ensemble du Robucar (carrosserie et siège enlevés)

Caractéristiques du Robucar	
Longueur	1.836 (m)
Largeur	1.306 (m)
Hauteur	0.616 (m)
Poids total avec Batteries	310 (Kg)
Vitesse maximale	18 Km/h (5 m/s)
Autonomie	2 Heures d'utilisation Continues
Capacité d'accueil	2 personnes
Motorisation	4 Moteurs électriques de 1200(W)
Roues Motrices	4 Roues Motrices et Directrices
Conduite	Automatique et Manuelle

TABLE 1.1: Caractéristiques du Robucar

1.1.1 Modèle cinématique du Robucar

Pour pouvoir déterminer le modèle cinématique de notre robot mobile qui est un robot non holonome, on doit calculer les composantes du vecteur vitesse instantanée d'un point M de la plateforme mobile, ainsi que la vitesse angulaire de celle-ci par rapport à l'axe des z. Soit le point M de cordonnées (x,y), centre de l'essieu arrière du robot et soit ϕ l'angle de braquage des deux roues avant, θ l'orientation du robot par rapport à la référence (axe des x) et L est la distance entre l'essieu arrière et avant (voir figure :1.2).

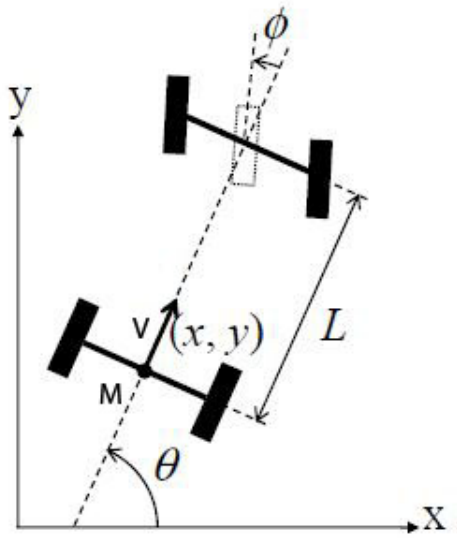


FIGURE 1.2: Modèle du Robucar. [9]

Si on considère les hypothèses simplificatrices suivantes :

- parmi les quatre roues du robucar, seulement les deux d'avant sont motrices, donc directrices.
- Le robot mobile est considéré comme un véhicule rigide qui évolue dans un plan horizontal
- les quatre roues du robucar roulent sans glisser sur le sol.
- la vitesse et l'angle de braquage envoyées au robucar représente respectivement la vitesse et l'angle de braquage réelles des roues du robucar.

Le modèle cinématique du Robucar sera le suivant [9]

$$\begin{cases} \dot{x} = & v.cos\theta \\ \dot{y} = & v.sin\theta \\ \dot{\theta} = & \frac{v}{L}.tan\phi \end{cases} \tag{1.1}$$

Contrairement à l'intégrale qui donne une information sur le passé, car elle ne représente qu'une somme des états passés, la dérivé donne une information sur le future. ainsi on peut déduire que le modèle discret correspondant est le suivant :

$$\begin{cases} X(n+1) & = & X(n) & + & T_{ech}.v.cos(\theta(n)) \\ Y(n+1) & = & Y(n) & + & T_{ech}.v.sin(\theta(n)) \\ \theta(n+1) & = & \theta(n) & + & \frac{T_{ech}.v}{L}tan(\phi(n)) \end{cases} \tag{1.2}$$

Où T_{ech} représente le temps d'échantillonnage choisit.

Remarque :

- L'angle de braquage des roues du robucar est limité entre $+20°$ et $-20°$.
- Pour commander le robucar en position, on doit choisir ϕ (l'angle de braquage des roues directrices) et v (module du vecteur vitesse instantané des deux roues avant), car c'est seulement à travers ces deux paramètres qu'on peut intervenir.

Le robucar est muni de différents capteurs proprioceptifs et extéroceptifs (voir annexe), parmi les capteurs extéroceptifs le capteur le plus important et le capteur laser LMS511.

1.2 Le Capteur laser LMS511

Pour accomplir des taches de navigation, de cartographie (Mapping) et d'évitement d'obstacles, le Robucar est muni d'un capteur laser du type sick-LMS511 (voir figure : 1.3) placé au milieu de la face avant du robot.

1.2.1 Caractéristiques

Le capteur laser LMS511 émet des impulsions laser qui sont réfléchies par un objet. Le temps mis entre l'émission et la réception de l'impulsion est directement

FIGURE 1.3: Le capteur laser LMS511.

proportionnel à la distance entre le laser et l'objet. les différentes connecteurs utilisés pour la transmission des données et pour l'alimentation du capteur sont illustrés dans la figure 1.4.

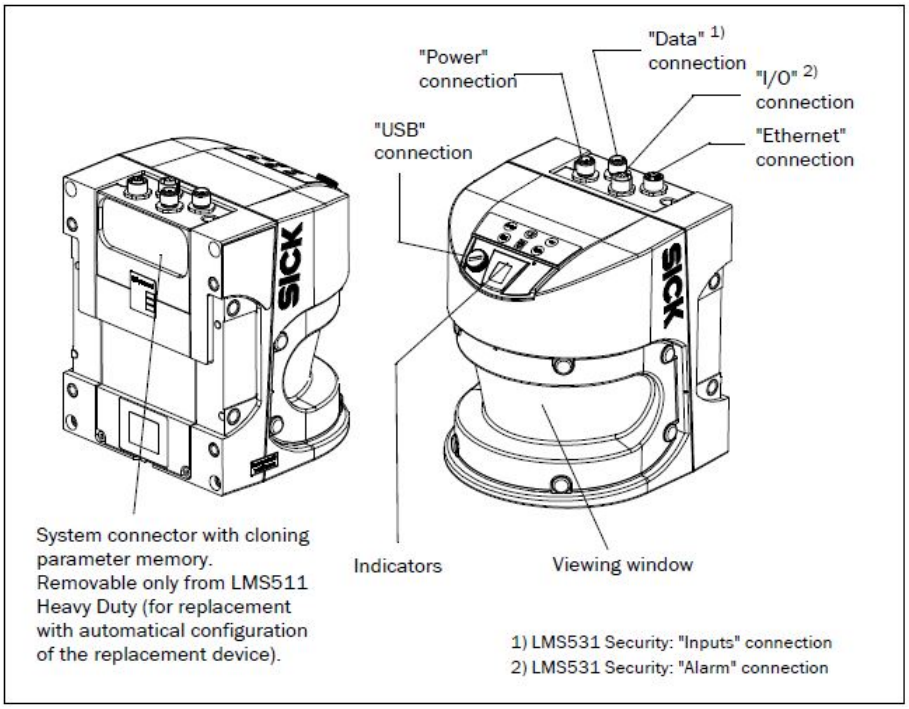


FIGURE 1.4: Vue extérieure du LMS511/581/531 [1].

Les différentes caractéristiques de ce capteur sont fournit par le constructeur sur sa fiche technique (data sheet)[1], elle sont représentées dans le tableau (1.2).

Caractéristiques du capteur laser LMS511	
Ouverture angulaire maximale	190°
Résolution angulaire	0.25/0.5/1°
Fréquence de balayage	25/35/50/75/100 Hz
Temps de réponse	13 ... 520 ms
Résolution	10(mm)
Température d'utilisation	–30°C ... +50 °C
Portée	plus de 80(m)
Interface de données	RS232/-422, USB
Vitesses de transmission des données	500 kb/s
Tension d'alimentation	19.2/24/28.8 V
Poids	3.7(kg)
Dimensions Hauteur-Largeur-profondeur	185-155-160(mm)
Courant maximale en charge à 24V (DC)	1.9 A
Consommation maximale en charge à 24V (DC)	50(W)

TABLE 1.2: Caractéristiques du capteur laser LMS511

Dans le cas du Robucar le capteur laser LMS511 est placé à une distance de 30 cm du sol, l'ouverture angulaire maximale est fixée à 180°. au lieu de 190° disponible par default (Fig 1.6). On peut aussi choisir la fréquence de balayage parmi les fréquences qui existes (voir tableau :1.2).

1.2.2 Principe de fonctionnement

Le LMS511 émet des faisceaux laser pulsés à l'aide d'une diode laser. Si le faisceau est réfléchi par un objet cible il sera détecté par le capteur.

La distance entre le LMS511 et le bord d'un objet est calculée par le temps nécessaire pour que le faisceau pulsé soit réfléchi et reçue par le capteur. Ce principe est appelé en anglais `Time of flight` (voir figure : 1.5), ce principe est aussi utilisé dans les systèmes de radar.

Les faisceaux laser émis sont déviés au moyen d'un miroir rotatif interne, en balayant l'environnement d'une manière circulaire. Les mesures sont déclenchées de façon régulière en utilisant un codeur angulaire.

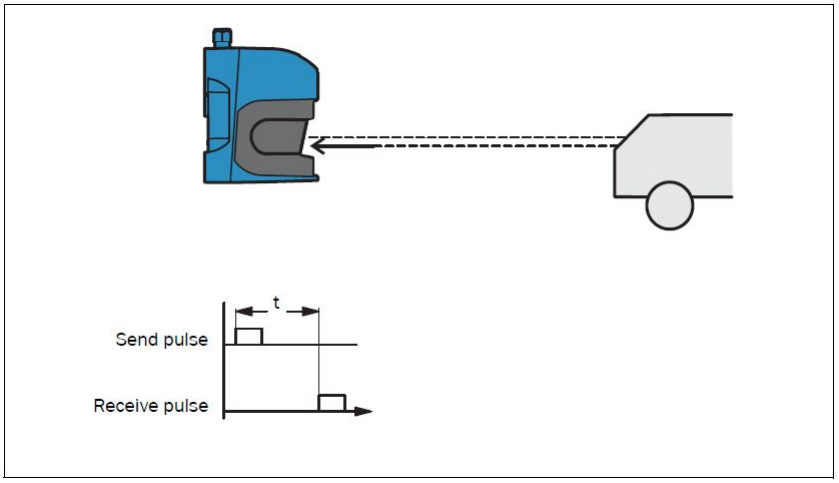


FIGURE 1.5: Principe de fonctionnement de la mesure du `Time of flight` [1].

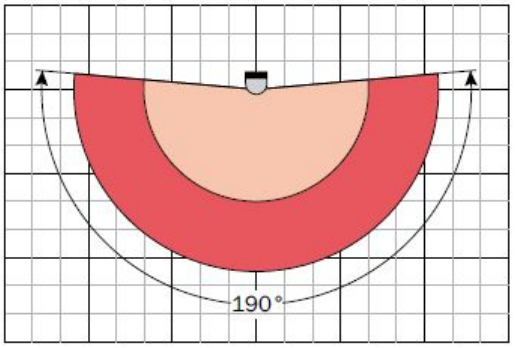


FIGURE 1.6: Ouverture angulaire maximale du LMS511 [1].

1.2.3 Récupération des données du LMS511

Le LMS511 est un capteur de mesure laser sans contact qui numérise le périmètre qui l'entoure radialement avec une ouverture angulaire maximale de 190° (voir figure : 1.6) et sur un plan unique à l'aide des impulsions de lumière (voir figure : 1.7). Les mesures récupérées du LMS511 permettent de localiser les bords d'un objet quelconque situé à une distance inférieure ou égale à la portée du laser (voir tableau :1.2), et qui ont une hauteur de 30 cm par rapport au sol (hauteur du capteur laser dans le Robucar), donc si un faisceau laser émis est réfléchie par un objet cible la position d'un point du bord de cet objet est exprimée en coordonnées polaires (D, γ).tel que :

- D : représente la distance entre le centre du capteur laser (centre d'émission des impulsions) et un point de l'objet.
- γ : représente l'angle correspondante à cette distance.

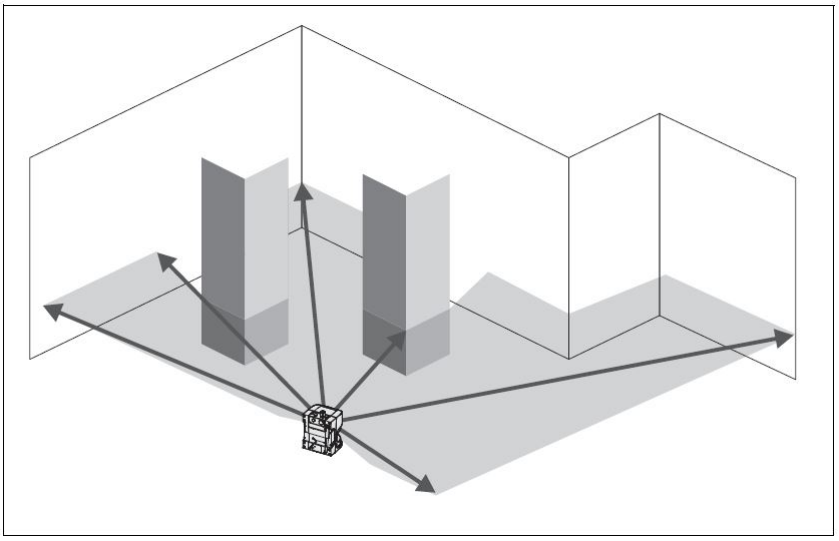


FIGURE 1.7: Principe de mesure du LMS511 [1].

Les données récupérées du capteur laser (LMS511) sont exprimées sous forme d'un vecteur de dimension (n x 1). où n est égale à 180 multiplié par la résolution angulaire (voir tableau :1.2). Donc pour une résolution angulaire de 0.5 °, la dimension du vecteur sera $n = 360$. d'un point de vue algorithmique ce vecteur peut être représente sous forme d'un tableau de n valeurs qu'on peut le parcourir via un indice **i** ou i=0,1...n. où l'angle $\gamma = 0$ correspond à la distance ayant l'indice i=0. ainsi pour connaitre l'angle qui correspond a l'une des distances du tableau il suffit de multiplier l'indice i par la résolution angulaire choisit pour le capteur laser.

$$\gamma \quad = \quad Res_{angLMS} \quad \times \quad i \tag{1.3}$$

Remarque :

• Le LMS511 ne peut pas voir à travers les objets lors du scan.

1.3 La CAM

Le Robucar est dotée d'une camera CCD SONY EVI 400/401 montée sur une platine commandable en lacet sur l'avant du robot. pour avoir une meilleure acquisition de l'image on l'a remplacé par une autre camera USB de type **logiteque** (fig : 1.8).

FIGURE 1.8: Logitech USB Camera.

1.3.1 Modélisation de la Camera

Pour la modélisation de la camera plusieurs modèles décrivant le processus de formation des images existent. Le plus simple est le modèle dit sténopé ou `pin-hole camera model` dans la littérature anglo-saxonne. Ce dernier est couramment utilisé en traitement d'image (voir figure : 1.9). le modèle sténopé prend pour hypothèse que tous les rayons passent par un seul point [6] ce point s'appel `centre optique` (point C, fig :1.9), donc un point p de l'espace ayant comme coordonnées $(x \quad y \quad z)^T$ va étre représenté par le point P dans le plan de l'image de coordonnées métrique $(X, Y)^T$ en utilisant la transformation 1.4

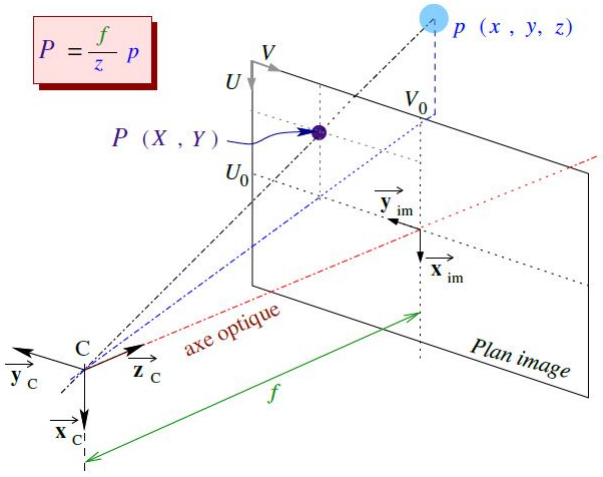


FIGURE 1.9: Modèle sténopé de la caméra.[6]

Où f représente la distance focale de la caméra.

Remarque :

le modèle sténopé est plus adapté lorsqu'on utilise des caméras ayant une focale f de

faibles dimension [6].

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \frac{f}{z} & 0 & 0 \\ 0 & \frac{f}{z} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1.4}$$

Ces coordonnées métriques du point P peuvent être ensuite transformées en coordonnées pixel (U,V) via la transformation suivante :

$$\begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & U_0 \\ 0 & -\alpha_v & V_0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{1.5}$$

- α_u est la taille des pixels selon les lignes.
- α_v est la taille des pixels selon les colonnes.
- U_0 et V_0 sont les coordonnées de la projection du centre optique C dans le plan image.

Les quatre paramètres ci-dessus représentent ce qu'on appel les paramètres intrinsèques de la caméra pour les déterminer il faut d'abord passer par le **calibrage** de la caméra.

1.3.2 Calibrage de la Camera

Pour pouvoir calibrer notre caméra on a utilisé l'outil *Camera Calibrator Toolbox* que fournit le logiciel *Matlab* qui nous permettra par la suite d'estimer les paramètres intrinsèques de la caméra. Afin d'effectuer cette tache on aura besoin d'imprimer un motif d'échiquier noir et blanc [1] (voir figure :1.10) ce dernier a une taille de 9 par 6 cases, chaque case de l'échiquier représente un carré dont le coté a une dimension de 0,025 mètres.

Pour avoir des résultats acceptables on prendra, via la caméra à calibrer, entre 10 et 20 captures de cette page imprimée, chaque capture doit être prise à partir d'une position et avec une direction différente d'une autre (voir figure :1.11).

L'étape suivante consiste à ouvrir l'outil du calibrage de *Matlab* en tapant `cameraCalibrator` dans le Workspace Matlab, on chargera par la suite les images en cliquant sur l'icône *Add Images* (fig :1.12)et sélectionnerons la taille réelle du côté

1. Le model d'échiquier est disponible sous format pdf sur le lien suivant : http://www.irisa.fr/lagadic/visp/img/OpenCV_Chessboard.pdf

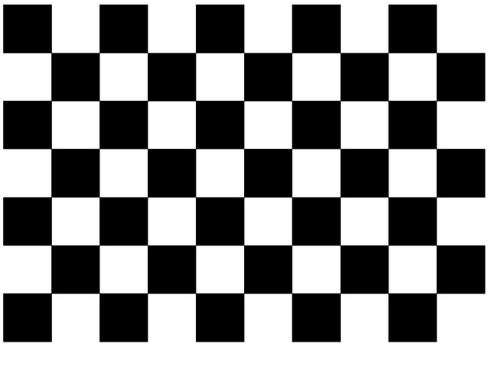


FIGURE 1.10: Motif d'échiquier utilisé pour le calibrage de la caméra.

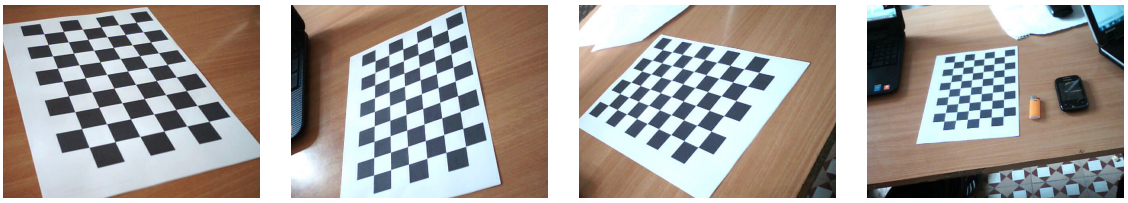


FIGURE 1.11: Exemples de captures utilisées pour le calibrage de la caméra.

d'un carré de l'échiquier (2.5cm dans notre cas), pour que le calibreur puisse rejeter les images dupliquées ou floues.

FIGURE 1.12: Étapes du calibrage de la Caméra.

Le volet d'image affiche les images avec des cercles verts indiquant les points détectés. Le carré indique la position de l'origine (0,0), x et y sont des flèches qui indiquent l'orientation des axes de l'échiquier. Une fois qu'on est satisfait avec les images acceptées, on doit cliquez sur le bouton Calibrer dans la barre d'outils pour pouvoir enfin récupérer les paramètres intrinsèques de la caméra en cliquant sur le bouton *Export Camera Parameters* sous forme d'une matrice de dimensions (3x3)1.6.

$$\begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix} \tag{1.6}$$

- c_x et c_y sont les coordonnées de la projection du centre optique C dans le plan image.
- $f_x \quad = F.s_x$ et $f_y = F.s_y$
- F est la distance focale de la caméra.
- s est un paramètre négligeable (s=0).
- s_x et s_y sont respectivement le nombre de pixels par millimètre dans la direction x et y.

Par identification on doit trouver que :

- $c_x = U_0$= 310.84 pixels.
- $c_y = V_0$= 233.62 pixels.
- F = f = 38.667 mm.

Remarque :
le calibrage de la caméra peut être effectué a travers d'autres méthodes en utilisant d'autres outils fournit par des logiciels et programmes :

- Par l'outil *camera calibration* du logiciel Cinéma 4D qui utilise une méthode très simple et pratique pour la détermination de la distance focale de la caméra.
- Par l'utilisation de l'un des bibliothèques OpenCV ou VISP qui proposent un programme en C++ qui peut être utilisé pour le calibrage de la caméra.

1.4 Conclusion

Dans ce chapitre, les différentes caractéristiques du robot mobile Robucar ont été présentées, elles incluent ses différents composants et moyens de perception utilisés pour la navigation de manière autonome. On a pu aussi déterminer le modèle cinématique de la plateforme mobile sous certains hypothèses.
Ainsi on a abordé et expliqué le principe de fonctionnement du capteur laser LMS511 et la méthode utilisée pour la récupération des données de ce dernier pour qu'il soit utilisées par la suite dans la partie programmation pour accomplir la tache de l'évitement des obstacles statiques.
Étant donné que la camera représente un outil de perception du robucar essentiel pour la récupération des informations visuelles de l'environnement extérieur, on a finalement

présenter le modèle *sténopé* de notre camera et montré comment la calibrer en utilisant l'outil *Camera Calibrator Toolbox de MATLAB*.

Chapitre 2

Initiation à ROS

Le domaine de la robotique a fait des progrès impressionnants au cours des dernières années. Des robots mobiles, aux quadrirotors, en passant par les robots manipulateurs, la technologie est plus développée que jamais. la communauté de la robotique a également développé des algorithmes qui aident ces robots à fonctionner avec un niveaux d'autonomie supérieur.
Malgré les progrès considérables dans ce domaine, il existe toujours des défis importants pour les développeurs de logiciels, dans ce chapitre on va faire la présentation d'une plate forme logiciel Robot Operating System (ROS), qui est destiné à atténuer certaines de ces difficultés, grâce aux outils et bibliothèques qu'il fournit pour l'obtention, la compilation, l'écriture et l'exécution de codes sur plusieurs ordinateurs.

2.1 Définition et présentation

Robot Operating System (ROS) est un nouvel outil (nouvelle structure de logiciel) développé récemment pour être utilisé dans le domaine de la robotique.

ROS fournit des bibliothèques et des outils pour aider les développeurs de logiciels à créer des applications robotiques. Il fournit une abstraction matérielle, des pilotes de périphériques, des bibliothèques, des visualiseur, un passage de messages, une gestion des paquets, et plus encore. ROS est sous licence open source, licence BSD. [7]

Le principe est de faire un morceau de logiciel qui pourrait travailler dans d'autres robots en faisant de petits changements dans le code. Ce que nous obtenons avec cette idée est de créer des fonctionnalités qui peuvent être partagés et utilisés dans d'autres robots sans trop d'efforts[10].

ROS a été initialement développé en 2007 par le Stanford Artificial Intelligence Laboratoire (SAIL) avec l'appui du projet de Robot Stanford AI. À partir de 2008, le développement se poursuit principalement à Willow Garage, un institut de recherche en robotique, avec plus de 20 institutions de collaborer au sein d'un modèle de développement fédéré[10].

ROS est un ensemble de bibliothèques logicielles et d'outils qui aident à construire des applications robotiques qui fonctionnent sur une grande variété de plates-formes robotiques Il ne supporte pas tout les langages de programmation mais seulement le c++, python et Lisp qui sont des langages orienté objet.

Après quelques années depuis son apparition, les sociétés ont commencées à adapter leurs produits pour être utilisées sous ROS. les images suivante montrent quelques plates-formes supportées par ROS.

Erle Copter

Pioneer LX-200h

turtlebot320

Nao robot

Les capteurs et les actionneurs utilisés en robotique ont également été adapté pour être utilisé avec ROS.

Oppeni Kinect

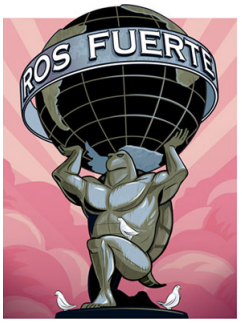
Sick Laser

VI Sensor

Depuis 2007, chaque année une nouvelle distribution ROS est développée.

22 Juillet 2014

04 Septembre 2013



23 Avril 2012

2.2 Installation

La démarche à suivre pour l'installation de ROS est bien expliquée sur le lien suivant : http://wiki.ros.org/fr/ROS/Installation

Remarques :

- Il n'est pas possible d'installer ROS pour tout les systèmes d'exploitations sauf si on veut l'utiliser de façon expérimentale (juste pour l'apprentissage), le seule environnement supporté par ROS est ubuntu de la distribution Linux qui est aussi un système d'exploitation **open source**
- la distribution ROS que nous allons utiliser dans notre projet est : **ROS Hydro** cette version peut être installée sous les trois version (12.04, 12.10, 13.04) d'Ubuntu.
- on peut aussi installer Ubuntu sur machine virtuelle sous Windows en utilisant le logiciel libre **virtual-Box**
- les instructions de l'installation si dessous sont obtenus depuis le site de la documentation officielle de ROS (lien de l'installation).

2.2.1 Installation de ROS Hydro sous Ubuntu

L'installation de ROS va être effectuée sur lignes de commandes, dans un nouveau terminal on va taper les commande suivantes :

```
sudo apt-get update
```

pour une installation **complète** :ROS, rqt, rviz, robot-generic librairies, les simulateurs 2D/3D , navigation et la perception 2D/3D.

```
sudo apt-get install ros-hydro-desktop-full
```

Pour une installation de bureau : ROS, rqt, rviz, et robot-generic librairies.

```
sudo apt-get install ros-hydro-desktop
```

Finalement pour effectuer une installation basique : seulement les paquets essentiels de ROS.

```
sudo apt-get install ros-hydro-ros-base
```

2.2.1.1 Installation des paquets manuellement

On peut également installer un package de ROS spécifique manuellement, la syntaxe est la suivante :

```
sudo apt-get install ros-hydro-PACKAGE
```

Exemple de l'installation du paquet VISP :

```
sudo apt-get install ros-hydro-visp
```

Pour trouver les paquets disponibles, on peut utiliser :

```
apt-cache search ros-hydro
```

2.2.1.2 Initialisation de Rosdep

Avant de pouvoir utiliser ROS, on aura besoin d'initialiser rosdep. rosdep permet d'installer facilement des dépendances du système pour les sources qu'on veut compiler et il est nécessaire pour faire fonctionner certains composants de base dans ROS.

```
sudo rosdep init
rosdep update
```

2.2.1.3 Configuration de l'environnement

Cette étape est très utile lors de l'utilisation des commandes que fournit ROS, surtout lors de la compilation ou du lancement d'un certain nœud par la commande **rosrun**. Son utilité est qu'à chaque fois qu'un nouveau shell est lancé, les variables d'environnement ROS sont automatiquement ajoutés à notre session de bash.

```
echo "source /opt/ros/hydro/setup.bash" >>  /.bashrc
source  /.bashrc
```

Mais si on veut juste changer l'environnement de votre shell courant, on peut taper à chaque fois :

```
source /opt/ros/hydro/setup.bash
```

2.2.1.4 La commande rosinstall

rosinstall est un outil de ligne de commande fréquemment utilisés dans ROS, et qui est distribué séparément. Il permet de télécharger facilement de nombreuses sources pour les paquets de ROS avec une seule commande.

Pour l'installer il suffit d'exécuter :

```
sudo apt-get install python-rosinstall
```

2.3 Architecture de ROS

2.3.1 Comprendre le system de fichiers ROS

Un ensemble de concepts sont utilisés pour expliquer comment ROS est organisé à l'intérieur, il faut comprendre la structure des dossiers et des fichiers minimes dont il a besoin pour travailler. Similaire à un système d'exploitation, un programme ROS est divisée en des dossiers qui contiennent des fichiers qui décrivent leurs fonctionnalités, parmi ces dossiers les plus important à citer sont :

- **packages(paquets)** Ils contiennent le minimum pour créer un programme sous ROS.
- **stacks** Il est formé par un ensemble de paquets (packages).
- **messages** C'est l'information que le processus envoie à d'autres processus.
- **services** C'est une structure de données sous forme de demande-réponse.

2.3.1.1 Packages

Habituellement, lorsque nous parlons de packages, nous nous référons à une structure typique de fichiers et de dossiers. Cette structure se présente comme suit :

- **`bin/` :** C'est le dossier où nos programmes compilés et liés sont stockées.
- **`include/package_name/` :** Ce répertoire comprend les en-têtes des bibliothèques qui seront utilisés ; il faut pas oublier d'exporter les manifeste, car ils sont destiné à être fourni à d'autres paquets.
- **`msg/` :** Si on développe des messages non standard, on doit les mettre ici.
- **`src/` :** C'est le répertoire où les fichiers source des programmes sont présents.
- **`srv/` :** Représente les services de type (SRV).
- **`CMakeLists.txt` :** C'est le fichier source qui contient tous les paramètres du projet et la démarche à suivre pour le construire.
- **`manifest.xml` :** Ceci est le fichier qui fournit des informations sur un package (paquet).

Si on veut créer, modifier ou manipuler des packages (paquets), il existe un ensemble de commandes qui sont fournit par ROS qui vont nous simplifier cette tache :

rospack : cette commande est utilisée pour trouver un package ou obtenir des informations sur celui-ci dans le system.

- **`roscreate-pkg` :** cette commande permet de créer un nouveau package (paquet).
- **`rosmake` :** elle est utilisée pour compiler un package.
- **`rosdep` :** Cette commande installe les dépendances du système d'un package .
- **`rxdeps` :** utilisée pour savoir les dépendances d'un package sous forme d'un graphe.

ROS nous fournit un autre package très pratique appelé 'rosbash', et qui a pour objectif de simplifier aux utilisateurs de se déplacer entre les dossiers et fichiers d'un package.

- **`roscd` :** cette commande permet de changer le répertoire courant.
- **`rosed` :** celle-ci est utilisée pour modifier un fichier.
- **`roscp` :** Cette commande est utilisée pour copier un fichier d'un paquet.
- **`rosd` :** pour lister les répertoires d'un paquet.
- **`rosls` :** pour lsiter les fichiers d'un paquet.

2.3.1.2 Stacks

Les paquets de ROS sont organisées en stacks,ainsi plusieurs paquets forment ce qu'on appel un stack.bien que l'objectif de paquets est de créer des collections minimales de code pour une réutilisation facile,un stack a besoin d'une structure de base des fichiers et dossiers. pour pouvoir créer un stack ROS nous fournit la commande roscreate-stack pour cela trois fichiers sont nécessaires :

- `CMakeList.txt`.
- `Makefile`.
- `stack.xml`.

Si on voit `stack.xml` dans un dossier, on peut être sûr que c'est bien un stack.

Remarque :
Un stack est constitué de plusieurs packages, chaque package est constitué de plusieurs nœuds (exécutables), et plusieurs stacks constituent ce qu'on appel repository (figure : 2.1).

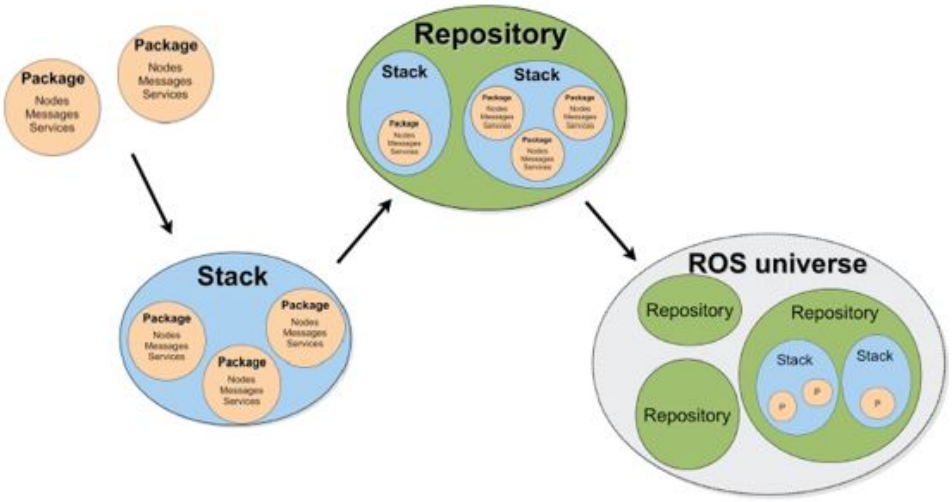


FIGURE 2.1: Les Packages Stacks et Repository

2.3.1.3 Messages

Dans un paquet il existe un ou plusieurs nœuds chaque nœud représente un exécutable, les nœuds inter-communiquent entre eux par l'intermédiaire des messages et services, c'est l'un des avantages de ROS.

ROS à beaucoup de messages prédéfinis, mais on peut toujours développer un nouveau message, ce message sera placé dans le répertoire msg/ de notre paquet et il sera enregistré dans un fichier avec l'extension .msg, c'est l'extension qui définit les messages.
Un message doit contenir les données qui doivent être transmises pour effectuer la communication entre les différents nœuds, ces données sont sous la forme de variables typées déclarées dans le fichier .msg comme le montre l'exemple suivant (message.msg) :

```
float32 vitesse
int64 nombreDeVoitures
bool marche
```

La liste des différents types qui existent et qui peuvent être utilisés dans les fichiers .msg sont résumés dans le tableau de la (figure : 2.2).

2.3.1.4 Services

Comme nous l'avons cités précédemment les nœuds communiquent entre eux par l'intermédiaire des messages et services. La seule différence entre les messages et services est que les messages sont envoyés d'un nœud vers un autre dans un seul sens (figure : 2.5) contrairement aux services qui ont la même structure que les messages sauf que la communication entre deux nœuds via des services se fait dans les deux sens (figure : 2.4).

L'extension d'un fichier service est .srv ce fichier doit être stocké dans le répertoire /srv dans un paquet.
La commande `rosserv` que fournit ROS imprime sur l'écran la description du fichier service et le paquet qui contient le fichier .srv.

Primitive type	Serialization	C++	Python
bool	Unsigned 8-bit int	uint8_t	bool
int8	Signed 8-bit int	int8_t	int
uint8	Unsigned 8-bit int	uint8_t	int
int16	Signed 16-bit int	int16_t	int
uint16	Unsigned 16-bit int	uint16_t	int
int32	Signed 32-bit int	int32_t	int
uint32	Unsigned 32-bit int	uint32_t	int
int64	Signed 64-bit int	int64_t	long
uint64	Unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ASCII string (4-bit)	std::string	string
time	Secs/nsecs signed 32-bit ints	ros::Time	rospy.Time
duration	Secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

FIGURE 2.2: Les différents types pour un message [10].

2.3.2 Comment se fait le calcul sous ROS

ROS crée un réseau où tous les processus sont connectés, tout nœud (exécutable dans un paquet) dans le système peut accéder à ce réseau, interagir avec d'autres nœuds et transmettre des données vers ce réseau.
Les éléments de ce réseau sont illustrés dans la figure(figure : 2.3)

Les concepts de base de ce réseau sont :

- **Nodes** (noeuds)
- **Master**
- **Parametre server**
- **Messages**
- **Topics**
- **Services**
- **Bags**

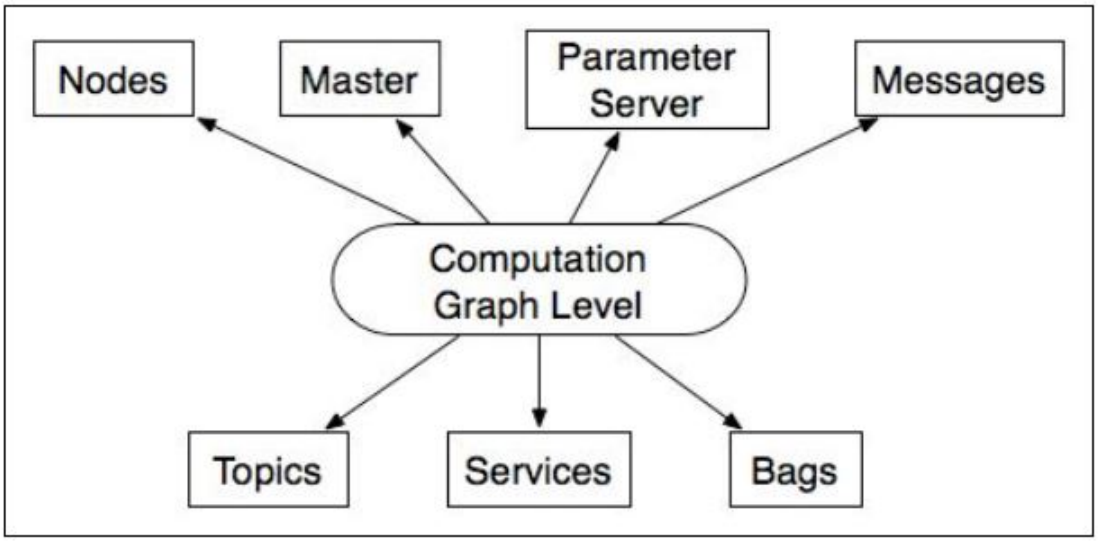


FIGURE 2.3: Réseau de connexion des processus sous ROS [10]

2.3.2.1 Nodes (les Nœuds)

Comme nous l'avons cités précédemment les nœuds sont des exécutables qui sont capable de communiquer entre eux via des messages ou des services, chaque nœud est crée pour accomplir une tache spécifique autrement dit ce sont des processus qui permettent d'effectuer des calculs sous ROS.
Un nœud doit avoir un nom unique dans un système pour éviter toute ambiguïtés lors de la communication, un nœud peut être écrit en utilisant des des bibliothèques différentes comme `roscpp` (pour un programme en c++) et `rospy` (pour un programme en python).
ROS dispose d'un certain nombre d'outils pour gérer les nœuds, la commande `rosnode` nous permet d'afficher des informations sur les nœuds, tels que la liste des nœuds en cours d'exécution.

Les commandes prises en charge sont les suivantes :

- **`rosnode info node :`** Affiche les informations concernant le nœud en question.
- **`rosnode kill node :`** Ferme un nœud en cours d'exécution.
- **`rosnode list :`** Affiche la liste des nœuds actives.
- **`rosnode machine hostname :`** Affiche la liste des nœuds actives dans une machine particulière.
- **`rosnode ping node :`** Teste la connectivité entre les nœuds.

2.3.2.2 les Topics

Chaque message doit avoir un nom pour être acheminé par le réseau ROS. Quand un nœud envoie des données, nous disons que le nœud publie un topic. Les nœuds peuvent aussi recevoir des topics des autres nœuds.
Il est important que le nom du topic soit unique pour éviter tout problème de confusion. Les topics sont les bus utilisés par les nœuds pour transmettre des données. La transmission peut se faire sans une connexion directe entre les nœuds. Un nœud peut publier un topic seulement s'il a le même type de message.

ROS dispose d'un outil pour travailler avec les topics appelés rostopic, c'est un outil de ligne de commande qui nous permet d'avoir des informations sur le topic ou de pouvoir publier des données directement sur le réseau.

- **`rostopic echo /topic` :** pour afficher les messages du topic a l'écran.
- **`rostopic find message_type` :** cette commande permet de chercher les topics qui ont un certain type.
- **`rostopic hz /topic` :** elle permet d'afficher la fréquence d'un topic.
- **`rostopic info /topic` :** afficher des informations concernant un topic active.
- **`rostopic list` :** afficher des informations concernant les topics actives.
- **`rostopic pub /topic type args` :** pour publier des données vers un topic de notre choix.
- **`rostopic type /topic` :** elle nous permet d'afficher le type d'un topic.

2.3.2.3 les services

Lorsqu'on a besoin de communiquer avec des nœuds et de recevoir une réponse on doit utiliser des services (figure : 2.4).

Les services sont développés par l'utilisateur. ROS dispose de deux outils de ligne de commande pour travailler avec les services, `rossrv` et `rosservice`. `Rossrv`, nous permet de voir des informations sur la structure de données des services. `Rosservice`, nous permet d'énumérer les services de la requête.

Les commandes prises en charge sont comme suit :

- **`rosservice find msg-type` :** chercher des services par leurs types.

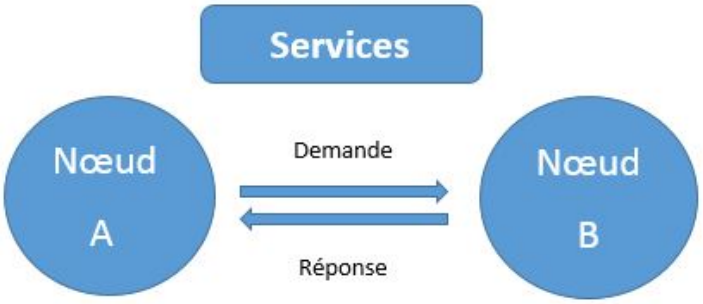


FIGURE 2.4: Communication des Nœuds via des services

- **`rosservice info /service :`** afficher des informations concernant un service.
- **`rosservice list :`** afficher une liste des services actives.
- **`rosservice type /service :`** afficher le type d'un service.

2.3.2.4 les messages

Un nœud envoie des informations à un autre nœud en utilisant des messages qui sont publiés par des topics (figure : 2.5). Le message a une structure simple qui utilise des types standards imposés par l'utilisateur.

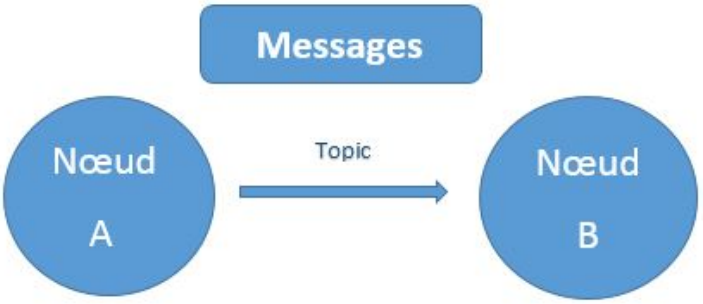


FIGURE 2.5: Communication des Nœuds via des messages

ROS a un outil de ligne de commande pour obtenir des informations sur les messages appelé rosmsg.

voiçi quelques commandes prises en charge :

- **`rosmsg show` :** afficher les champs d'un message.
- **`rosmsg list` :** afficher la liste de tout les messages.
- **`rosmsg package` :** afficher la liste de tout les messages dans un paquet.
- **`rosmsg packages` :** afficher la liste de tout les paquets qui contiennent le message.
- **`rosmsg users` :** rechercher les fichiers de code qui utilisent le type de message.

2.3.2.5 les Bags

Un bag est un fichier créé par ROS et qui a une extension .bag, il est destiné pour enregistrer toutes les informations (les messages, les topics, les services, ...) pour pouvoir ensuite utiliser ces données ultérieurement pour visualiser ce qui s'est passé pendant l'exécution d'un certain processus sous ROS, on peut par la suite visualiser les donnés enregistrées sous le fichier bag, qui peut être reproduit en ROS comme une vraie session. Nous utilisons cette fonctionnalité pour nous aider à déboguer nos algorithmes.

Pour utiliser les fichiers de bag, ROS nous fournit les outils suivants :

- **`rosbag` :** Cette commande est utilisé pour enregistrer, lire et effectuer d'autres opérations.
- **`rxbag` :** Utilisée pour visualiser les données dans un environnement graphique.

2.3.2.6 ROS Master

ROS Master est un nœud centrale qui permet au nœuds de communiquer entre eux (fig :2.6).

Remarque :
pour pouvoir démarrer une session ROS, on doit d'abord lancer la commande.

```
roscore
```

dans un nouveau terminal, c'est la première étape qu'on doit effectuer lors de l'utilisation de ROS (figure : 2.7). Cette commande se charge d'exécuter les composantes essentielles pour démarrer ROS :

— **ROS Master**.

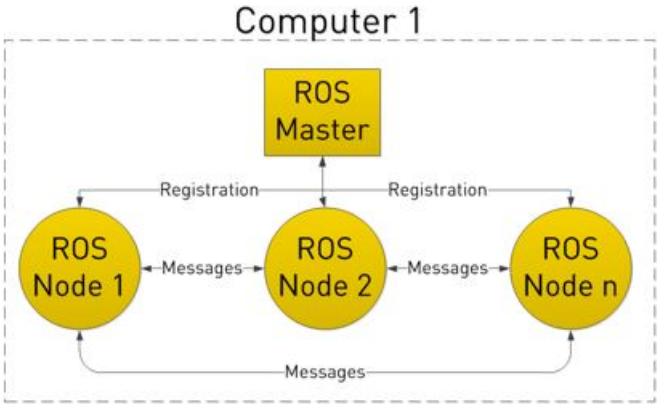


FIGURE 2.6: ROS Master assure la communication des nœuds

— **ROS parametre server**.
— **ROS logging node**.

cette figure montre le résultat d'exécution de cette commande dans un nouveau terminal :

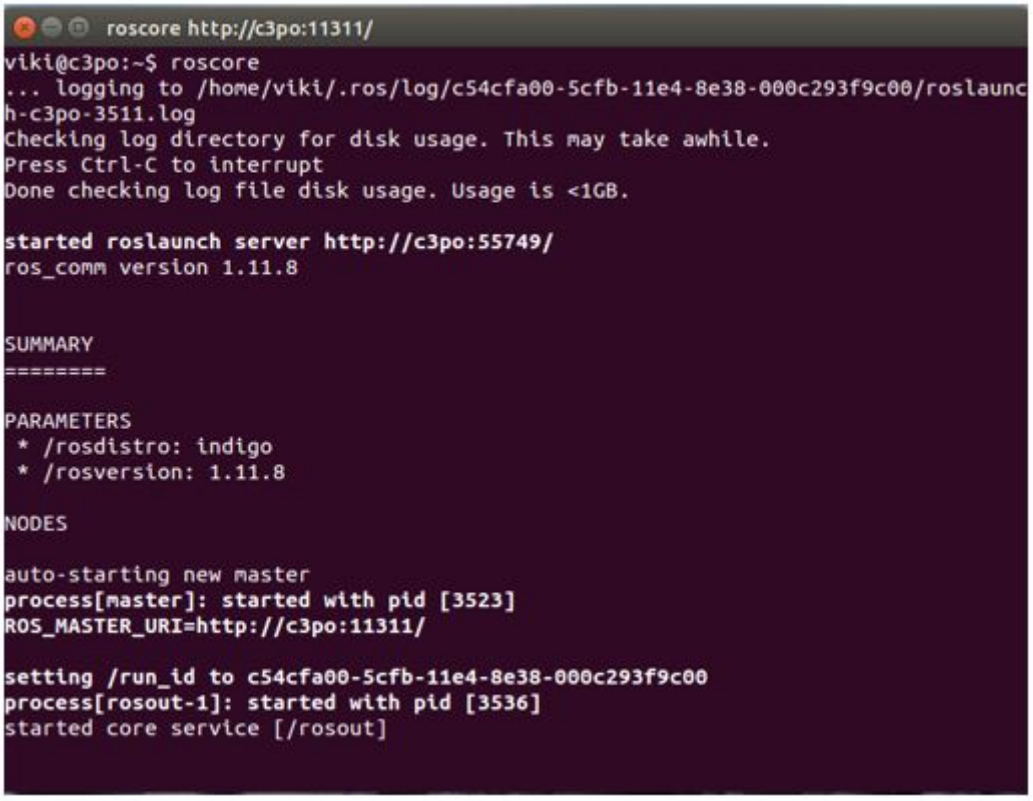


FIGURE 2.7: Résultat d'exécution de roscore

Chapitre 3

Asservissement visuelle

L'utilisation de l'asservissement visuel en robotique mobile connaît depuis quelques années un essor important, notamment pour les applications des véhicules autonomes, la difficulté essentielle réside dans l'élaboration des lois de commande prenant en compte les contraintes non holonomes de ce type de robots [2]. L'asservissement visuel consiste à contrôler les mouvements d'un système robotique en utilisant des informations visuelles, notées s, issues d'une ou plusieurs caméras

Les différentes techniques d'asservissement visuel sont classées en fonction des mesures utilisées en entrée du module de commande. Parmi les techniques les plus connues on distingue :

- L'asservissement visuel 3D ou *Position Based Visual servoing* (PBVS).
- L'asservissement visuel 2D ou *Image Based Visuel servoing*(IBVS).

3.1 L'asservissement visuel 3D

L'asservissement visuel 3D utilise en entrée de la boucle de commande des informations tridimensionnelles exprimées dans un repère euclidien, à savoir la position r de la caméra par rapport à l'objet d'intérêt, la boucle de commande consiste alors à estimer à chaque itération la position r de la caméra par rapport à cet objet, pour atteindre une position de référence r^* souhaité (voir Fig 3.1), à partir des informations visuelles issues de l'image.

Pour l'estimation de la position de la caméra par rapport à l'objet, plusieurs méthodes existent. Ces méthodes reposent généralement sur la connaissance à priori

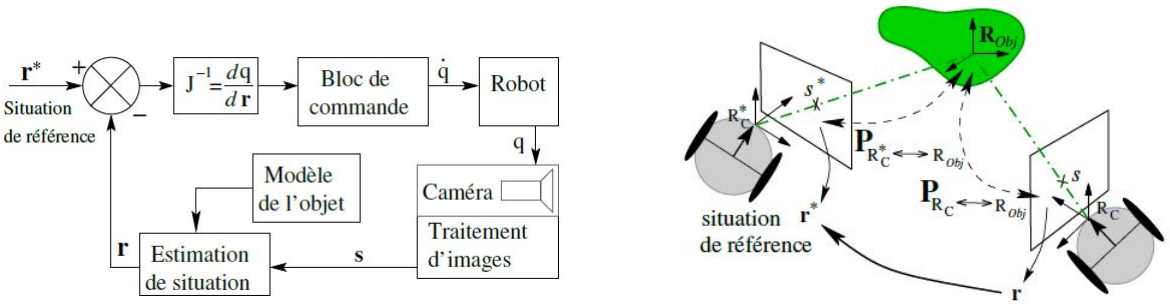


FIGURE 3.1: Principe de l'asservissement 3D [6]

d'un modèle 3D de l'objet et des paramètres intrinsèques de la caméra issues du calibrage de cette dernière. Malgré la simplicité de la définition des lois de commandes, l'asservissement visuel 3D présente un inconvénient majeur causé par les erreurs importantes introduites d'une part, par les paramètres intrinsèques de la caméra et d'autre part par le modèle géométrique du robot.

3.2 L'asservissement visuel 2D

Contrairement à l'asservissement visuel 3D, les techniques de l'asservissement visuel 2D consiste à utiliser directement les informations visuels s extraites de l'image sans passer par la phase d'estimation de r, la commande consiste alors à contrôler le mouvement de la caméra afin que les mesures dans l'image $s(t)$ atteignent une valeur désiré s^* ou suivent une trajectoire spécifié $s^*(t)$ (voir Fig 3.2).

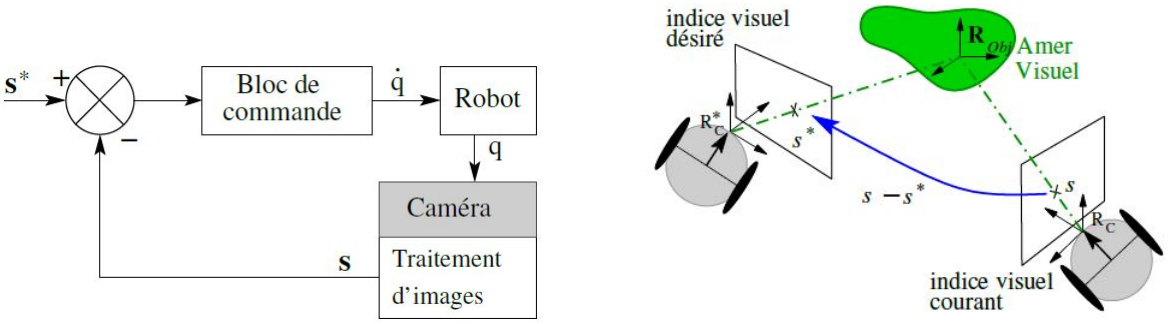


FIGURE 3.2: Principe de l'asservissement 2D [6]

Cependant pour l'élaboration des diverses lois de commande pour un asservissement visuel 2D, l'obtention de la relation liant les informations visuels s au mouvement de la caméra est une action incontournable. Cette relation est fondamentale et elle est obtenue par la dérivation des informations sensoriels s par rapport à la situation r de la caméra,

elle est définit par une matrice appelée *Jacobien de l'image* ou *Matrice d'interaction* notée L (éq 3.1).

$$L = \frac{\partial s}{\partial r} \tag{3.1}$$

Les lois d'asservissements visuels 2D sont, d'une manière générale, des lois de commandes relativement rapides à calculer, puisqu'il n'y a pas de phase de reconstruction 3D. Cette simplicité représente un gain considérable au niveau du calcul de la commande, ce qui contribue à la stabilité du système à contrôler.

Il existe une approche qui permet de synthétiser des lois de commande utilisant les données visuelles, cette approche consiste à exploiter le formalisme des fonctions de tâches. Ce formalisme permet de modéliser la tâche considérée sous la forme d'une fonction, puis de synthétiser (voir Annexe) un asservissement permettant de la réguler à zéro.

3.3 Interaction caméra environnement

Comme il mentionné ci dessus, dans le cadre de l'asservissement visuel 2D, le choix des informations visuelles $s(q,t)$ qui caractérisent les mesures effectuées au moyen d'une caméra, et l'obtention de la relation caractérisant leur variation sont deux points fondamentaux de cette approche. Il est nécessaire d'établir une relation connue sous le nom de la matrice d'interaction qui permet de relier le mouvement de la caméra aux informations visuelles. En considérant que seul le mouvement de la caméra, et éventuellement celui de la cible, sont susceptibles de faire varier la valeur du signal sensoriel et en différentiant les informations visuelles $s(q,t)$ par rapport au temps, il est possible d'exprimer la variation des informations visuelles en fonction des mouvements de la caméra (éq :3.2)[6].

$$\dot{s}(q,t) = \frac{\partial s}{\partial q}\frac{dq}{dt} + \frac{\partial s}{\partial t} = \frac{\partial s}{\partial r}\frac{\partial r}{\partial q}\frac{dq}{dt} + \frac{\partial s}{\partial t} \tag{3.2}$$

Où :

- q est le vecteur de configuration du robot.
- $\frac{\partial s}{\partial r}$ est la matrice d'interaction ou jacobien de l'image notée L(s,z).
- $J(q) = \frac{\partial r}{\partial q}$ le jacobien du robot, il ne dépend que de la géométrie du robot.
- $\dot{q} = \frac{dq}{dt}$ le vecteur de commande du system robotique.
- $\frac{ds}{dt}$ est un terme correspondant au mouvement propre de l'objet par rapport à la caméra.

On remarque que la matrice d'interaction dépend de la nature des informations visuelles choisies, de la situation de la caméra par rapport à l'objet observé et plus particulièrement de la profondeur z qui représente la distance entre la caméra et un point P(x,y,z) de l'espace dont les informations sensorielles s(q,t).

La matrice d'interaction se détermine selon le type de l'information visuelle choisit, des méthodes sont proposées pour calcul analytique de la matrice d'interaction pour différentes primitives géométriques simples (des points, des lignes, sphères, ellipses...). Dans le cas d'une primitive de type droite la matrice d'interaction (éq 3.3) est donné sous la forme suivante : [4]

$$\begin{bmatrix} -\lambda_\theta sin\theta & \lambda_\theta cos\theta & -\lambda_\theta \rho & \rho sin\theta & -\rho cos\theta & -1 \\ -\lambda_\rho sin\theta & \lambda_\rho cos\theta & \lambda_\rho \rho & (1+\rho^2)cos\theta & (1+\rho^2)sin\theta & 0 \end{bmatrix} \tag{3.3}$$

Avec :

$$\begin{cases} \lambda_\theta & = & \frac{1}{d_i}.(a_i.cos\theta & + b_i.sin\theta) \\ \lambda_\rho & = & \frac{1}{d_i}.(a_i.\rho.cos\theta & + b_i.\rho.sin\theta + c_i) \end{cases} \tag{3.4}$$

Dans le cas où une droite est représentée par l'intersection de deux plans d'équations :

$$\begin{cases} a_1X & + & b_1X & + & c_1.Z & + & d_1 & = & 0 \\ a_2X & + & b_2X & + & c_2.Z & + & d_2 & = & 0 \end{cases} \tag{3.5}$$

Où (ρ, θ) sont les coordonnées polaires représentant une droite dans le plan de l'image.

Le calcul de la matrice d'interaction L(s,z) nécessite généralement une mesure ou un modèle de la profondeur z. Toutefois, cette information n'étant pas toujours disponible, il est nécessaire, dans ce cas, d'utiliser un modèle ou une approximation de la matrice d'interaction, notée $\hat{L}$.[6] où cette dernière représente la pseudo inverse de la matrice d'interaction. Cette approximation pourrait en effet fausser le calcule de la commande, entrainer des oscillations et réduire la vitesse optimale du suivie.

- **Énoncé du problème dû à la déformation de la caméra**

La déformation de la camera peut être justifiée par son modèle qui a été définit précédemment nous pouvons le remarquer à travers cet exemple :

Comme il est représenté dans la figure (3.3), la ligne détectée est parallèle au robucar alors que sur l'image on voit bien que la ligne détectée est fortement incliner, cette même image pourrait être perçu dans le cas où la trajectoire est belle et bien inclinée par rapport au robot ceci pourrait poser des problèmes au niveau de la commande.

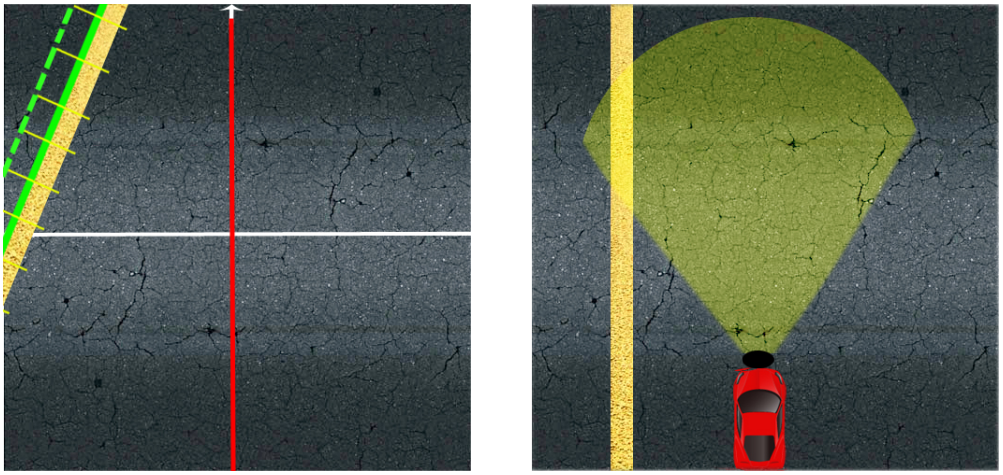


FIGURE 3.3: Problème due à la déformation de la caméra.

Pour remédier à ce problème (ou du moins le rendre moins fréquent) il faut que la camera soit fortement incliné vers le bas, cela permet de faire tendre le plan image (moyennant une transformation pixel-mètre) vers le plan de la scène(OXY) sur lequel est tracé la trajectoire, cependant une faible inclinaison de la camera par rapport à l'horizontale permet une meilleur visibilité de la trajectoire et permet donc de réduire les oscillations du robot après l'implémentation de la commande (chapitre 4), un compromis s'impose pour le choix de l'angle d'inclinaison de la camera.

Tenant compte de ces problèmes nous avons décidé de nous orienter vers la logique floue, afin mener à bien le calcule de la commande, qui illustre ces situations problématiques rencontrés sous forme de règles dont chacune apporte part au calcule de la commande finale à appliquer au système.

Pour cela nous allons commencer l'étape de détection de trajectoire et extraction des données visuelles.

3.4 Détection de trajectoire

Dans notre cas une trajectoire est définit par une droite, des segments de droites ou par une ligne courbée qui sera assimilée par des segments de droite formés par la différence d'intensités de couleurs de deux plans par exemple : marquage routier.

Pour faciliter la détection visuelle de la trajectoire nous nous somme orienté vers une bibliothèque *VISP* [[11]] qui présente des fonctions qui peuvent être utilisées pour la détection de contour (bord) de la trajectoire tracée.

La première tache du programme est de convertir une image en couleurs en niveau de gris, sachant que dans une image numérique, le niveau de gris représente la luminosité d'un pixel, lorsque les valeurs de ses composantes de couleur sont identiques. Pour convertir une image couleur en niveau de gris il faut remplacer, pour chaque pixel les trois valeurs représentant les niveaux de rouge, de vert et de bleu, en une seule valeur représentant la luminosité.

Après l'affichage de l'image en niveau de gris nous sélectionnerons avec le curseur un premier point présent sur le bord de la trajectoire dont les coordonnées dans l'image seront reconnues en utilisant la fonction appartenant à la classe *vpDisplay* de la bibliothèque *VISP* nous choisirons également un deuxième point pour donner une information sur l'orientation initiale de la droite à détectée.

En réalité le point sélectionné sera représenté par un carré qui le contient dont les dimensions sont (25x25) Pixel, ce « point » a donc des propriétés concernant la variation du niveau de gris (à l'intérieur du carré).

De la position précédente (celle du premier point sélectionné), le programme détecte la position du prochain point, pressentant une similitude avec le premier point, le long de la direction du contour (et donc celle du deuxième point sélectionné). La détection se fait comme suit : Pour chaque pixel le long de cette direction, il y a calcule du produit de convolution entre le signale représentant la variation du niveau de gris (luminosité) du pixel précédent et celui du pixel suivant, sachant que plus les deux signaux se ressemblent plus le produit de convolution est important le pixel qui sera sélectionné par l'algorithme sera celui qui aura une convolution supérieur à 15000.

Parmi les points détectés l'algorithme gardera ceux qui sont alignés, pour en tirer l'équation de la droite correspondante au segment de droite qui contient les points détectes.

Ce qui fait que même si la trajectoire est courbé (n'est pas constituée par des segments de droite) ce programme arrive à détecter deux points, appartenant à cette trajectoire courbé, représentants les mêmes propriétés de variation du niveau de gris et d'assimiler la trajectoire les séparant par un segment de droite (voir fig 3.4).

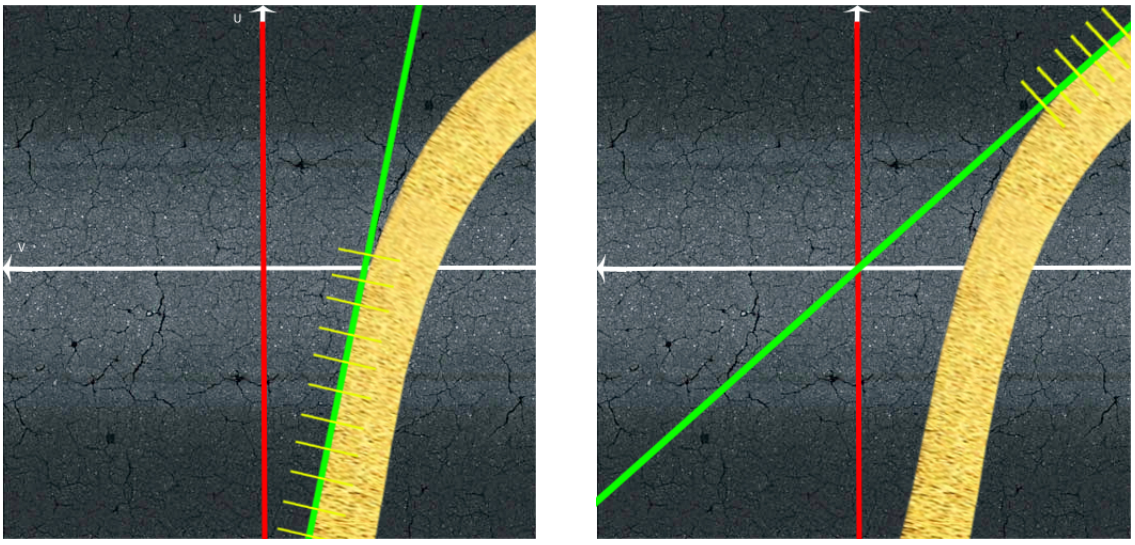


FIGURE 3.4: Détection d'une partie droite et d'une partie courbée d'une trajectoire.

La droite sera définie à l'aide des deux paramètres ρ et θ par l'équation suivante (voir fig 3.5). :

$$i.cos\theta + j.sin\theta + \rho = 0 \tag{3.6}$$

Où :

- θ représente l'angle entre l'axe vertical limitant l'image par la gauche et la normale de droite détectée.
- ρ représente la distance en pixel entre le coin supérieur gauche de l'image et la droite détectée.

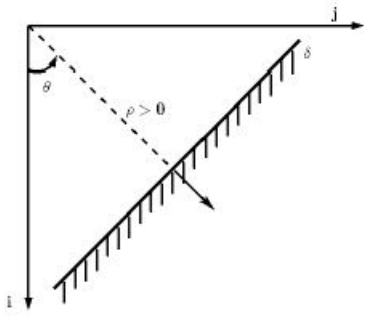


FIGURE 3.5: Représentation des paramètres retournés par la droite détectée [11].

Le raisonnement avec les variables (ρ,θ) retournées par le programme de détection de ligne étant fastidieux nous avons préféré effectuer un changement de repère et de variables (α,β) avec :

- β représente « la distance » en Pixels entre l'origine et l'intersection de la droite détectée avec l'axe (OV) (voir fig 3.6).
- α représente la tangente de la droite par rapport à l'axe (OU) et non pas par rapport à (OV) pour que la tangente de la ligne désiré (en rouge) soit égale à zero, d'où erreur(alpha) = alpha, et non pas à l'infini (ce qui poserait un grand problème pour le calcule d'erreur).

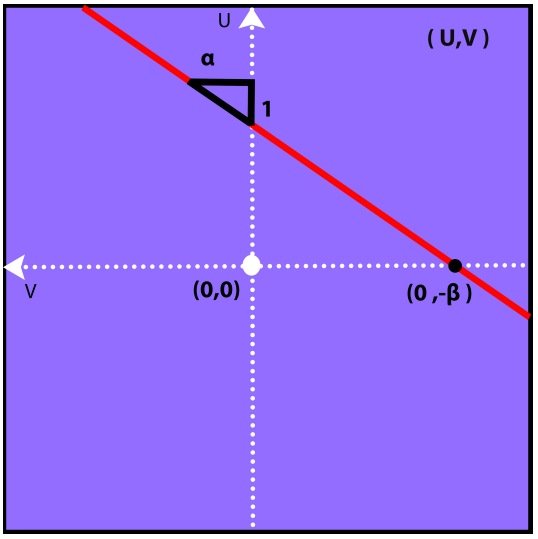


FIGURE 3.6: Représentation des nouveaux paramètres de la droite détectée.

Développement de la relation entre (ρ, θ) et (α, β) :

$$i = -j.\frac{sin\theta}{cos\theta} + \frac{\rho}{cos\theta} \tag{3.7}$$

$$i = -j.tan\theta - \frac{\rho}{cos\theta} \tag{3.8}$$

$$U = -(V + \frac{l_r}{2}).tan\theta - \frac{\rho}{cos\theta} + \frac{l_g}{2} \tag{3.9}$$

$$U = -V.tan\theta + tan(\frac{l_r}{2}) - \frac{\rho}{cos\theta} + \frac{l_g}{2} \tag{3.10}$$

On pose

$$\begin{cases} A = tan\theta \\ B = \frac{l_r}{2}tan\theta - \frac{\rho}{cos\theta} + \frac{l_g}{2} \end{cases} \tag{3.11}$$

d'où

$$U = -A.V + B \tag{3.12}$$

$$V = -\frac{1}{A}.V + \frac{B}{A} \tag{3.13}$$

Par identification :

$$\begin{cases} \alpha = -\frac{1}{A} \\ \beta = \frac{B}{A} \end{cases} \tag{3.14}$$

D'où :

$$\begin{cases} \alpha = -tan^{-1}\theta \\ \beta = \frac{l_r}{2} - \frac{\rho}{sin\theta} + \frac{l_g}{2.tan\theta} \end{cases} \tag{3.15}$$

Où l_g et l_r représentent respectivement la longueur et la largeur en pixels de l'image.

Nous avons utilisé une fonction en c++ qui permet lancer un enregistrement vocale que nous enregistré et qui se lance pour avertir le conducteur au moment où la ligne détectée serais perdue.

Une multitudes de testes nous ont confirmé la robustesse du programme de détection et nous ont donc permis d'entamer la partie *suivie de trajectoire* avec la commande par logique floue référencée vision.

Chapitre 4

Application de la Logique Floue pour le suivie de ligne

Au départ théorie, la logique floue s'affirme comme une technique opérationnelle. Utilisée à côté d'autres techniques de contrôle avancé, elle fait une entrée discrète mais appréciée dans les automatismes de contrôle industriel. La logique floue ne remplace pas nécessairement les systèmes de régulation conventionnels ; elle est complémentaire. Ses avantages viennent notamment de ses capacités à :

- Formaliser et simuler l'expertise d'un opérateur ou d'un concepteur dans la conduite et le réglage d'un procédé,
- Donner une réponse simple pour les procédés dont la modélisation est difficile.
- Prendre en compte sans discontinuité des cas ou exceptions de natures différentes, et les intégrer au fur et à mesure dans l'expertise.
- Prendre en compte plusieurs variables et effectuer de la « fusion pondérée » des grandeurs d'influence.

4.1 Intérêt et utilisation de la logique floue pour le contrôle

L'être humain résout souvent des problèmes complexes à l'aide de données approximatives la précision des données est, donc, souvent inutile. Ce qui fait que plutôt que de modéliser le système, il est souvent intéressant de modéliser le comportement d'un opérateur humain face au système et plutôt que par des valeurs numériques précises, le fonctionnement doit être décrit par des qualificatifs globaux traduisant l'état approximatif des variables.

Un traitement flou se fait suivant le schémas suivant :

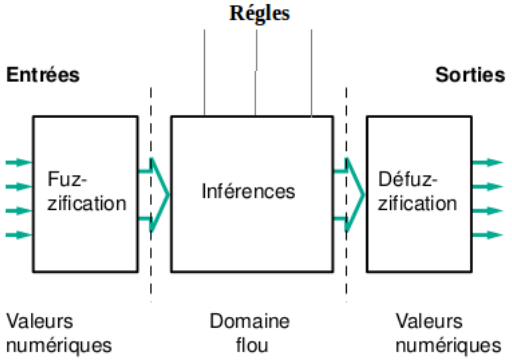


FIGURE 4.1: Régulateur floue.

les grandeurs numériques d'entrées que nous avons utilisé dans notre cas sont alpha, beta, dalpha et dbeta où :

- alpha et beta nous permettent de savoir la position actuelle de la ligne détectée.
- dalpha et dbeta sont respectivement les variations de alpha et de beta, elles nous permettent d'avoir une estimation sur la position future de la ligne détecté et de pouvoir ainsi anticiper et éviter qu'elle s'éloigne de la position désirée (voir fig4.2).

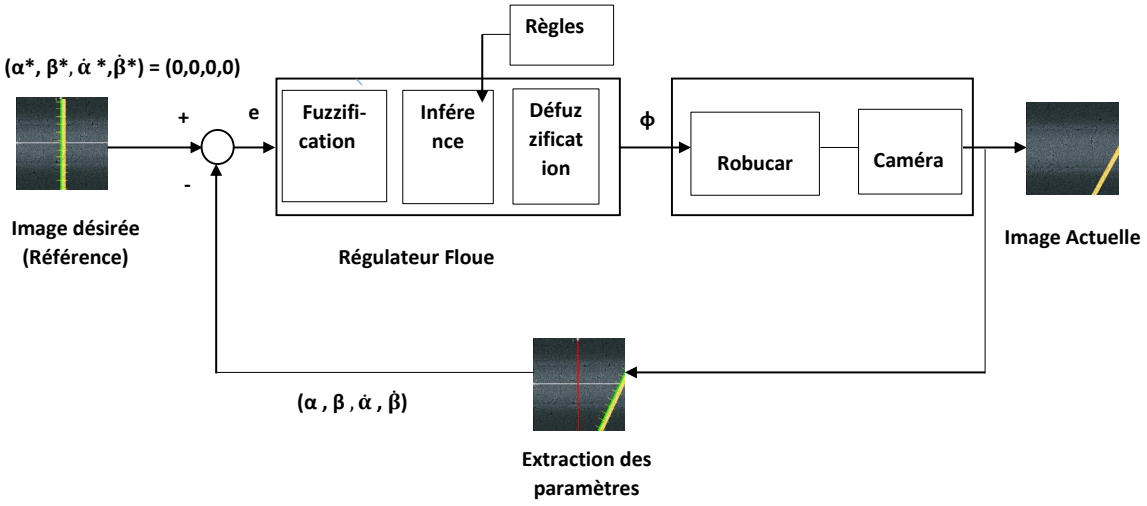
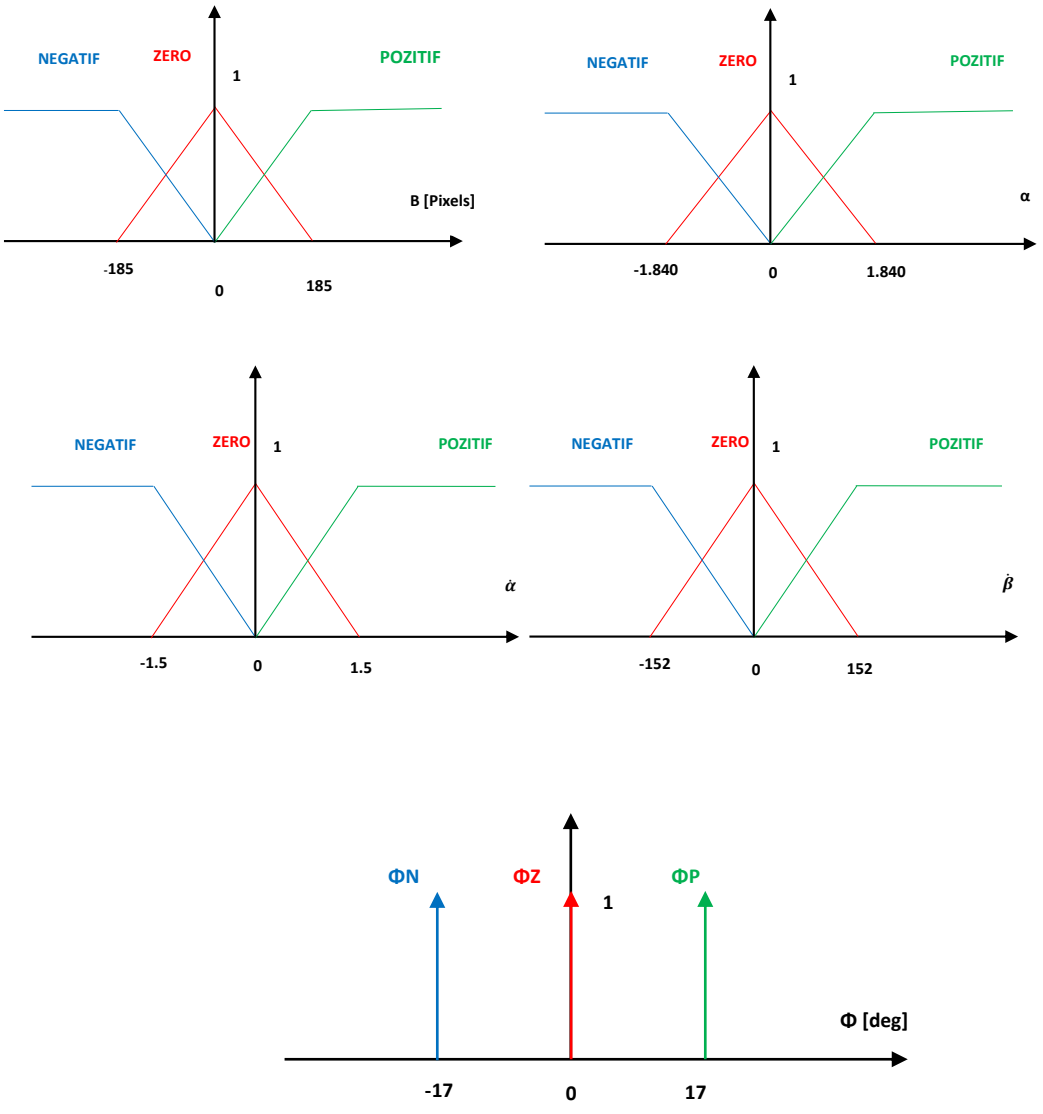


FIGURE 4.2: Schéma Fonctionnel du suivie de linge.

4.2 Fuzzification

La fuzzification consiste à évaluer les fonctions d'appartenance qui vont nous permettre d'établir une relation entre la grandeur d'entrée et le degré de vérité de la variable floue correspondante ceci est illustré par les graphes des fonctions d'appartenances.

Les fonctions d'appartenances choisit pour le suivie de ligne sont les suivants :



où $\phi = 17deg$ représente l'angle de braquage maximal.

Les fonctions d'appartenances ont été choisies triangulaires et symétriques pour simplifier le calcule de la commande et ainsi permettre de réduire le temps d'exécution du programme mais ceci a été fait sous les hypothèses suivantes :

- les quartes roues du robucar ont le même rayon (parfaitement identiques).
- la vitesses appliquer aux deux roues avant est exactement la même.
- l'angle maximum ainsi que la vitesse angulaire de braquage des deux roues avant sont exactement les mêmes. Si ces hypothèses ne sont pas prises en considération ne devrions enlever la symétrie présente dans les fonctions d'appartenance et ceci pour ne pas favoriser une direction du robot par rapport à une autre.

4.3 Énoncé des règles

Les systèmes à logique floue nécessitent une certaine expertise et connaissance en ce qui concerne le fonctionnement du système à commander exprimé sous forme d'une base de règles du type : Si ...(prémisses) ... Alors ... (conclusion).

Une bonne énonciation des règles est une étape cruciale du bon fonctionnement de la commande par logique floue car cette étape constitue la partie « raisonnement » du robot, c'est pourquoi le choix des règles doit être minutieux.

Pour cela nous nous somme basé sur les décisions prises par un "bon" conducteur de voiture qui essaye de rejoindre la trajectoire à suivre et de maintenir son véhicule sur cette dernière tout en nous inspirant des travaux mener au *Département of Intelligent Mechanical System Japan* [8].

Remarque :

- ——— : Position désiré de la ligne détectée (consigne).
- ——— : Position actuelle de la ligne détectée (à l'instant t).
- - - - - - - - - : Position de la ligne détectée a l'instant t-1.
- - - - - - - - - : Position estimée de la ligne a l'instant t+1.

4.3.1 Règles qui favorise un angle de braquage positif (vers la Gauche)

- RP1 Si (alppha est POSITIF et beta est POSITIF) alors phi est POSITIF

En effet dans ce cas (représenté dans la figure ci-dessous) le conducteur doit braquer à gauche afin de rejoindre la ligne et d'éviter qu'elle soit perdu du champs visuel.

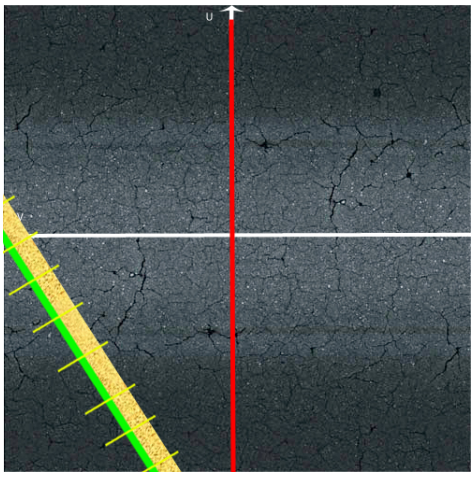


FIGURE 4.3: RP1

Même chose pour les deux règles qui suivent.

- RP2 Si (alppha est POSITIF et beta est ZERO) alors phi est POSITIF

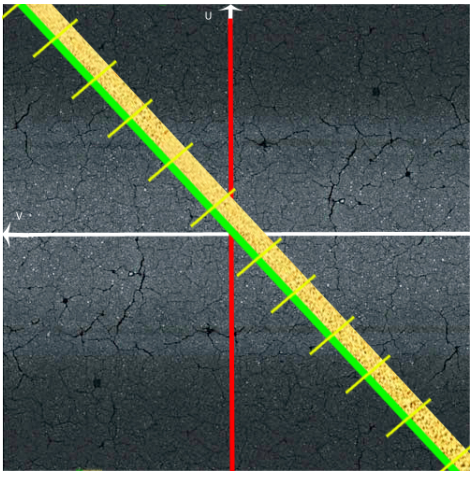


FIGURE 4.4: RP2

- RP3 Si (alppha est ZERO et beta est POSITIF) alors phi est POSITIF
- RP4 Si (alppha est ZERO et beta est ZERO et dalpha est POSITIF et dbeta est POSITIF) alors phi est POSITIF

Dans ce cas malgré que la ligne détectée soit pratiquement confondue avec la ligne désirée, on doit prendre en considération les variations dalpha et dbeta qui nous permettent d'avoir une estimation sur l'emplacement futur de la trajectoire détectée (représenté

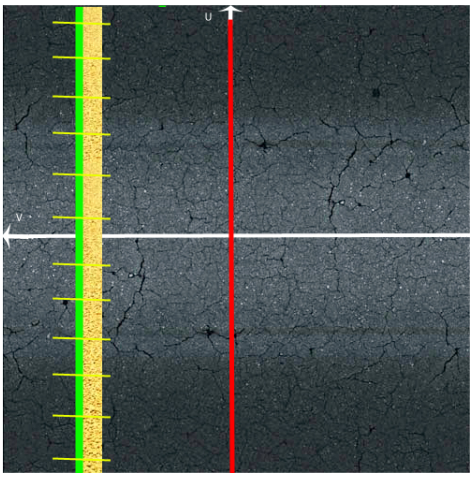


FIGURE 4.5: RP3

dans la figure 4.6), qui s'éloigne de la ligne désiré, et de pouvoir ainsi anticiper et appliquer une commande (braquer à gauche) afin d'éviter cette emplacement futur de la ligne détectée.

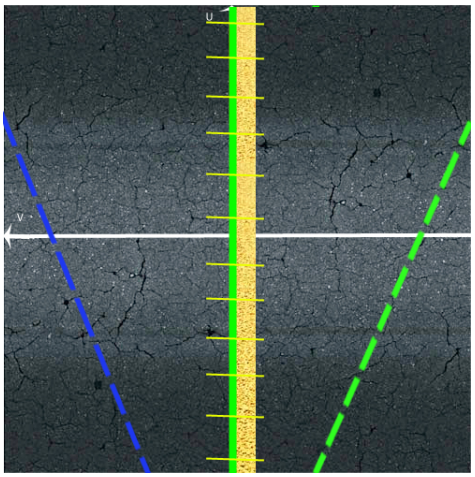


FIGURE 4.6: RP4

Même chose pour les deux règles qui suivent.

- RP5 Si (alppha est ZERO et beta est ZERO et dalpha est POSITIF et dbeta est ZERO) alors phi est POSITIF
- RP6 Si (alppha est ZERO et beta est ZERO et dalpha est ZERO et dbeta est POSITIF) alors phi est POSITIF
- RP7 Si (alppha est NEGATIF et beta est POSITIF et dalpha est ZERO et dbeta est ZERO) alors phi est POSITIF

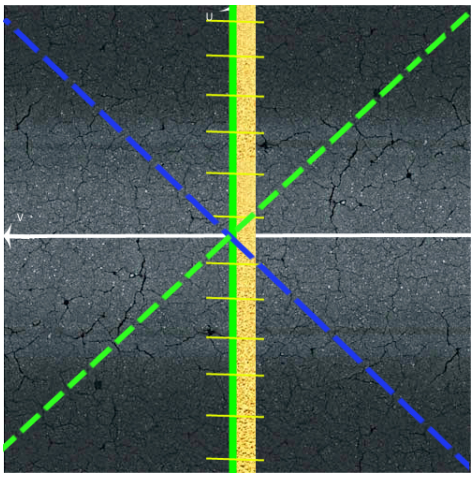


FIGURE 4.7: RP5

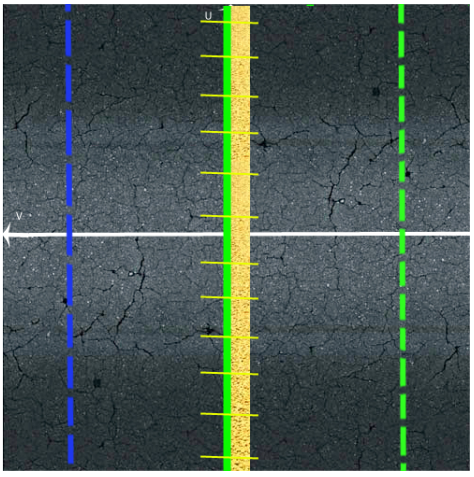


FIGURE 4.8: RP6

Cette règle a été ajoutée pour remédier au problème cité précédemment dû à la déformation de la camera, pour savoir qu'il s'agit bien d'une ligne parallèle au robucar, on vérifie (via dalpha et dbeta) si il n'y pas de variation de alpha et de beta malgré l'avancement du robot si c'est le cas le conducteur doit braquer à gauche afin de rejoindre la ligne (fig 4.9).

4.3.2 Règles qui favorise un angle de braquage négatif (vers la droite)

- RN1 Si (α est NÉGATIF et β est NÉGATIF) alors ϕ est NÉGATIF

En effet dans ce cas (représenté dans la figure4.10) le conducteur doit braquer à droite afin de rejoindre la ligne et d'éviter qu'elle soit perdu du champs visuel.

Même chose pour les deux règles qui suivent (représentés dans les figures 4.11 et 4.12).

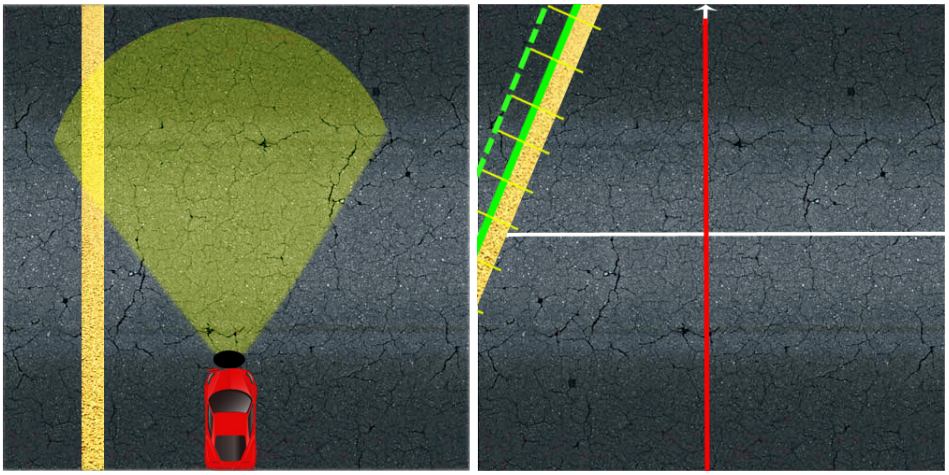


FIGURE 4.9: Problème de la déformation de l'image.

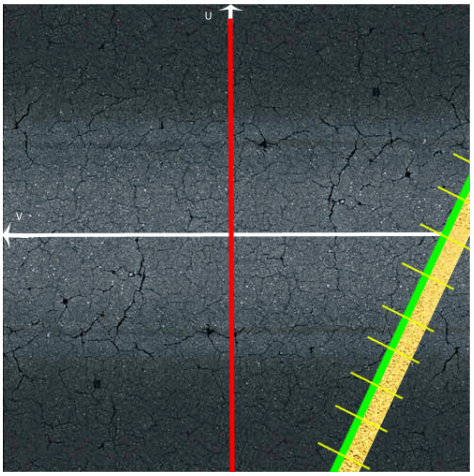


FIGURE 4.10: RN1.

- RN2 Si (α est NÉGATIF et β est ZÉRO) alors phi est NÉGATIF
- RN3 Si (alppha est ZERO et beta est NEGATIF) alors phi est NEGATIF.
- RN4 Si (alppha est ZERO et beta est ZERO et dalpha est NEGATIF et dbeta est NEGATIF) alors phi est NEGATIF

Dans ce cas malgré que la ligne détectée soit pratiquement confondue avec la ligne désirée, on doit prendre en considération les variations dalpha et dbeta qui nous permettent d'avoir une estimation sur l'emplacement futur de la trajectoire détectée (représenté dans la figure 4.13), qui s'éloigne de la ligne désiré, et de pouvoir ainsi anticiper et appliquer une commande (braquer à droite) afin d'éviter cette emplacement futur de la ligne détectée.

Même chose pour les deux règles qui suivent (représentés dans la figure4.14 4.15).

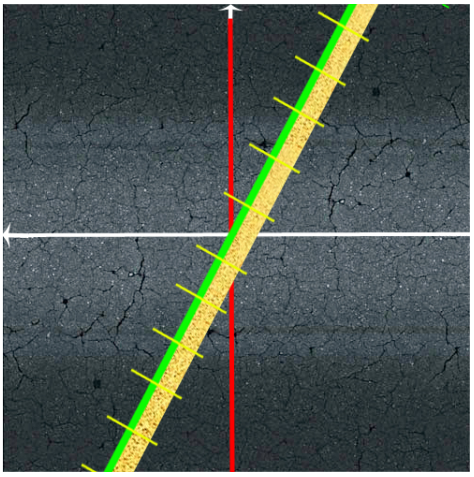


FIGURE 4.11: RN2.

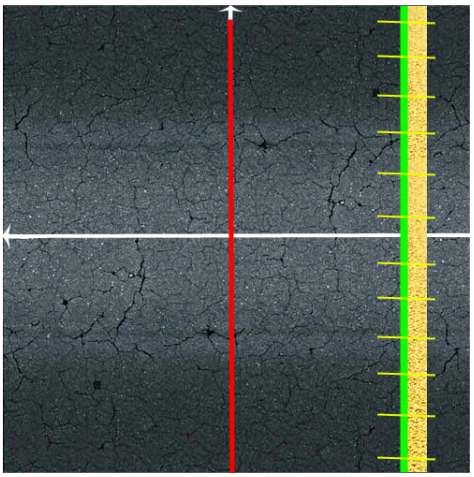


FIGURE 4.12: RN3.

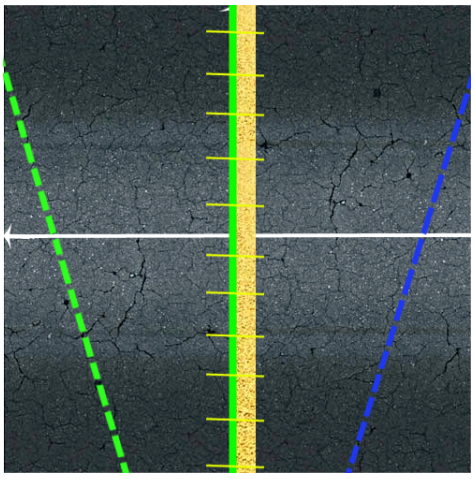


FIGURE 4.13: RN4.

- RN5 Si (alppha est ZERO et beta est ZERO et dalpha est NEGATIF et dbeta est ZERO) alors phi est NEGATIF

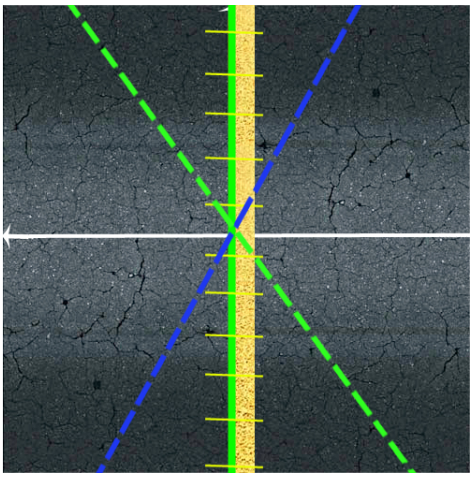


FIGURE 4.14: RN5.

- RN6 Si (alppha est ZERO et beta est ZERO et dalpha est ZERO et dbeta est NEGATIF) alors phi est NEGATIF

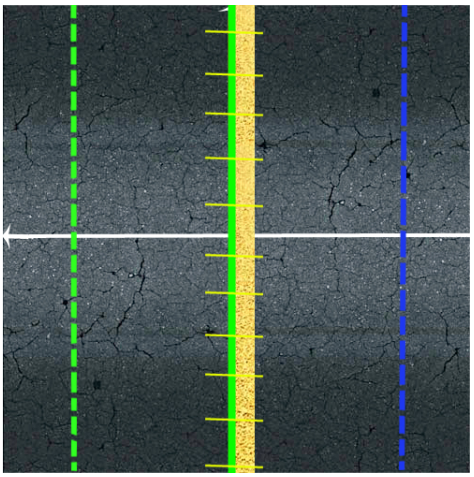


FIGURE 4.15: RN6.

- RN7 Si (alppha est POSITIF et beta est NEGATIF et dalpha est ZERO et dbeta est ZERO) alors phi est NEGATIF

Cette règle a été ajoutée pour remédier au problème cité précédemment dû à la déformation de la camera, pour savoir qu'il s'agit bien d'une ligne parallèle au robucar, on vérifie (via dalpha et dbeta) si il n'y pas de variation de alpha et de beta malgré l'avancement du robot si c'est le cas le conducteur doit braquer à droite afin de rejoindre la ligne (figure 4.16).

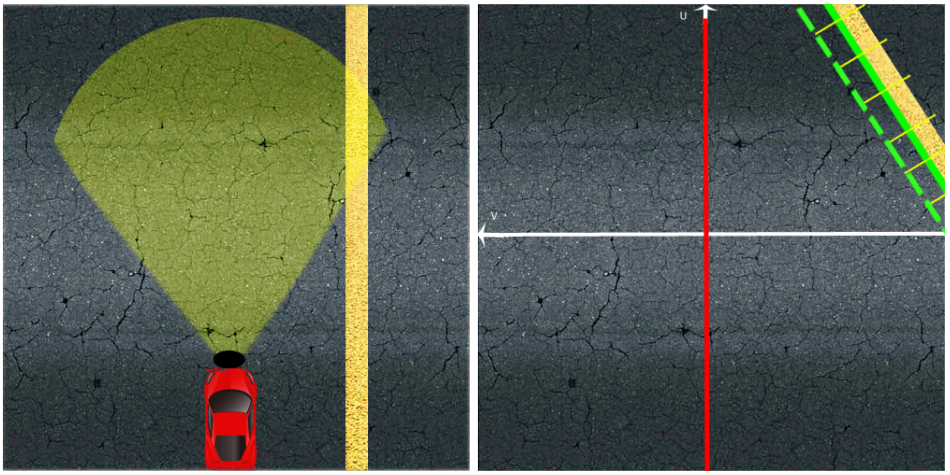


FIGURE 4.16: Problème de la déformation de l'image 2.

4.3.3 Règles qui favorise un braquage nulle (avancer tout droit)

- RZ1 Si (alppha est ZERO et beta est ZERO et dalpha est NEGATIF et dbeta est POSITIF) alors phi est ZERO

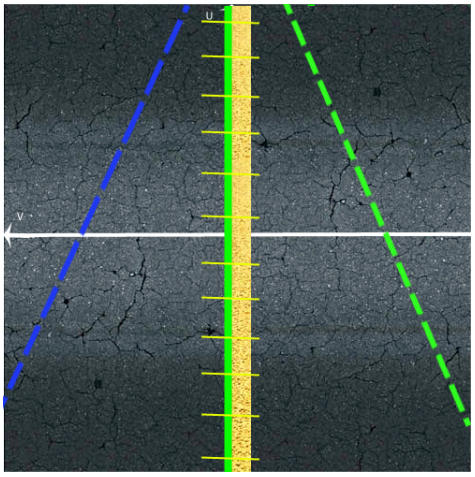


FIGURE 4.17: RZ1.

- RZ2 Si (alppha est ZERO et beta est ZERO et dalpha est ZERO et dbeta est ZERO) alors phi est ZERO
- RZ3 Si (alppha est ZERO et beta est ZERO et dalpha est POSITIF et dbeta est NEGATIF) alors phi est ZERO
- RZ4 Si (alppha est POSITIF et beta est NEGATIF et dbeta est POSITIF) alors phi est ZERO
- RZ5 Si (alppha est POSITIF et beta est NEGATIF et dalpha est POSITIF) alors phi est ZERO

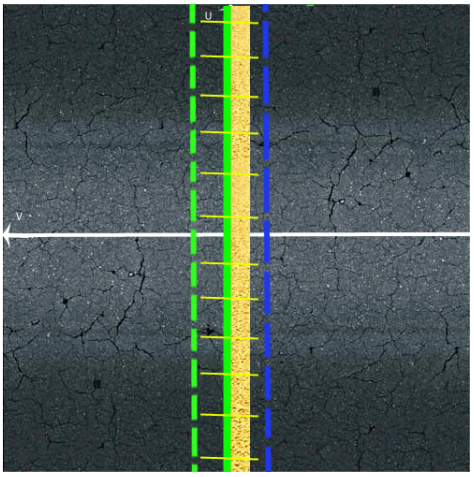


FIGURE 4.18: RZ2.

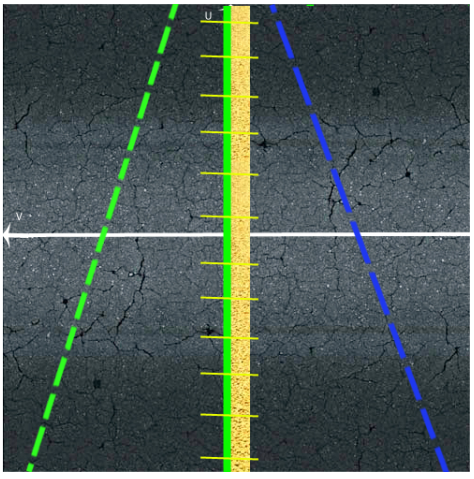


FIGURE 4.19: RZ3.

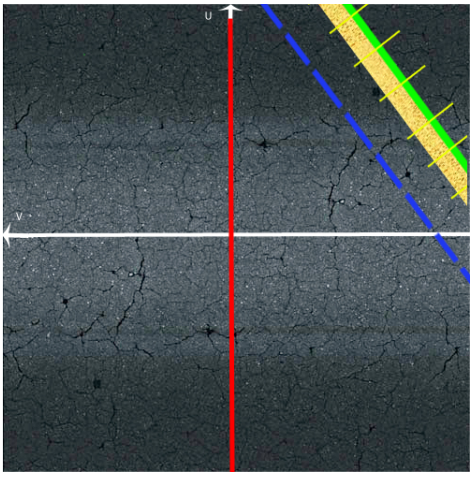


FIGURE 4.20: RZ4.

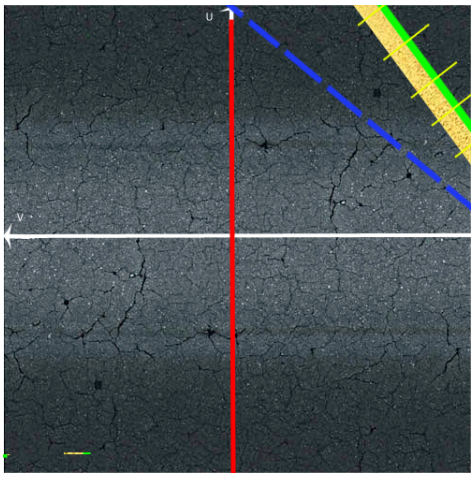


FIGURE 4.21: RZ5.

RZ4 et RZ5 nous permettent de savoir que le marquage routier est bien incliné par rapport au robucar et qu'il ne s'agit pas de la déformation de la caméra, ce qui nous permet de prendre la décision de continuer tout droit pour rejoindre la trajectoire.

Contrairement à la règle RN7 qui nous permet de savoir que le marquage routier est en fait parallèle au robucar et qu'il s'agit de la déformation de la caméra ce qui nous permet de prendre la décision de braquer à gauche pour rejoindre la trajectoire.

- RZ6 Si (alppha est NEGATIF et beta est POSITIF et dbeta est NEGATIF) alors phi est ZERO

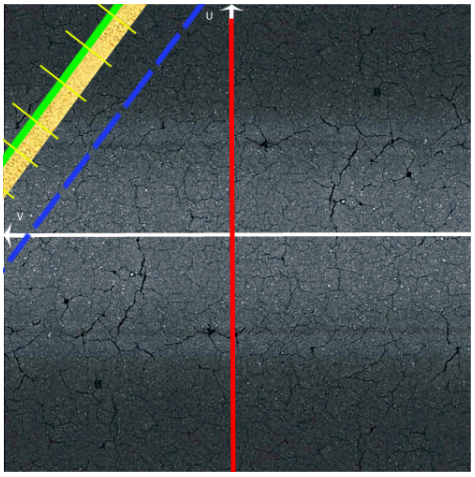


FIGURE 4.22: RZ6.

- RZ7 Si (alppha est NEGATIF et beta est POSITIF et dalpha est NEGATIF) alors phi est ZERO.

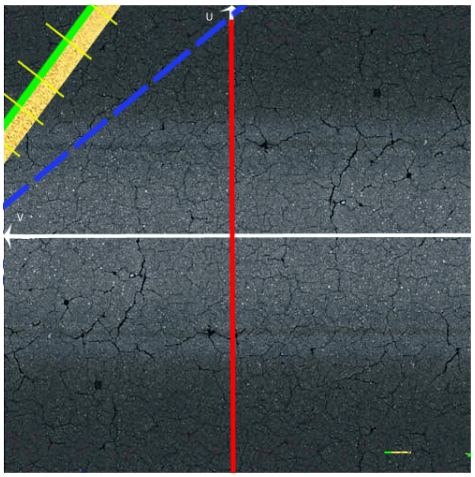


FIGURE 4.23: RZ7.

RZ6 et RZ7 nous permettent de savoir que le marquage routier est bien incliné par rapport au robucar et qu'il ne s'agit pas de la déformation de la caméra ce qui nous permet de prendre la décision de continuer tout droit pour rejoindre la trajectoire.

Contrairement à la règle RP7 qui nous permet de savoir que le marquage routier est en fait parallèle au robucar et qu'il s'agit de la déformation de la caméra ce qui nous permet de prendre la décision de braquer à droite pour rejoindre la trajectoire.

4.4 Tableau des Règles

On précise que dans la commande par logique floue toutes les règles sont appliquées en même temps, cependant certaines règles seront plus représentatives de l'état actuel du robot par rapport à la trajectoire, elles auront donc un degrés d'influence plus important sur la commande.

4.5 Inférence

Le mécanisme d'inférence le plus couramment utilisé, et que nous allons utiliser tout au long de notre travail, est celui dit « de Mamdani ». Il représente une simplification du mécanisme plus général basé sur « l'implication floue » et le « modus ponens généralisé »

n	Règles	α	β	$\dot{\alpha}$	$\dot{\beta}$	ϕ
1	RP1	P	P	-	-	P
2	RP2	P	Z	-	-	P
3	RP3	Z	P	-	-	P
4	RP4	Z	Z	P	P	P
5	RP5	Z	Z	P	Z	P
6	RP6	Z	Z	Z	P	P
7	RP7	N	P	Z	Z	P
8	RN1	N	N	-	-	N
9	RN2	N	Z	-	-	N
10	RN3	Z	N	-	-	N
11	RN4	Z	Z	N	N	N
12	RN5	Z	Z	N	Z	N
13	RN6	Z	Z	Z	N	N
14	RN7	P	N	Z	Z	N
15	RZ1	Z	Z	N	P	Z
16	RZ2	Z	Z	Z	Z	Z
17	RZ3	Z	Z	P	N	Z
18	RZ4	P	N	-	P	Z
19	RZ5	P	N	P	-	Z
20	RZ6	N	P	-	N	Z
21	RZ7	N	P	N	-	Z

TABLE 4.1: Définition des Règles Floues pour le suivie de linge

4.6 Degré d'activation

Le degré d'activation d'une règle est l'évaluation du prédicat de chaque règle par combinaison logique des propositions du prédicat. Le « ET » est réalisé en effectuant le minimum entre les degrés de vérité des propositions [3]

En effet une conclusion ne peut pas être « plus vrai » que les conditions qui la génèrent ceci peut être expliquer à travers cet exemple :
La puissance d'une chaine, constituée par plusieurs maillons, est définit par la puissance du maillon le plus faible.

4.7 Agrégation

L'ensemble flou global de sortie est construit par agrégation des ensembles flous obtenus par chacune des règles concernant cette même sortie. On considère que les règles sont liées par un « OU » logique et on calcule donc le maximum entre les fonctions d'appartenance résultantes pour chaque règle ce qui correspond à la règle la plus représentative de l'état actuel c'est à dire qu'on calcule le maximum :

- des degrés d'activation des règles POSITIF (RP) qui sera noté $M\Phi_P$.
- des degrés d'activation des règles NEGATIF (RN) qui sera noté $M\Phi_N$.
- des degrés d'activation des règles ZERO (RZ) qui sera noté $M\Phi_Z$.

4.8 Deffuzzification

A la fin de l'inférence, le braquage (commande floue) est déterminé mais il n'est pas directement utilisable pour commander le braquage des roues. Il est nécessaire de passer du « monde flou » au « monde réel », c'est la défuzzification. Il existe plusieurs méthodes, la plus souvent utilisée est celle du calcul du « centre de gravité » de l'ensemble flou, cependant pour simplifier les calculs nous utilisons une autre méthode qui est aussi très pratique, cette méthode s'intitule *Moyenne pondérée*(éq 4.1).

$$\Phi = \frac{-\Phi_{max} M\Phi_N + 0 \times M\Phi_Z + \Phi_{max} M\Phi_p}{M\Phi_N + M\Phi_Z + M\Phi_P} \tag{4.1}$$

4.9 Résultats

Les membres du CDTA nous ont conseillé d'implémenter directement notre programme sur le robot sans passer par la simulation mais nous nous sommes d'abord assuré d'avoir bien limité l'angle de braquage ainsi que la vitesse et ceci pour ne pas endommager le matériel, en plus du fait qu'il y avait un boutant d'arrêt d'urgence pour éviter toute collision du robot lors du suivie de trajectoire

Les résultats sont illustrés dans la figure 4.24.

On remarque que la commande (angle de braquage) est de l'ordre de 10(-1)rad ce qui démontre que cette méthode de commande assure le suivie sans oscillations ce qui permet de ne pas endommager l'actionneur (vérin) de braquage.

Les pics présent dans les graphes sont en partie due :

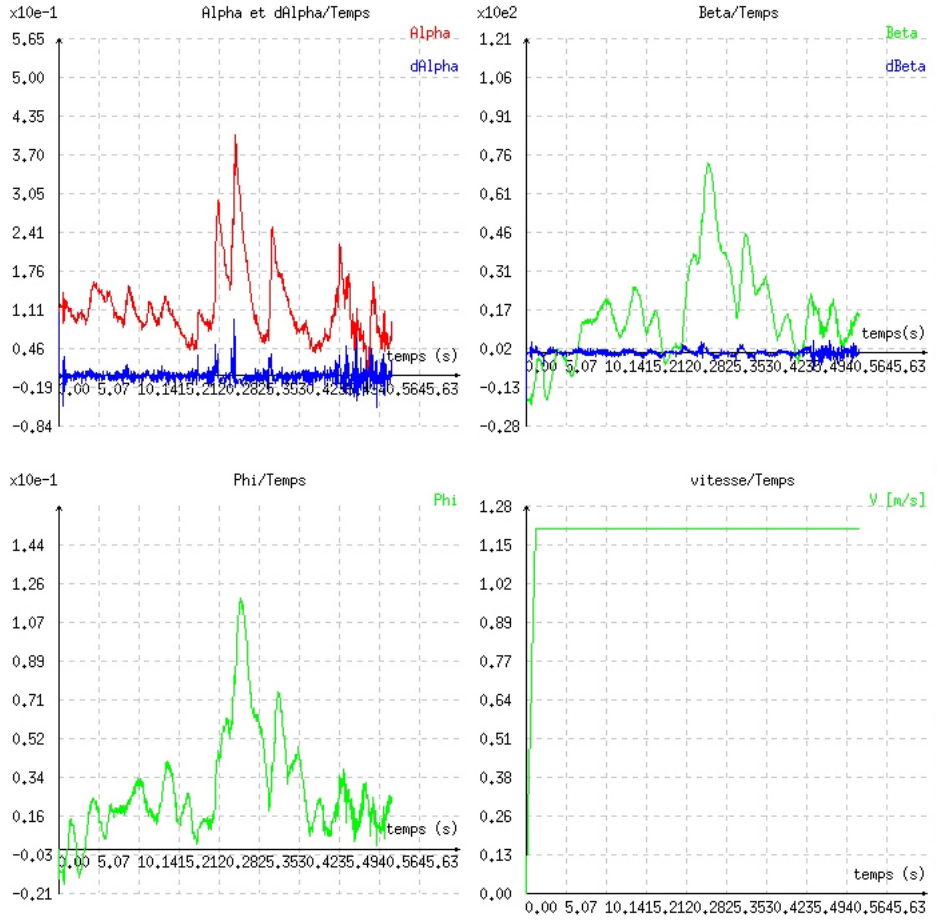


FIGURE 4.24: Résultats des essais en temps réel du suivie de ligne.

- à la variation brusque de l'orientation de la trajectoire.
- au fait que que le robot ne réagit pas instantanément à la commande appliqué ceci est justifié par son inertie (lente).

rappelons que la synthèse de la commande à été faite sous les hypothèses suivantes :

- les roues du robot sont parfaitement commandables en vitesse et en braquage.
- les quartes roues du robucar ont le même rayon (parfaitement identiques).
- la vitesses appliquer aux deux roues avant est exactement la même.
- l'angle maximum ainsi que la vitesse angulaire de braquage des deux roues avant sont exactement les mêmes.

On en conclue que les résultats obtenus nous permettent d'augmenter la vitesse de suivie et atteindre la vitesse maximale du robot (5m/s), ils démontrent encore une fois la robustesse de la commande par logique floue.

Chapitre 5

Évitement d'obstacles

L'évitement d'obstacles est un comportement de base présent pratiquement dans tous les robots mobiles (holonomes et non holonomes), Il est indispensable pour permettre au robot de fonctionner dans un environnement statique ou dynamique. Traditionnellement le domaine de l'évitement d'obstacle se divise en deux grandes catégories :

- **L'approche Globale**.
- **L'approche Locale**.

5.1 Approche Globale

Cette approche repose sur la génération d'un chemin ou d'une trajectoire à l'aide d'un planificateur, ce dernier dispose d'une carte de l'environnement du robot mobile qui doit être statique, et qui va assurer la connaissance de la position des obstacles, de plus cette approche suppose qu'on peut toujours localiser le robot dans cet environnement, pour pouvoir ensuite générer le chemin (l'itinéraire) reliant la position initiale du robot et la cible (position finale) en évitant tout au long de la trajectoire la collision avec les obstacles connus.

Il existe d'autres extensions de l'approche globale, chaque extension pressente une amélioration ou une solution qui vise à augmenter les performances de cette approche, par exemple :

- **L'algorithme de bande élastique(elastic band)** Cet algorithme permet de déformer la trajectoire initialement conçue durant la navigation du robot, pour pouvoir éviter les obstacles non prévus.[6]
- **L'approche de la fenêtre dynamique(Dynamic Window Approach – DWA)** Cette approche est développée pour commander des robots omnidirectionnels (free flying robot). [6]

Toutes ces méthodes se basent cependant sur la modification d'un chemin ou d'une trajectoire préalablement établis. Par conséquent, elles requièrent la construction d'une carte pour générer l'itinéraire, ainsi que la localisation du robot par rapport à celui-ci. [6]

5.2 Approche Locale

Les méthodes basées sur l'approche Locale utilisent les informations conceptuelles qui proviennent des capteurs pour se renseigner sur l'environnement extérieur proche du robot en temps réel, et pour pouvoir le guider à travers une commande pour atteindre son objectif en tenant compte de la présence des différents obstacles qu'il doit éviter durant la navigation. Les capteurs les plus utilisés et les mieux adaptés pour cette méthode sont les capteurs à infrarouge, les lasers et les capteurs à ultrasons qui donnent la possibilité de caractériser les obstacles au voisinage du robot.

Après traitement de données, les capteurs fournissent des paramétrés utiles pour que le robot puisse accomplir la tache de l'évitement d'obstacles, comme on peut le voir sur la figure 5.1 qui montre les différents paramètres provenant de seize capteurs ultrasons, montés sur un robot mobile (Super Scout).

Où d_{coll} est la distance entre le centre M du robot et le point B le plus proche de l'obstacle lorsque le robot est situé à une distance d_+ de cet obstacle, et tel que α est l'orientation entre la tangente $\vec{T}$ à l'obstacle et la direction $\vec{X}_M$ du véhicule [6].

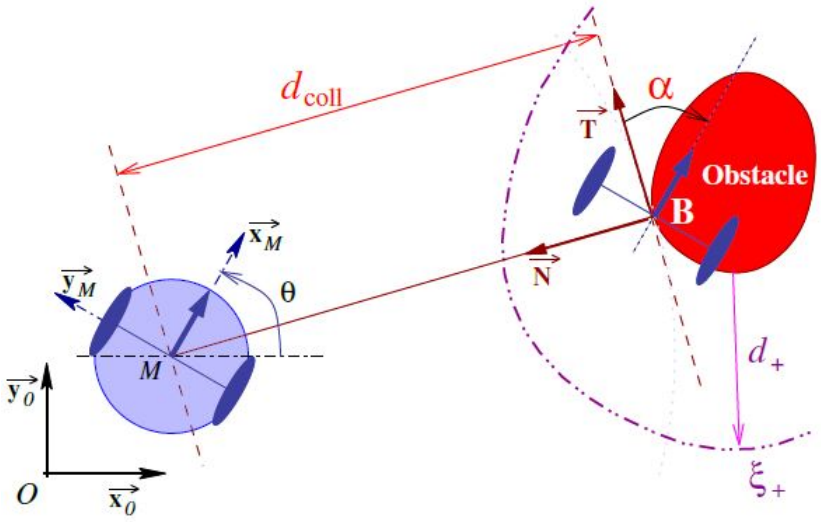


FIGURE 5.1: Paramètres pour l'évitement d'obstacle.[6]

Les principaux méthodes qui se basent sur l'approche locale sont les suivants :

- **La méthode des potentiels**
- **Le formalisme du suivie de chemin**

5.2.1 La méthode des potentiels

Cette approche est adaptée à la problématique du contrôle de mouvement des robots dans un milieu encombré, le principe est de créer un champs de potentiel artificiel qui va guider le robot dans son environnement, pour qu'il puise atteindre la cible ou le but souhaité, où les obstacles à éviter sont représentés par des potentiels répulsifs et le but à atteindre joue le rôle d'un potentiel attractif qui représente un minimum de potentiel que le robot doit chercher pendent sa navigation. La direction du déplacement du robot et son accélération vont être calculées respectivement à partir de la direction et de l'intensité de la force qui dérive du potentiel. Le robot va donc glisser le long du gradient de potentiel qui résulte de la somme des différents potentiels pour atteindre son but.

Le problème essentiel de cette approche est de pouvoir minimiser l'influence des *minima locaux* ou ce qu'on appel *puits de potentiel*. Lorsque le robot tombe dans un puit de potentiel il peut se trouvé immobilisé dans une situation dont il ne peut pas sortir comme le montre la fig 5.2.

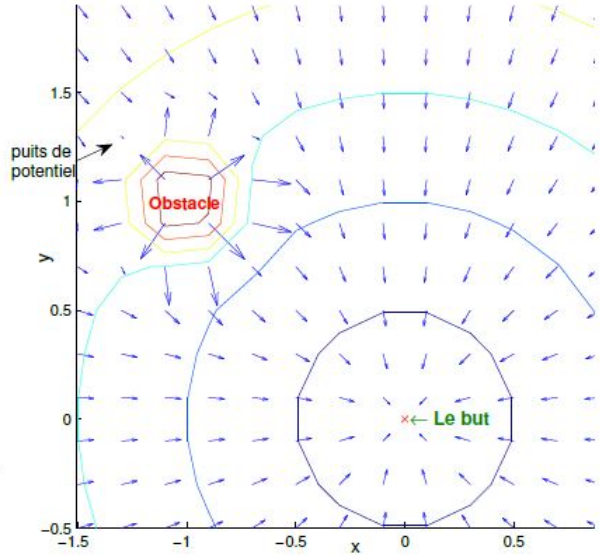


FIGURE 5.2: Principe de la méthode des potentiels. [6]

Pour tenter de faire sortir le robot de cette situation des solutions sont proposées qui nécessitent la réorientation vers d'autres méthodes tel que *la méthode des potentiels rotatifs* et *le formalisme du suivie de chemin* qui consiste à assurer la convergence géométrique du robot vers un chemin de référence défini à priori.

5.2.2 Méthode des potentiels rotatifs

L'idée de la méthode des potentiels rotatifs est de générer un champ de potentiel autour de l'obstacle qui impose au robot de suivre une enveloppe particulière, sur laquelle on doit assurer que la norme de la force répulsive soit non nulle. Le potentiel répulsif doit donc être capable de maintenir la commande du robot au voisinage de l'obstacle afin de pouvoir le contourner, et cela même en l'absence de potentiel attractif. Pour plus de détail ce référer à [6].

5.3 Application de la logique floue pour l'évitement d'obstacles

Un des principaux objectifs de notre projet et que le robucar soit présent dans un environnement externe et qu'il puisse interagir avec des obstacles, des véhicules, ou éventuellement d'autres robots mobiles qui pourraient être présents dans cet environnement externe. La commande par la logique floue nous a montrée sa robustesse, c'est pourquoi nous l'utilisons également pour l'évitement d'obstacles.

Pour réaliser un évitement d'obstacles nous avons décidé de développer une méthode simplifiée qui consiste à utiliser seulement deux paramètres (Dmin, Gamma(Dmin)) sous certaines conditions ;

où Dmin correspond la plus petite distance qui sera déterminée par une fonction ayant en entrée toute les valeurs des distances estimées et retournées par le capteur laser Lms511

et Gamma(dmin) l'angle correspondant à la distance Dmin (voire figure 5.3).

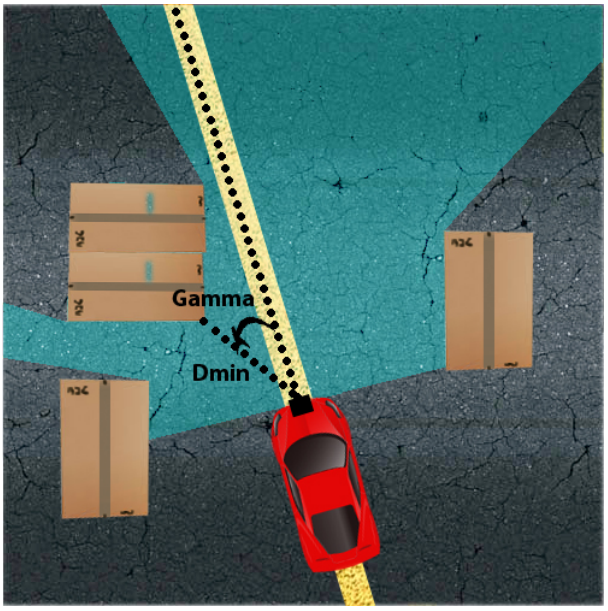
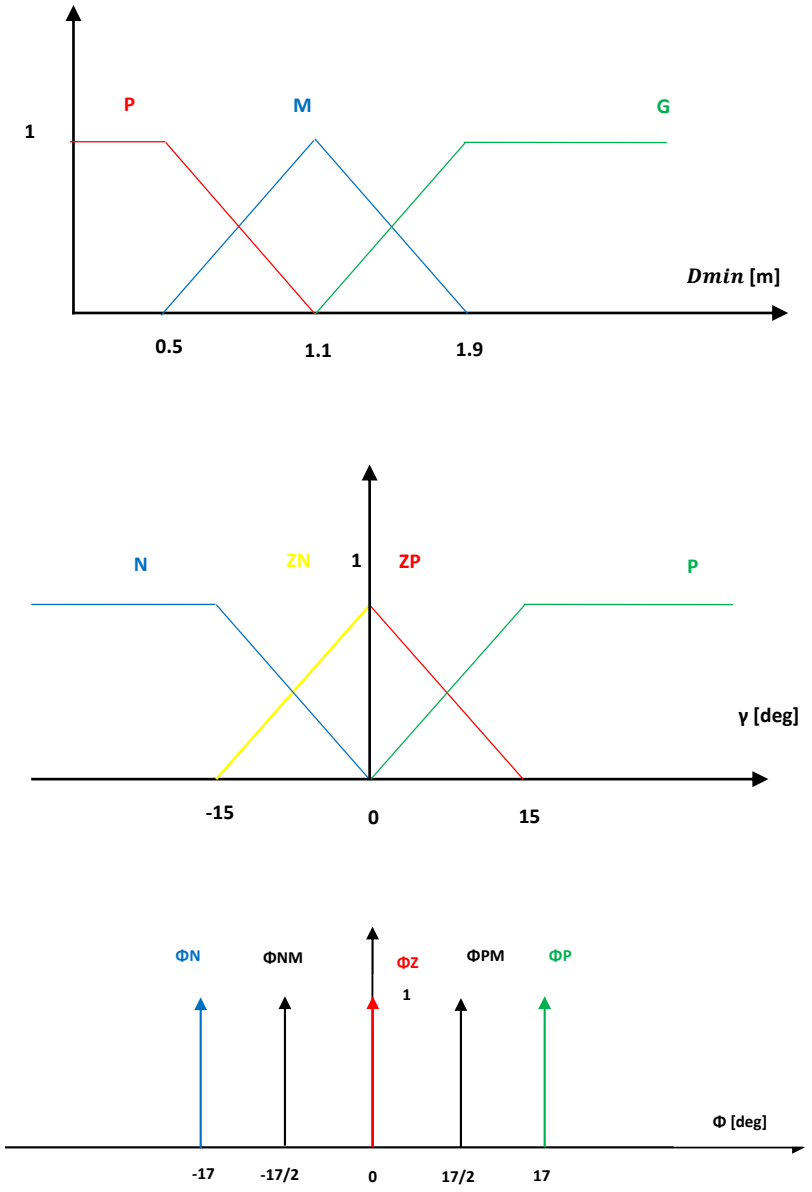


FIGURE 5.3: Représentation des pramétres Dmin et γ .

5.3.1 Fuzzification

5.3.1.1 Les fonctions d'appartenance

Les fonctions d'appartenances utilisées pour l'évitement d'obstacles sont les suivant :



tel que :

D_{min} peut être :	et γ peut étre :
• G : Grande.	• N : NEGATIF.
• M : Moyenne.	• NZ : NEGATIF ZERO.
• P : Petite.	• PZ : POSITIF ZERO.

Règles	D_{min}	γ	ϕ
1	PT	N	PM
2	PT	ZN	P
3	PT	ZP	N
4	PT	P	N
5	M	N	PM
6	M	ZN	P
7	M	ZP	N
8	M	P	N
9	G	N	Z
10	G	ZN	P
11	G	ZP	N
12	G	P	Z

TABLE 5.1: Définition des Règles Floues pour l'évitement d'obstacles.

5.3.2 Règles

Le tableau (5.1) résume les règles de la commande par logique floue appliquée pour l'évitement d'obstacles.

5.3.3 Degrés d'activation

Le degrés d'activation de chaque règle est égale au minimum des degrés de vérité des propositions (Prémisse) qui la génèrent.

5.3.4 Agrégation

On calcule le maximum :

- des degrés d'activation des règles POSITIF (RP) qui sera noté $M\Phi_P$.
- des degrés d'activation des regles POSITIF ZERO (RPZ) qui sera noté $M\Phi_{PZ}$.
- des degrés d'activation des regles ZERO (PZ) qui sera noté $M\Phi_Z$.
- des degrés d'activation des regles NEGATIF ZERO (RZN) qui sera noté $M\Phi_{NZ}$.

- des degrés d'activation des regles NEGATIF (RN) qui sera noté $M\Phi_N$.

5.3.5 Défuzzification pour l'évitement

$$\Phi = \frac{\Phi_{max}(-M\Phi_N - 0.5M\Phi_{NZ} + 0.5M\Phi_{pZ} + M\Phi_p) + 0 \times M\Phi_Z}{M\Phi_N + M\Phi_{NZ} + M\Phi_Z + M\Phi_{pZ} + M\Phi_P} \quad (5.1)$$

5.4 Commande du support de la caméra

L'évitement d'obstacle pourrait en effet causer la perte de de la trajectoire du champs visuelle de la caméra dans le cas où cette dernière est immobile par rapport au robucar :

Un programme de détection automatique de la trajectoire nous aurais peut être permis la "redétection" de la trajectoire sous certaines conditions :

-Soit il y a une seule ligne tracé sur la route. -Soit il y a plusieurs lignes tracés mais d'intensité de couleurs différentes.

et cela pour qu'il n'y ait pas de confusion lors de la détection automatique et ainsi ne pas choisir une mauvaise trajectoire.

Pour éviter toutes ces hypothèses nous avons préféré maitre au point une commande du support de la camera pour pouvoir garder un point de la trajectoire dans le champs visuelle de la caméra, les coordonnées de ce point sont (0, Yrobot+portée initiale de la caméra) dans le repère orthonormé dont l'origine est le point à partir duquel le robucar a commencé à évité l'obstacle.

Où la portée initiale de la caméra : représente la distance entre le point perçu au centre de l'image et le point représentant la projection de la camera sur le sol

On commandera donc les angles du support de la camera (PTU) – tetha PTU par rapport au plan vertical – gama PTU par rapport au plan horizontal

Comme il est montré dans la figure (ci-dessus) une rotation du robot avec un angle (tetharobot) implique à ce que tetha PTU = tetha PTU rotation = - tetharobot, pour garder la camera en direction du point de coordonnées (0, Yrobot+portée initiale de la caméra)

comme il est montrer dans la figure (ci-dessus) une translation du robot avec distance (Xrobot) implique :

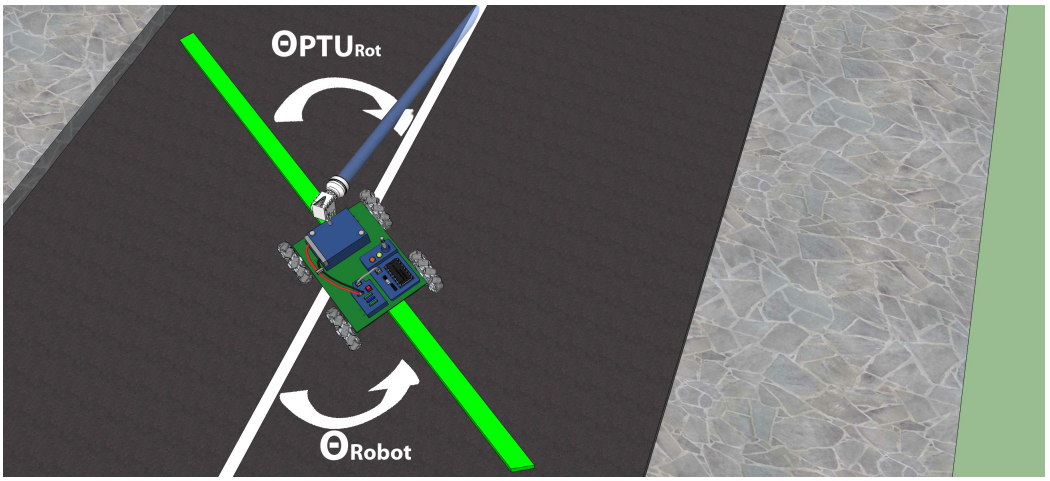


FIGURE 5.4: Représentation de θ_{PTURot}

non seulement à tetha PTU = tetha PTU translation, pour garder la camera en direction du point de coordonnées (0, Yrobot+portée initiale de la caméra).

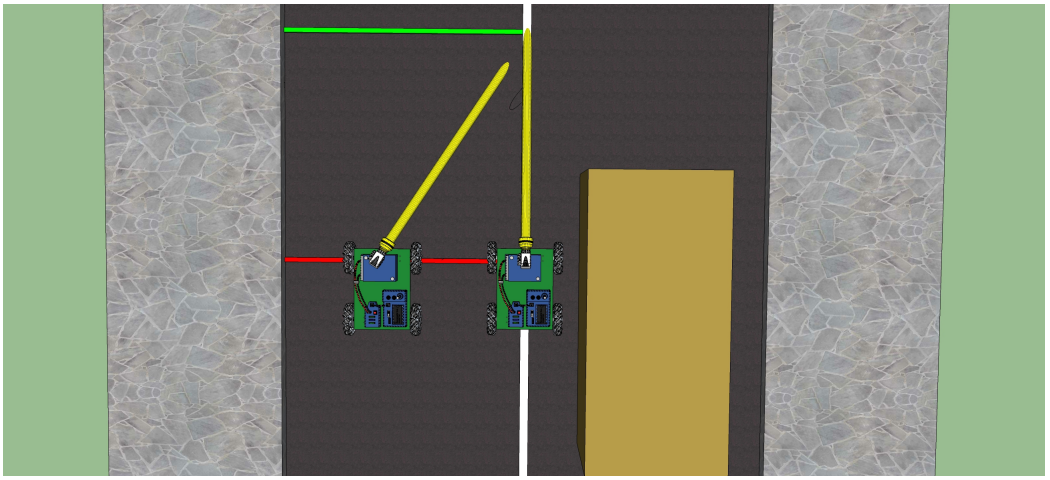


FIGURE 5.5: Représentation de θ_{Trans}

$$\theta_{PtuTrans} = arctan(\frac{X_{Robot}}{Portee \quad de \quad la \quad Camera(X_{Robot} = 0)} \tag{5.2}$$

mais aussi $\gamma_{PTU} = \gamma_{PtuTrans}$ pour augmenter la portée de la camera et ainsi garder le point de coordonnées (0, y_{Robot} + portée initiale de la caméra) dans le champs visuel de la caméra.

$$\gamma_{PTU} \quad = \quad arctan \quad (\frac{Portee \quad de \quad la \quad Camera(X_{Robot})}{Hauteur \quad de \quad Camera}) \tag{5.3}$$

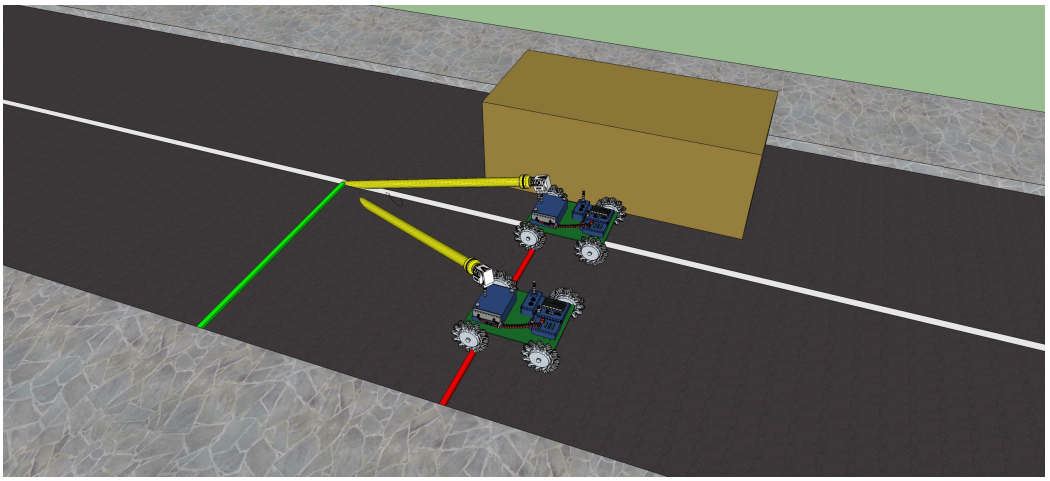


FIGURE 5.6: Influence de γ_{PTU}

Où :

$$Portee \quad de \quad la \quad Camera(X_{Robot}) = \sqrt{X^2 + Portee \quad de \quad la \quad Camera(X_{Robot} = 0)} \tag{5.4}$$

5.5 Application de la logique floue pour le repositionnement

On enchainera avec cette partie "repositionnent" directement après l'évitement d'obstacle, qui sera déterminé par une variation brusque de Dmin, afin de repositionner le robot sur la trajectoire.

Les paramètres utilisés pour la synthèse de la commande dans cette partie sont X_{Robot} et θ_{Robot} (voir fig 5.7)

Étant donné qu'il subsiste une forte ressemblance, nous prolongeons le raisonnement établit dans la partie suivie de trajectoire, en enlevant les règles correspondantes à la déformation de la caméra, sans pour autant détailler le cheminement de la synthèse de la commande.

5.5.1 Fuzzification

5.5.1.1 Les fonctions d'appartenance

Les fonctions d'appartenances utilisées pour le repositionnent sont les suivants :

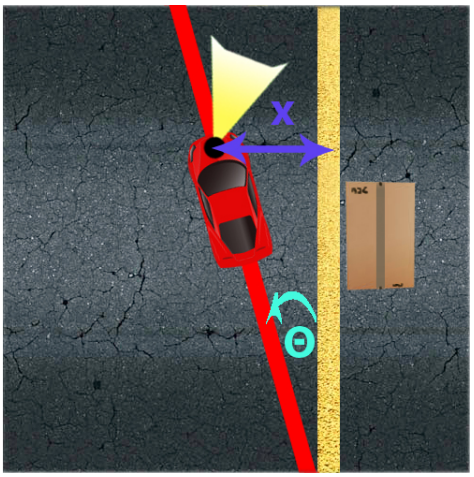
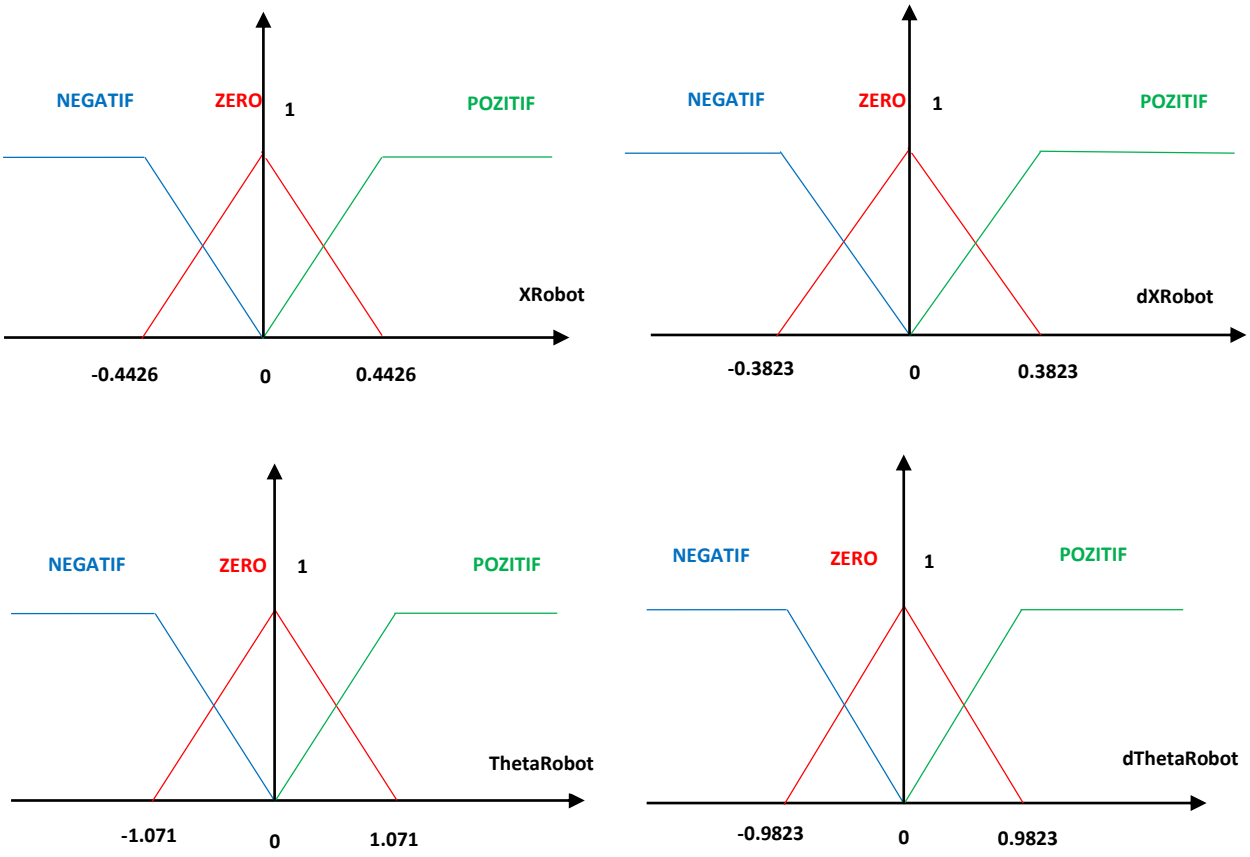
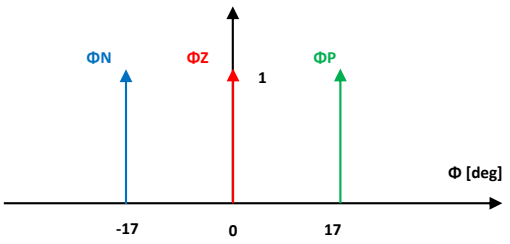


FIGURE 5.7: Paramètres utilisées pour le repositionnent



n	Règles	θ_{Robot}	X_{Robot}	$\dot{\theta}_{Robot}$	$\dot{X}_{Robot}$	ϕ
1	RP1	N	Z	-	-	P
2	RP2	N	Z	-	-	P
3	RP3	Z	P	-	-	P
4	RP4	Z	Z	P	P	P
5	RP5	Z	Z	P	Z	P
6	RP6	Z	Z	Z	P	P
7	RN1	P	N	-	-	N
8	RN2	P	Z	-	-	N
9	RN3	Z	N	-	-	N
10	RN4	Z	Z	N	N	N
11	RN5	Z	Z	N	Z	N
12	RN6	Z	Z	Z	N	N
13	RZ1	Z	Z	N	P	Z
14	RZ2	Z	Z	Z	Z	Z
15	RZ3	Z	Z	P	N	Z
16	RZ4	N	N	-	-	Z
17	RZ5	P	P	-	-	Z

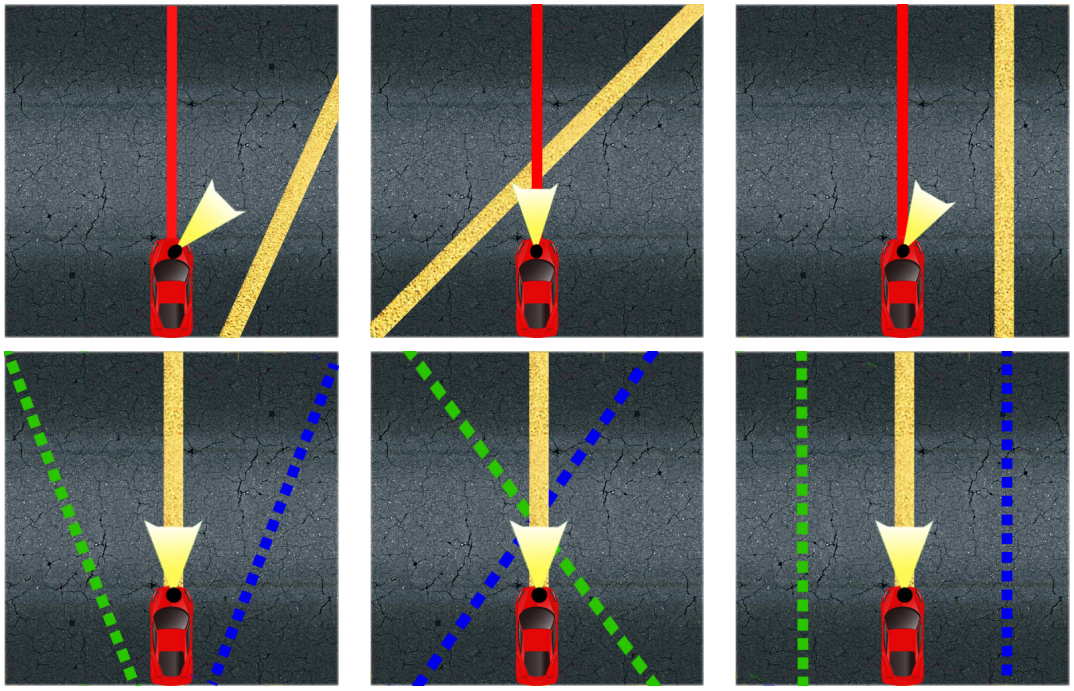
TABLE 5.2: Définition des Règles Floues pour le repositionnent



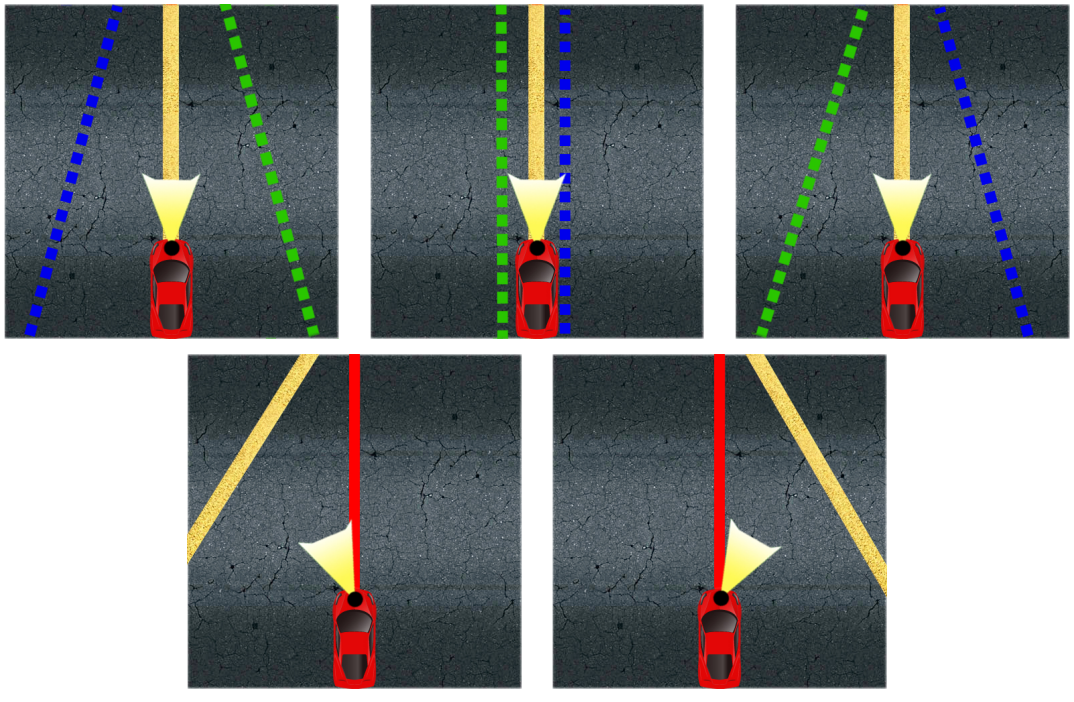
5.5.2 Règles

Pour éviter toutes ambiguïtés, nous avons préférés schématiser respectivement chaque règle dans les figures ci-dessous.

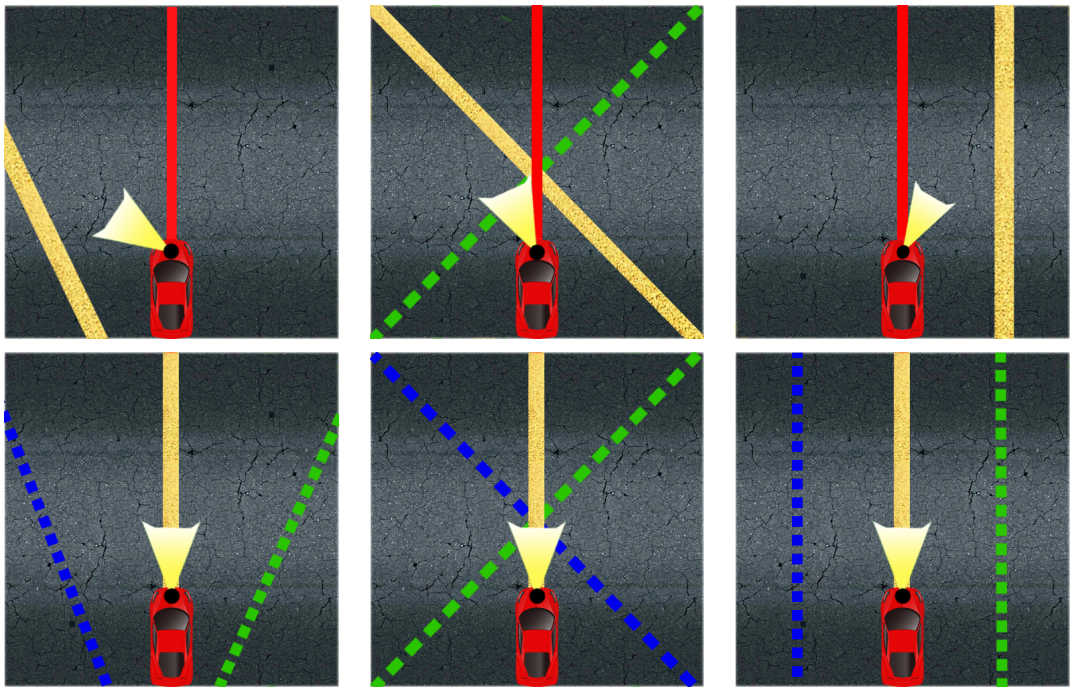
Les règles qui favorisent un angle braquage NEGATIF (braquer à droite) sont les suivants :



Les règles qui favorisent un angle braquage ZERO (continuer tout droit)sont les suivants :



Les règles qui favorisent un angle braquage POZITIF (braquer a gauche) sont les suivants :



5.5.3 Défuzzification pour le repositionnent

La relation suivante permet de défuzzifier la commande floue et ainsi de pouvoir l'implémenter sur le robot pour qu'il puisse rejoindre la trajectoire.

$$\Phi = \frac{-\Phi_{max} M\Phi_N + 0 \times M\Phi_Z + \Phi_{max} M\Phi_p}{M\Phi_N + M\Phi_Z + M\Phi_P} \tag{5.5}$$

Chapitre 6

Conclusion générale

Les travaux présentés dans cette thèse concernent la navigation référencée vision d'un robot mobile de type voiture dans un environnement statique.

Dans notre thèse nous avons muni le robot mobile de type voiture voiture ≪ robucar ≫ d'une commande par logique floue lui permettant de détecter via une caméra et de suivre, dans un environnement interne et externe, une trajectoire (marquage routier) sans oscillations et ainsi préserver les l'actionneur (vérins) de braquage.

Nous avons également présenter une méthode d'évitement d'obstacle avec une commande par logique floue, ainsi qu'une autre méthode permettant de commander la plateforme qui soutien la caméra pour garder la trajectoire dans le champs visuel.

Nous avons terminé par présenter une méthode repositionnent du Robot sur la trajectoire après évitement d'obstacle.

Les résultats du travail présenté dans ce mémoire permettent de dégager les perspectives suivantes :

- Améliorer le programme de détection de la trajectoire.
- La poursuite de ce travail devrait naturellement s'orienter vers son application dans environnement dynamique.
- Développer ces méthodes pour pouvoir passer à leurs applications à grande échelle.

Annexe A

Annexe A

A.1 La programmation orientée objet

A.1.1 Les deux grands mondes de la programmation

Il existe deux grands types de programmation :

- **Le Procédural** ou structuré comme le C où on définit des fonctions s'appelant mutuellement. Chaque fonction ou procédure est associée à un traitement particulier qui peut être décomposé en sous traitements jusqu'à obtenir des fonctions basiques. Globalement, le procédural travaille sur l'action ou le verbe. Par exemple, pour calculer la vitesse d'une voiture, la philosophie du procédural conduira à faire : `calculVitesse(voiture)` partout où cela est nécessaire. Le code faisant référence à une voiture est donc "fondu" et dispersé dans l'ensemble des programmes. [5]
- **L'Orienté Objet (ou OO)** où on manipule uniquement des objets c'est-à-dire des ensembles groupés de variables et de méthodes associées à des entités intégrant naturellement ces variables et ces méthodes. Dans cette configuration, le sujet est prépondérant : disposant d'un objet Voiture, on effectuera spontanément : `Voiture.calculVitesse()`. Tout le code touchant à l'objet Voiture se trouve ainsi regroupé. [5]
  Remarque :

  Le C++ est l'un des langages de programmation orientée objet, ou l'objet est créé via la déclaration d'une structure informatique particulière qui s'appelle **classe**, où cet objet représente une instance de cette classe.

A.1.2 Avantages de la programmation Orientée Objet

L'Orienté Objet présente d'énormes avantages :

- En général, en OO, cohabitent deux types de développeurs : ceux qui conçoivent les objets et ceux qui utilisent ces objets dans leurs programmes. De cette façon, la programmation orienté objet permet de diviser la complexité [5], car chaque utilisateur peut utiliser des objets prédéfinit qui le concerne pour les implémenter dans un programme dans le but de réaliser un projet.
- Les informations concernant un domaine étant centralisées en objets, il est facile de sécuriser le programme en interdisant ou autorisant l'accès à ces objets aux autres parties du programme.[5]

A.2 La notion de classe

Une classe permet de regrouper dans une même entité des données et des fonctions membres (appelées aussi méthodes) permettant de manipuler ces données. La classe est la notion de base de la programmation orientée objet. Il s'agit en fait d'une évolution de la notion de structure, qui apporte de nouvelles notions orientées objet absolument fondamentales. Un objet est un élément d'une certaine classe : on parle de l'instance d'une classe.

A.3 Ubuntu

Fondé sur la distribution Linux Debian, Ubuntu est un système d'exploitation open source, il est disponible gratuitement, pour les utilisateurs et aussi pour les entreprises, Ubuntu présente plusieurs avantages par rapport aux autres systèmes d'exploitation.

A.4 Le formalisme de fonctions de tâches

Initialement conçue pour des bras manipulateurs ce formalisme ne peut être directement étendu au cas des robots mobiles non-holonomes du fait de la contrainte de roulement sans glissement. En effet, cette contrainte entrave les mouvements de la caméra, et les tâches robotiques nécessitent alors la réalisation de manœuvres qui peuvent conduire à la perte du motif visuel [6]. la solution consiste a introduire un degrés de

liberté supplémentaire pour rendre les mouvements de la caméra holonomes par l'introduction d'un degré de liberté supplémentaire permettant à la caméra de se déplacer indépendamment du robot mobile, pour pouvoir en fin appliquer le formalisme des fonctions de tâches.

Bibliographie

[1] SICK AG. *LMS5xx Laser Measurement Sensors*. Sick Sensor Intelligence, Erwin-Sick-Str. 1 79183 Waldkirch Germany, fevrier 2015.

[2] François Chaumette. *De la perçeption à l'action : l'asservissement visuel, de l'action à la perçeption : la vision active*. PhD thesis, Université de Reinnes 1 Institut de formation Superieure en Informatique et en Communication, Janvier 1998.

[3] F. Chevrie and F. Guély. La logique floue. *Collection Technique Schneider*, (191) :1–30, Mars 1998.

[4] AMMOUR Rabah et BOUSSIF Abderraouf. *COMMANDE RÉFÉRENCÉE VISION DU ROBOT MOBILE ROBUCAR*. PhD thesis, Ecole Nationale Polytechnique (ENP), Juin 2012.

[5] Bertrand Florat. Chapitre 1 l'orienté objet. http://www.florat.net/tutorial-java/chapitre01.html, Consulté en Mai 2015.

[6] David FOLIO. *Stratégies De Commande Référencées Multi-Capteurs Et Gestion De La Perte Du Signal Visuel Pour La Navigation D'un Robot Mobile*. PhD thesis, Université Paul SABATIER de Toulouse III, Sept 2007.

[7] Open Source Robotics Foundation. Documentation ros. http://wiki.ros.org/fr, Consulté en Mai 2015.

[8] Shoichi Maeyama Keigo Watanabe et Tatsuya Kato. Image-based fuzzy trajectory tracking control for four-wheel steered mobile robots. *Proc. of the 37th Annual Conference of the IEEE Industrial Electronics Society*, Mars 2012.

[9] Shoichi Maeyama Keigo Watanabe et Tatsuya Kato. Obstacle avoidance for mobile robots using an image-based fuzzy controller. *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, 69 :6392 – 6397, March 2013.

[10] Aaron Martinez and Enrique Fernández. *Learning ROS for Robotics Programming A practical, instructive, and comprehensive guide to introduce yourself to ROS, the top-notch, leading robotics framework*. PACKT, Birmingham B3 2PB, UK, 2013.

[11] Visual Servoing Platform VISP. Bibliothèque multi plate-forme modulaire pour l'asservissement visuel. http://www.irisa.fr/lagadic/visp/visp.html, Consulté en Janvier 2015.