UNIVERSITE Sidi Mohamed Ben Abdellah

Faculté Des Sciences Et Techniques Fès
Département d'Informatique

Projet de Fin d'Etudes

Master Sciences et Techniques
Systèmes Intelligents & Réseaux

Système de détection et d'évitement des obstacles pour la locomotion des personnes déficientes visuelles

Lieu de stage : Laboratoire systèmes intelligents et applications

Réalisé par : Abdelhalim KAMRANI

Soutenu le : 15 Juin 2016

Encadré par :

Pr. Said NAJAH

Devant le jury composé de :

Pr. Loubna LAMRINI
Pr. Said NAJAH
Pr. Aicha MAJDA
Pr. Khalid ZENKOUAR

Année Universitaire 2015-2016

Remerciements

Avant de présenter mon rapport du projet de fin d'études, je tiens à remercier tous ceux qui ont contribué, de près ou de loin, à sa réalisation. Ce travail est la concrétisation des deux années de formation à la ***Facultés des Sciences & Techniques FST*** de ***Fès*** au ***Labo Systèmes Intelligents & Applications LSIA***, c'est grâce aux connaissances acquises durant toutes ces années de formation que j'ai pu réaliser ce travail.

A cet effet, le minimum de justice impose que l'apport de chacun des acteurs soit reconnu ne serait-ce que par des remerciements :

J'adresse mes remerciements les plus sincères à Monsieur ***Saïd NAJAH*** professeur enseignant à la ***FST***, pour son encadrement, sa disponibilité et ses conseils fructueux qu'il m'a prodigué le long de mon projet.

J'adresse aussi mes remerciements à Monsieur ***Azzedine ZAHI***, professeur enseignant à la ***FST***, qui, malgré ses multiples engagements, a accepté de se mettre à mon service en dirigeant avec d'extrême magnanimité mes travaux.

Je remercie Monsieur ***Khalid ZENKOUAR,*** professeur enseignant à la ***FST***, pour ses directives pertinentes et l'aide qu'il m'a apporté afin d'accomplir ce travail.

Je remercie également les honorables membres du *Jury* Mesdames ***Loubna LAMRINI*** et ***Aicha MAJDA***, aussi Monsieur ***Khalid ZENKOUAR*** pour le temps qu'ils m'accordent afin d'évaluer mon travail de fin d'études.

J'adresse, par la même occasion, mes remerciements aux membres du labo ***LSIA***, particulièrement les doctorants ***Sofiane*** et ***Rachid*** qui n'ont hésité à partager leurs connaissances et leurs brillants conseils.

Je tiens à remercier, aussi, tout le corps enseignant de la faculté, pour leurs connaissances qu'il a bien voulu nous les partager.

Finalement, je ne peux me passer de remercier ***Imane BIYYA***, doctorante à *l'**Ecole Mohammedia d'Ingénieurs*** pour notre échange d'informations et idées.

Résumé

Dans le cadre de notre formation de ***MASTER*** à la ***F**acultés des **S**ciences et **T**echniques **FST*** de ***Fès***, nous sommes menés à élaborer un projet de fin d'études à notre deuxième année. Nous avons choisi un *PFE* de recherche au ***L**abo **S**ystème **I**ntelligents & **A**pplications **LSIA*** afin d'élargir nos connaissances scientifiques et d'enrichir nos aptitudes analytiques.

Ce projet de fin d'études porte sur la réalisation d'un système de détection et évitement des obstacles pour la locomotion des personnes déficientes visuelles. Notre motivation était de développer un système intelligent tout en réduisant les outils matériels. Certes, nous avons pu répondre à nos attentes par l'utilisation d'une seule caméra et la portabilité de notre programme.

Ce projet met en œuvre un large nombre de méthodes et de procédures, à savoir la segmentation par soustraction du background, la mixture de gaussienne, la méthode de *Lucas Kanade*, les contours de *Canny*, la logique floue, la programmation parallèle, la programmation *GPU*...

Le développement du code source est établit en se servant de multiples bibliothèques à savoir *OpenCV, fuzzylite, CUDA, OpenMP*….

Nous avons éprouvé notre système suivant divers environnements de tests et obtenu des résultats positifs en fonction de plusieurs paramètres. De même, nous avons étendu notre projet à une étude complémentaire pour dégager des résultats similaires au temps-réel.

Lors de notre étude, nous avons croisé quelques voies d'amélioration que nous laissons le soin d'y creuser aux prochains travaux de recherche.

Abstract

As part of our *MASTER* degree in the ***T**echnical **S**ciences **F**aculty of **Fez (FST),*** we are led to accomplish a graduation project in our second year. We have chosen to conduct a research project at ***I**ntelligent **S**ystem & **A**pplications **L**ab **LSIA*** in order to expand our scientific knowledge and enrich our analytical skills.

This graduation project consists in the realization of a system for detecting and avoiding obstacles for the locomotion of visually impaired people. Our motivation was to develop an intelligent system while reducing hardware tools. Certainly, we could meet our expectations through the use of a single camera and the portability of our program.

The project implements a large number of methods and procedures, namely segmentation by background subtraction, the mixture of Gaussians, the method of *Lucas Kanade*, *Canny* contours, fuzzy logic, parallel programming, *GPU* programming...

The development of the source code is established by using multiple libraries namely *OpenCV , fuzzylite, CUDA, OpenMP ...* .

We tested our system following various testing environments and achieved positive results based on several parameters. Likewise, we expanded our project to further study to generate similar results in the real-time.

During our study, we have met a few areas for improvement that we leave for next research works.

Sommaire

Liste des figures

Liste des tableaux

Liste des acronymes

2D : Espace de deux dimensions
3D : Espace de trois dimensions
API : Application Programing Interface
BSD : Berkeley Software Distribution
CPU : Central Processing Unit
CUDA : Compute Unified Device Architecture
DM : Disparity Map
ENVS : Electron-Neural Vision System
ETA : Electronic Travel Aids
FC : Fuzzy Controler
FLC : Fuzzy Logic Controller
GMM : Gaussian Mixture Model
GPGPU : General Purpose Graphics Processing Unit
GPS : Global Positioning System
GPU : Graphics Processing Unit
HVES : Heidelberger Vision Enhancement System
OMS : Organisation Mondial de la Santé
OpenCV : Open Computer Vision
OpenMp : Open Multi-Processing
RGB : Red Green Blue
SONAR : Sound Navigation and Ranging
SSD : Sous Système de Décision
SSV : Sous Système de Vision
TENS : Transcutaneous Electrical Nerve Stimulation
XML : EXtensible Markup Language

Introduction générale

La finalité ultime de la technologie est d'améliorer le niveau de vie et d'étendre la sphère contrôlable par les humains. De la médecine ancienne aux microchurigies, des grandes découvertes à la conquête de l'espace et des attelages en paire aux aéronefs. Toujours est-il que les récentes décennies profèrent une allure technologique escarpée, une large gamme de besoins latents et explicites est timidement abordée notamment les besoins des handicapés.

La déficience visuelle est un handicap sensoriel affectant la fonction visuelle. Elle peut être partielle, dite l'amblyopie, ou entière, dite la cécité. Maintes techniques ont été mises à la disposition des malvoyants afin de développer leur accommodation. Cependant, ces techniques sont limitées au Braille, aux aides visuelles optiques ou non optiques et aux aides électroniques qui font appel à des outils sophistiqués, et par conséquent à un coût élevé. Par ailleurs, la population concernée oscille entre les différentes classes sociales, des privilégiées aux démunies ; et plus encore, lesdites aides électroniques sont techniquement encombrantes.

Faute de ce qui est cité, cette surface technologique s'avère fertile pour développer de nouvelles inventions et peaufiner l'existant. Une surface qui nous a séduit à dégager une solution qui compromis l'objectif au coût. Et en vertu de notre formation, nous avons choisi de devancer le volet technique en réduisant les ressources matérielles nécessaires.

Notre projet consiste à réaliser un système de détection et d'évitement des obstacles pour une navigation autonome des malvoyants, un système qui n'est doté que d'une seule caméra et portable sur diverses plateformes matérielles notamment les systèmes embarqués et les Smartphones. Néanmoins, le choix de n'utiliser qu'une seule caméra puisse engendrer des contraintes techniques, vu que la projection plane de l'espace tridimensionnel rend impossible l'estimation de la profondeur de la scène, d'où l'introduction de la logique floue pour corriger ces entraves.

Notre système est composé de deux sous-systèmes. Le premier est désigné pas le sous-système de vision qui utilise différents algorithmes de traitement d'image. Ces algorithmes servent à segmenter une scène en pixels appartenant au chemin de la navigation et les obstacles présents sur ce chemin. Les positions des obs-

tacles sont estimées, et elles sont envoyées au second sous-système dit sous-système de décision. Il est implémenté en logique floue, et traite les imprécisions et les incertitudes pour trouver la direction et la vitesse de déplacement.

Pour expliquer l'approche adoptée, le présent mémoire est organisé en quatre chapitres, dans l'ordre suivant :

En premier lieu, nous allons formuler un nombre important des technologies mises à la disposition des personnes déficientes visuelles, nous allons présenter particulièrement les systèmes d'aide à la navigation, en ressortissant les avantages et les inconvénients de chacune.

En second lieu, nous allons présenter les différents algorithmes que nous avons utilisés. Nous présentons les différentes méthodes de segmentation implémentées dans notre approche, puis nous allons voir des notions de la logique floue essentielles pour comprendre l'approche que nous avons adoptée.

Ensuite, notre approche sera présentée en détails, nous allons voir l'architecture générale du système proposé, nous verrons en détails le sous-système de vision ainsi que le sous-système de décision.

Finalement dans le quatrième chapitre, nous verrons l'implémentation de l'approche présentée. Nous allons décrire les tests que nous avons menés et les critères utilisés pour évaluer la performance du système, avant de présenter les résultats obtenus.

Chapitre 1 : Généralités

I. Introduction

Plusieurs technologies sont mises à la disposition des personnes déficientes visuelles. Elles ont pour fonction principale l'aide des sujets cibles à accomplir leurs tâches quotidiennes. Ce chapitre formulera un nombre important desdites technologies particulièrement les systèmes d'aide à la navigation. Pour ce, nous y introduirons les chiffres soumis par les organismes intéressés avant de mettre l'accent sur les points forts et les limites de chacune.

Depuis vingt ans, l'organisation mondiale de la santé OMS, créé en 1948, et sise à Genève-Suisse avec 150 bureaux de pays et six bureaux régionaux [1] s'est adressée aux problématiques de cécité et de déficience visuelle. Il s'agit, en effet, d'un sujet de santé substantiel dont les chiffres parlent d'eux-mêmes. L'issue des recensements menés par l'OMS sur les quatre coins de la Terre peut être récapitulée comme suit :

- 285 millions de personnes dans le monde présentent une déficience visuelle, 39 millions d'entre elles sont aveugles et 246 millions présentent une baisse de l'acuité visuelle
- 90% de celles présentent une déficience visuelle vivent dans des pays à faible revenu.
- 82% des aveugles sont âgés de 50 ans et plus.

Par conséquent, une population mondiale considérable retrouve des difficultés majeures à accomplir les tâches quotidiennes. Malgré les efforts recourus à développer les moyens techniques spécifiques, les services offerts demeurent restreints, ceux notamment qui adressent la lecture. Cependant, les sujets déficients visuels adoptent toujours les solutions classiques, étant la canne et les chiens d'accompagnement, pour les fins de navigation.

II. Technologies en faveur des déficients visuels

A partir du 1960 et avec le développement technologique, des systèmes plus sophistiqués ont commencé à apparaitre, une vue générale sur ces systèmes permet de les composer en trois catégories [2] :

II.1. Renforcement de la vision (*Vision Enhancement*)

Il s'agit d'un ensemble de dispositifs destinés principalement aux personnes malvoyantes, autrement dit les personnes ayant une vision très restreinte (une acuité visuelle inférieure à 1/20 pour le meilleur œil après correction [3], sans présenter une déficience visuelle totale). Ces dispositifs assistent les sujets cibles dans des tâches quotidiennes spécifiques à savoir la lecture et l'utilisation de l'ordinateur. Ils peuvent êtres des outils optiques comme des lunettes équipées de loupes pour assister à la lecture, ou des loupes tenues à la main(*fig.2)*, sinon, ils se servent d'une caméra pour filmer la scène, traitent l'information, puis l'affichent sur un media d'affichage visuel adéquat *(fig.2)*.

a) **b)** **c)**

***Figure 1*: exemples de dispositifs pour le renforcement de la vision. a) Lecture avec une loupe. b) Une loupe portée à la main. c) HVES (*Heidelberger Vision Enhancement System*).**

II.2. Remplacement de la vision (*Vision Replacement*)

Comme son nom l'indique, il s'agit d'une gamme des technologies nécessitant des interventions chirurgicales pour intervertir le système oculaire humain. Ils consistent à afficher l'information directement sur le cortex visuel ou sur le nerf optique.

II.3. Substitution de la vision (*Vision Substitution*)

Comme les outils de renforcement de la vision requièrent un niveau de vision, quoiqu'il soit bas, les personnes déficientes visuelles, proprement dit les aveugles ne peuvent s'en servir. D'autre part, la catégorie concernant les dispositifs de remplacement de vision n'a pas encore atteint le niveau de maturité nécessaire pour être largement utilisé, en outre, les implants pareils s'avèrent chers. Une adaptation est faite en vertu des sujets déficients visuels, présentée sous la troisième catégorie (*Substitution de la vision*). D'ailleurs, cette catégorie est la plus répandue, étant donné la multitude des tâches, notamment la lecture et la locomotion, qu'elle admet aussi que son principe. Généralement, ses dispositifs acquièrent l'information ambiante grâce à des capteurs, les traduisent en langage

informatique, les traitent pour reproduire l'action à élaborer par le sujet sous forme auditive, tactile ou en combinant les deux. Une transformation sensorielle est alors effectuée, en substituant le sens oculaire par les capteurs et changeant la route de l'information qui s'effectuera via les sens d'ouïe et de toucher.

Dans le cadre de notre travail, nous nous orienterons vers la troisième catégorie des technologies visuelles, particulièrement les dispositifs d'aide à la locomotion (***E**lectronic **T**ravel **A**ids **ETA***).

III. Systèmes de navigation

Généralement la recherche liée à ce domaine corrèle fortement avec d'autres domaines à savoir l'assistance des chauffeurs dans les voitures modernes, ou encore la robotique autonome, à fin militaire ou industrielle. La recherche dans les domaines précités était poussée par les financements mis en faveur de la ***R**echerche **&** **D**éveloppement*, ainsi, plusieurs travaux de recherche orientés vers l'assistance à la conduite et l'auto parking, se sont avérés bénéfiques.

Ces systèmes destinés au même but, la navigation autonome, se distinguent suivant les spécificités de chaque cas d'utilisation, à titre d'exemple la vitesse de navigation et la tolérance d'erreurs.

Les ***ETA*** peuvent être catégorisés selon la forme de l'information acquise à partir de l'environnement extérieur, les moyens les plus utilisés sont : les sonars, les scanners laser, les cameras ou une combinaison de ces moyens.

III.1. Le sonar

Le sonar (***S**ound **Na**vigation and **R**anging*) : est un appareil qui utilise les propriétés particulières de la propagation du son pour détecter et localiser les objets, sa première apparition était durant la première guerre mondiale et il était utilisé pour la navigation maritime. L'utilisation des sonars a était inspiré du domaine biologique, où on parle de l'écholocalisation qui consiste à envoyer des sons et écouter leur écho pour localiser, et dans une moindre mesure identifier, les éléments d'un environnement. Elle est utilisée par certains animaux, notamment des chauves-souris et des cétacés.

À noter dans ce cadre, qu'il a était documenté [4] que certains aveugles utilisent l'écholocalisation pour localiser des obstacles tels que des murs, ces gens ont développé une certaine faculté pour produire des sons (par leurs mains, ou des sons phonétiques) et en se basant sur l'écho de ces sons qu'ils localisent les objets. Les auteurs dans [5] ont étudiés les fonctions au niveau neurologiques, et ont remarqué, que le traitement des échos se fait au niveau des régions du cerveau qui sont normalement consacrées à la vision et non à l'ouïe.

Le principe général des sonars se base sur les propriétés physiques des ondes acoustiques, le sonar émet des ondes ultrasoniques sur une fréquence donné par un émetteur, ces ondes seront éventuellement réfléchies par les obstacles et captées par le sonar qui peut déterminer la distance aux obstacles :

$$\boldsymbol{d = v \times t}$$

Avec v la vitesse du son et t la moitié du temps de réception.

III.2. Scanner Laser

Un scanner Laser mesure le positionnement d'un échantillonnage de points dans un système de coordonnées - un nuage de points - de la surface d'un sujet pour ensuite en extrapoler la forme à partir de leur répartition : ce procédé est appelé une reconstruction 3D. Si la couleur de chacun des points est analysée, alors celle de la surface peut également être reconstituée.

Des analogies existent entre un appareil photo et un scanner 3D. Les deux ont un champ de vision et ne peuvent voir ce qui est masqué, les deux technologies étant optiques. Si le premier capture les couleurs des surfaces dans son champ l'autre mesure son positionnement relatif par rapport à un échantillon de points des surfaces.

Pour la détection des obstacles, le scanner balaye le laser sur une scène en deux dimensions, et pour chaque pixel calcule le temps nécessaire au signal pour quitter le capteur, heurte l'obstacle et revient au capteur, puis construit une carte de profondeur (*Disparity Map*) pour la scène en face. Des algorithmes peuvent être utilisés pour détecter les discontinuités dans la carte pour éventuellement détecter des obstacles. Donné la vitesse de la lumière, la distance à chaque obstacle est systématiquement déduite.

III.3. Caméra

Malgré les différentes approches de détection des obstacles, la caméra reste le moyen le plus proche à la perception humaine, en fournissant des informations diverses sur l'environnement extérieur. Dans ce cadre on peut distinguer deux grandes catégories, la stéréo vision qui utilise deux caméras et les approche qui utilisent une seule caméra.

i. Stéréovision (stéréoscopie)

Pour "*voir*" un espace ou un objet en relief il est nécessaire d'acquérir une image de ce même espace ou objet mais sous deux angles de vue différents. Ce sont donc deux images qui sont produites et c'est la superposition de ces deux images qui crée une seule image en relief. C'est tout simplement le principe de la vision humaine : l'œil gauche regarde sous un angle, l'œil droit, décalé de l'œil gauche regarde sous un autre angle ; l'information est transmise au cerveau qui recompose une seule image en relief.

Pour pouvoir reproduire artificiellement la vision stéréo humaine deux conditions sont donc nécessaires. La première est de réaliser deux images d'un même espace sous deux angles de vue différents (appareil photo stéréo) mais également simuler la restitution en relief en remettant les deux yeux dans les conditions normales de la vision (le restituteur).

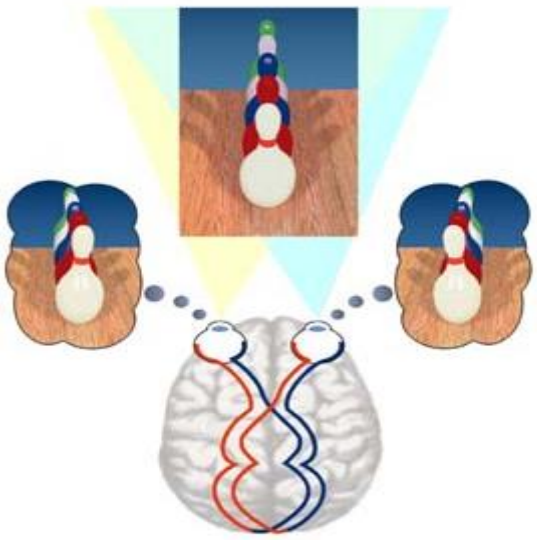


Figure 2 **: Principe de la Stéréovision**

En effet en plaçant deux caméras horizontalement séparées d'une distance définie, il est possible d'estimer la profondeur dans l'image, et ce en calculant la carte de disparité (***D****isparity* ***M****ap DM*), tout en essayant de faire une correspondance entre l'image prise par la caméra gauche, et la caméra droite.

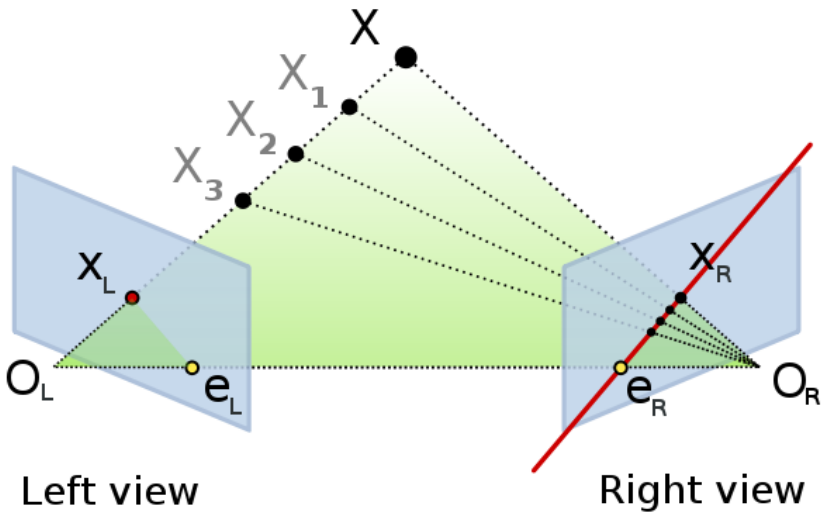


Figure 3 **: projection d'un point sur deux plans**

Considérons par exemple le point *X,* et soit sa projection sur le plan gauche X_L = *(u,v),* et X_R=*(p,q)* sa projection sur le plan droit. Notre objective est de déterminer la disparité (*disparity*) en calculant le module du vecteur entre le point *(u,v)* et le point *(p,q).*

La carte de disparité correspondant à une scène désigne une image en niveau de gris qui contient uniquement l'information sur les correspondances des points entre deux vues d'une même scène prises à deux endroits différents, cette correspondance est l'essort d'une procèdure de calibrage.

L'intuition dérrière la carte de disparité, est qu'une grande valeur de disparité pour un pixel donné, indique qu'il y'a un changement grossier dans la position du pixel d'une image à l'autre, cela est le résultant du fait que le pixel présente un objet proche à la caméra.

Ainsi en prenant l'image gauche d'une scène et l'image droite, on calcule la carte de disparité :

a) b) c)

Figure 4: **Carte de disparité (c) calculée à partir de deux vues différentes (a) et (b). Les pixels les plus foncés sont des pixels avec une petite disparité, présentant des objets loin de la scène.**

ii. Les approches utilisant une seule caméra

Alors que l'utilisation d'une seule caméra s'avère séduisante, par son coût et sa simplicité d'utilisation, les caméras sont largement disponibles sur des dispositifs indépendants ou encore implémentées sur des téléphones portables. Cependant son utilisation est strictement limitée par l'incapacité de retrouver la profondeur d'une scène 3D à partir d'une image 2D.

En fait, en projetant un espace tridimensionnel sur une surface bidimensionnelle, la surface du capteur, l'information sur la troisième dimension z est perdue.

Soit la projection géométrique d'une caméra sténopé (*pinhole camera*).

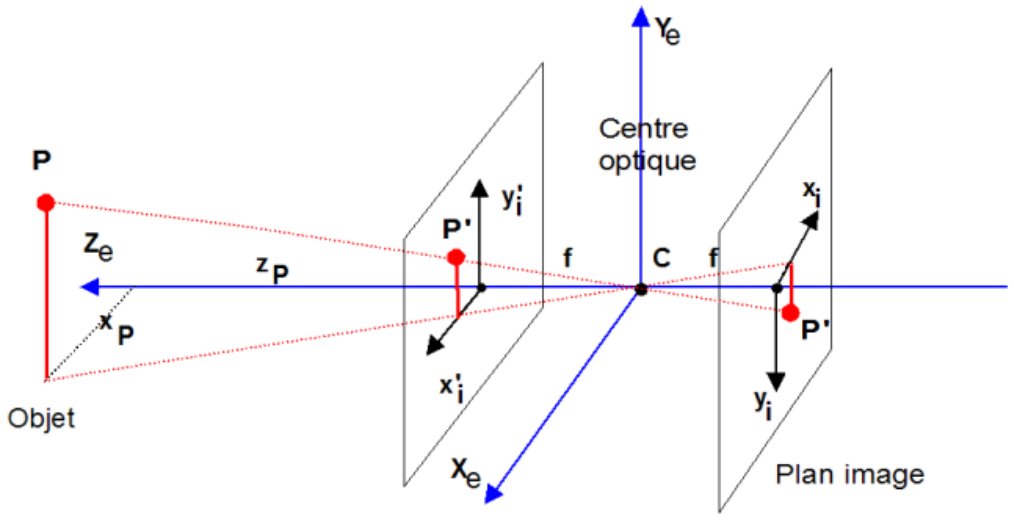


***Figure 5*:Projection géométrique**

Afin de simplifier la conception du problème nous proposons les hypothèses suivantes :

- Le repère de l'espace des objets (environnement extérieur) correspond à celui de la caméra.
- L'image se forme dans un plan (surface sensible du capteur) perpendiculaire à l'axe optique du système.
- L'origine du repère environnement extérieur (X_e, Y_e, Z_e) est supposée être en C, centre optique de la lentille (point nodal du plan principal objet pour un système optique).
- L'axe Z est confondu avec l'axe principal du système optique.
- Les axes x et y du plan image sont parallèles aux axes X et Y du repère objet et leur origine est sur l'axe Z.
- Le système optique est du type sténopé *(pinhole)*.

Alors, soit un objet P de coordonnées (x_e, y_e, z_e) dans le repère (X_e, Y_e, Z_e) et $(x_i y_i)$ les coordonnées de son image dans le plan du capteur. L'image se forme avec une ***inversion***, nous adoptons souvent comme repère caméra le repère symétrique par rapport au centre optique(x'_i, y'_i).

Nous remarquons la relation par triangles semblables:

$$\frac{x_i}{x_e} = \frac{f}{z_e}$$

De même

$$\frac{y_i}{y_e} = \frac{f}{z_e}$$

Nous en déduisons :

$$x_i = f\frac{x_e}{z_e}$$

Et

$$y_i = f\frac{y_e}{z_e}$$

On constate que les coordonnées de la projection P' de P dépendent de la profondeur z_p et que sa localisation $(x_p y_p)$ ne peut se faire à partir de $(x_{p'} y_{p'})$ que si $z_p$est connu, ce qui exige une connaissance à priori sur la scène.

IV. Systèmes développés

IV.1. *Navebelt*

Navebelt [6] est un système développé par *Borenstein* et ses collègues à l'université de *Michigan*. Le prototype utilise huit capteurs ultrasoniques, un ordinateur et des écouteurs. L'ordinateur reçoit l'information à partir des capteurs et crée une carte (un plan) pour chaque direction, calcule la distance au plus proche obstacle dans chaque direction, ensuite le système envoie un signal acoustique à l'utilisateur, indiquant la direction la plus « adéquate » pour la navigation.

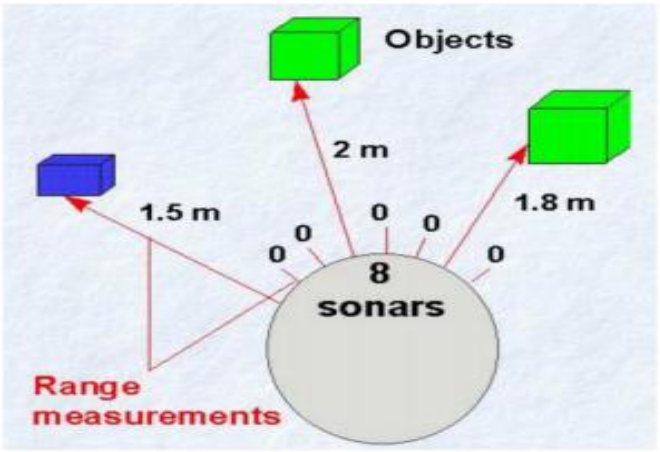


Figure 6 **: Système de *Navebelt* avec 8 capteurs pour présenter la distance à chaque obstacle**

Quelques limites de cette catégorie des systèmes (utilisant le sonar comme moyen d'acquisition de l'information), sont liées à la nature physique des ondes ultrasoniques, notamment le fait que les échos ne sont pas le produit ultime de la dis-

tance i.e. la nature physique de l'objet modifie la manière suivant laquelle l'onde est reflétée.

D'autre part la nature de l'information délivrée à l'utilisateur final, présente un deuxième inconvénient de ce système. ***Navebelt*** utilise des signaux distincts, pour indiquer la direction à suivre, cela exige une initiation des utilisateurs au produit pour savoir comment interpréter chaque signal.

IV.2. Electron-Neural Vision System *ENVS*

Développé par *Meers* et Ward à l'université de *Wallongong* en Australie [7], il a pour but de détecter et d'éviter des obstacles dans un environnement extérieur quelconque. Leur système est équipé d'un ensemble de dispositifs : des caméras, le *GPS* et une simulation électro-tactile.

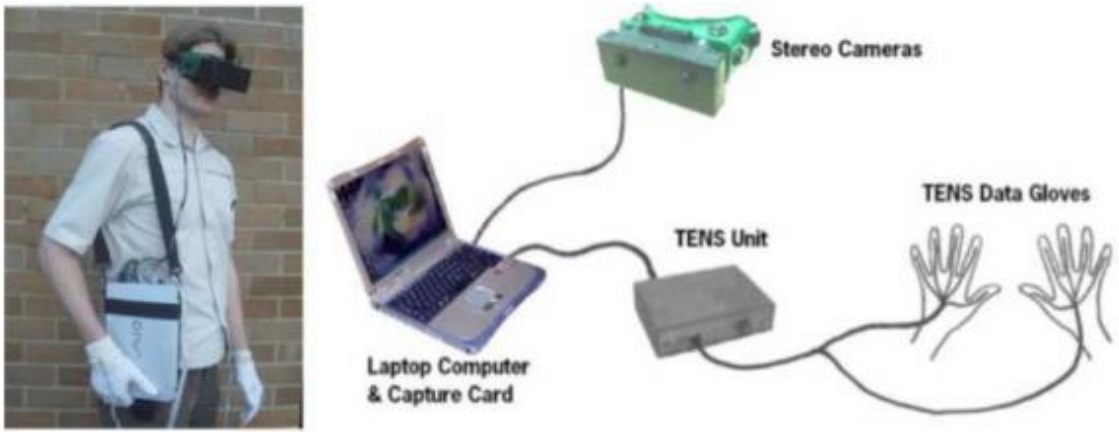


Figure 7 **: *ENVS* et ses composantes**

Le prototype (*fig.7*) est composé d'un casque équipé de deux caméras stéréo, une boussole numérique, un ordinateur portable avec des options *GPS*, une base de données des points de repère (*Landmark*) un dispositif (un microcontrôleur) appelé ***TENS*** (***T****ranscutaneous* ***E****lectrical* ***N****erve* ***S****timulation* ou stimulation nerveuse par impulsions électriques) et des gants spéciaux équipés de la technologie *TENS*.

Le système acquit l'information sur la scène avec les deux caméras, l'ordinateur calcule les cartes de dispersions pour la caméra de chaque région et y localise les obstacles, utilise ensuite la base de données des *Landmark* pour se récupérer à l'aide du *GPS*.

L'information est ensuite transformée vers les gants sous forme d'impulsions électriques qui stimulent les nerfs sur la peau des doigts, avec des électrodes placées sur les gants.

La main de l'utilisateur est supposée positionnée devant l'abdomen avec les doigts écartés, la force des stimulations est proportionnelle à la distance aux obstacles dans la direction de chaque doigt.

Le prototype était testé avec des gens aux yeux bandés, à l'extérieur de l'université, travail en temps réel (*15 trames/seconde*) et un apprentissage d'une heure minimum, les utilisateurs ont pu à l'aide de ce système de détecter des obstacles, les éviter et arriver à une destination prédéfinie.

V. Conclusion

Dans ce chapitre, nous avons vu les différents systèmes mis en place pour l'assistance des personnes déficientes visuelles. Nous avons mis l'accent en particulier sur les systèmes d'aide à la navigation. Nous avons exposé les différentes techniques utilisées dans la conception de ces systèmes. Finalement Nous avons donné l'exemple de deux de ces systèmes.

Chapitre 2 : Algorithmes utilisés et logique floue

I. Introduction

Dans ce chapitre nous allons présenter les différents algorithmes que nous avons utilisés. Nous présentons les différentes méthodes de segmentation implémentées dans notre approche, puis nous allons voir des notions de la logique floue, essentielles pour comprendre l'approche que nous avons adoptée.

II. Segmentation par soustraction du background

La soustraction du *background* est une approche répandue. Son idée consiste à calculer la « différence » entre l'image courante, et une image de référence, souvent appelée modèle du *background*.

Ce modèle doit être une représentation de la scène sans obstacles, et il doit être constamment mis à jours pour capturer les différents changements de luminance et de géométrie [8].

Considérons une séquence d'images *I* (*séquence vidéo*), avec un modèle de background *B*. Pour chaque pixel s d'une image à l'instant t, noté $I_{s,t}$. Nous y affectons un masque$\chi_t(s)$, de valeur 1 si le pixel est « suffisamment » loin du pixel correspondant dans le modèle (pixel ayant les même coordonnées *x, y)* et 0 sinon, la formulation mathématique est comme suit :

$$\chi_t(s) = \begin{cases} 1 & si \quad d(I_{s,t}, B_s) > \tau \\ 0 & sinon \end{cases}$$

Avec d est la distance entre le pixel s de l'image t, et le pixel B_s, alors que τ est un seuil à définir.

L'ensemble des méthodes de soustraction du background se diffère par la construction du modèle du background et la métrique de distance utilisée pour d.

II.1. Une méthode de soustraction basique

La méthode la plus simple que nous pouvons utiliser pour la soustraction consiste à prendre une image du background sans obstacle, puis y soustraire chaque trame de la séquence, pour voir les pixels qui sont loin de ce modèle.

Le modèle peut être mis à jour par la formule :

$$\boldsymbol{B}_{s,t+1} = (1-\alpha)\boldsymbol{B}_{s,t} + \alpha.\boldsymbol{I}_{s,t}$$

Avec α une constante appelée constante de mise-à-jour, sa valeur varie entre *0* et *1*.

Ensuite le masque peut être calculé par l'une des métriques suivantes :

$$\begin{aligned} d_0 &= |I_{s,t} - B_{s,t}| \\ d_1 &= |I^R_{s,t} - B^R_{s,t}| + |I^G_{s,t} - B^G_{s,t}| + |I^B_{s,t} - B^B_{s,t}| \\ d_2 &= (I^R_{s,t} - B^R_{s,t})^2 + (I^G_{s,t} - B^G_{s,t})^2 \\ &\quad + (I^B_{s,t} - B^B_{s,t})^2 \\ d_\infty &= \max\{|I^R_{s,t} - B^R_{s,t}|, |I^G_{s,t} - B^G_{s,t}|, |I^B_{s,t} - B^B_{s,t}|\} \end{aligned}$$

Où R, G et B notent respectivement les trois canaux des couleurs : rouge, vert et bleu, à noter que la distance d_0 opère uniquement sur les images au niveau de gris.

Cette méthode comme son nom l'indique, est assez basique, d'abord, il exige une connaissance à priori de la scène pour la construction du modèle du *background*, et d'un autre côté la méthode utilise une approche locale pour la soustraction. Puisqu'elle opère sur chaque pixel indépendamment, elle omet l'information sur les pixels voisins, ce qui la rend sensible au bruit.

Pour remédier à ces problèmes, plusieurs approches ont été proposées dans la littérature, nous exposons dans ce travail la méthode d'apprentissage du background par une mixture de gaussiennes.

II.2. Apprentissage du background

i. Estimation par une Gaussienne

Plusieurs auteurs [11] ont proposé de présenter chaque pixel du modèle par une fonction de densité gaussienne qui peut tolérer un certain niveau de bruit, chaque pixel alors sera présenté par une distribution gaussienne.

$$\eta(\mu_{s,t}, \sigma_{s,t})$$

Avec $\mu_{s,t}$ la moyenne de l'intensité, et $\Sigma_{s,t}$ la variance du pixel s à l'instant t.

Ces paramètres sont déduits à partir d'un nombre de trames d'apprentissage [9].

La distance utilisée est celle du *log likelihood* :

$$d_G = \frac{1}{2} log((2\pi)^3 |\Sigma_{s,t}|) + \frac{1}{2}(I_{s,t} - \mu_{s,t})\Sigma_{s,t}^{-1}(I_{s,t} - \mu_{s,t})^T$$

Ou bien la distance dite de *Mahalanobis* :

$$d_M = |I_{s,t} - \mu_{s,t}| \, \Sigma_{s,t}^{-1} \, |I_{s,t} - \mu_{s,t}|^T$$

Avec $I_{s,t}$ et $\mu_{s,t}$ des vecteurs *RGB* et $\Sigma_{s,t}$ la matrice de covariance, pour prendre en considérations des variations subtiles qui peuvent affecter un pixel *s* au cours du temps *t*.

Ensuite la moyenne et la covariance du modèle peuvent être mises à jour par les formules suivantes :

$$\mu_{s,t+1} = (1 - \alpha).\mu_{s,t} + \alpha.I_{s,t}$$

$$\Sigma_{s,t+1} = (1 - \alpha).\Sigma_{s,t} + \alpha.(I_{s,t} - \mu_{s,t})(I_{s,t} - \mu_{s,t})^T$$

ii. Mixture de gaussiennes (*GMM*)

Plus souvent les *background*s ne sont pas statiques, un pixel du *background* peut prendre plusieurs valeurs au cours du temps, par exemple des feuilles d'un arbre secouées par le vent. Dans ce cas à un moment donné *t*, un pixel peut prendre la couleur d'une feuille de l'arbre, et à un moment *t+1* il prend la couleur du ciel. L'objectif est de présenter les changements rapides et répétitifs dans le modèle.

Pour répondre à ces attentes, nous construisons un modèle par des densités multimodales. *Stauffer* et *Grimson* [10] ont proposé de présenter chaque pixel par *K*

Gaussiennes, où la probabilité d'occurrence d'une couleur à un pixel donné est présentée par :

$$P(I_{s,t}) = \sum_{i=1}^{K} \omega_{i,s,t}.\eta(I_{s,t}, \mu_{i,s,t}, \Sigma_{i,s,t})$$

Avec $\eta(I_{s,t}, \mu_{i,s,t}, \Sigma_{i,s,t})$ la distribution de l'*ï-éme* gaussienne. Et $\omega_{i,s,t}$ son poids.

Les pixels du *background* qui sont comparables aux nouveaux pixels au moment *t* (dans un intervalle de 2.5 écart type) seront mis à jour par les formules suivantes :

$$\omega_{i,s,t} = (1-\alpha)\omega_{i,s,t} + \alpha$$
$$\mu_{i,s,t} = (1-\rho).\mu_{i,s,t-1} + \rho.I_{i,s,t}$$
$$\sigma^2_{i,s,t} = (1-\rho).\sigma^2_{i,s,t-1} + \rho.(I_{i,s,t} - \mu_{i,s,t})^2$$

Avec α le taux d'apprentissage défini par l'utilisateur, et ρ un autre taux d'apprentissage défini comme suit :

$$\rho = \alpha\eta(I_{s,t}, \mu_{i,s,t}, \Sigma_{i,s,t})$$

Les paramètres μ et σ des gaussiennes non comparables, ne seront pas modifiés mais leurs poids seront réduits par :

$$\omega_{i,s,t} = (1-\alpha)\omega_{i,s,t-1}$$

Jusqu'à que leurs poids s'annulent. A chaque fois qu'un pixel $I_{s,t}$ n'est présenté par aucune gaussienne dans le background, la gaussienne avec le poids le plus faible sera remplacé par une nouvelle gaussienne, de moyenne $I_{s,t}$, une large variance initiale σ_0 et un petit poidsω_0. Une fois toutes les gaussiennes sont mises à jour, les *K* poids $\omega_{i,s,t}$ sont normalisés et leur somme est mise à 1. Ensuite les *K* distributions sont ordonnées selon une valeur de fitness$\omega_{i,s,t}/\sigma_{i,s,t}$, et seulement *H* distributions seront gardés comme représentants du modèle :

$$H = \underset{h}{argmin}(\sum_{i=1}^{h} \omega_i > \tau)$$

Avec τ un seuil. Les pixels qui sont à plus de *2.5* écarts types de toutes les *H* distributions seront considérés non appartenant au modèle.

III. Segmentation par mouvement : optical Flow

Le flux optique (ou défilement visuel) est le mouvement apparent des objets, surfaces et contours d'une scène visuelle, causé par le mouvement relatif entre un observateur (l'œil ou une caméra) et la scène.

Le concept de flux optique a été étudié dans les années 1940 et des travaux ont été publiés dans *American psychologist* par *James J. Gibson* [11] [12] . Les applications du flux optique telles que la détection de mouvement, la segmentation d'objets et la mesure de la disparité stéréoscopique utilisent le mouvement des textures de l'objet et de ses bords.

Dans le traitement d'image, entre deux trames consécutives d'une séquence vidéo, nous cherchons à trouver, pour chaque pixel de la trame T_1, le pixel correspondant dans la trame T_2, et nous cherchons à calculer le vecteur de déplacement, qui présente la direction et la vitesse relative du mouvement (*fig.8*), ainsi nous obtenons un champs de vecteurs pour chaque trame qui nous permet de distinguer les objets statiques (généralement le *background*) des objets en mouvement (généralement des obstacles).

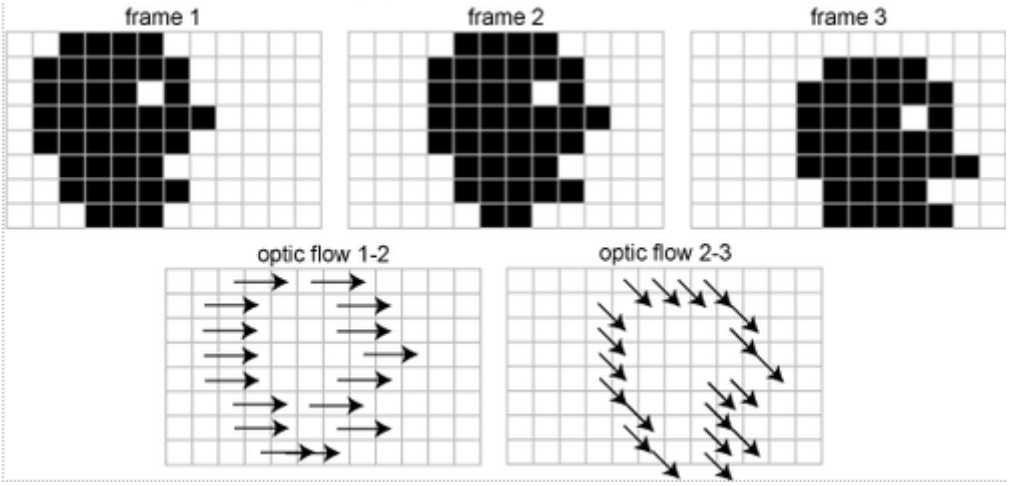


Figure 8 : **trois trames consécutives présentant le mouvement de la silhouette d'une tête, et les vecteurs de flux correspondants**

Le calcul du flux optique s'avère très lourd en termes du temps de calcul, puisque nous devons d'abord chercher les pixels qui ont des correspondants de trame en trame, puis calculer leurs vecteurs de mouvement. Différentes méthodes ont été proposées pour résoudre ce problème dans un temps de calcule raisonnable, nous présentons ici la méthode que nous avons utilisé celle de ***Lucas-Kanade*** [13] [14] [15].

Méthode de *Lucas-Kanade*

La méthode de *Lucas–Kanade* est une méthode différentielle utilisée pour l'estimation du flux optique. Cette méthode a été développée par *Bruce D. Lucas* et *Takeo Kanade*. Elle suppose que le flux est essentiellement constant dans un voisinage local du pixel considéré, et résout l'équation du flot optique pour tous les pixels dans ce voisinage par la méthode des moindres carrés.

La méthode de *Lucas-Kanade* suppose que le déplacement d'un point de l'image entre deux instants consécutifs soit petit et approximativement constant dans un voisinage du point p. L'équation du flux optique peut être supposée vraie pour tous les pixels dans une fenêtre centrée au point p (les pixels dans une fenêtre sont supposés avoir le même mouvement). Le vecteur de vitesse local *(Vx,Vy)* doit satisfaire :

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1) &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2) &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n) &= -I_t(q_n) \end{aligned}$$

Avec $q_1, q_2, \ldots, q_n$ sont les pixels à l'intérieur de la fenêtre, généralement *n*=25 (une fenêtre de 5 sur 5), $I_x(q_i), I_y(q_i), I_t(q_i)$ sont les dérivées partielles de l'image *I*, par rapport à la position *x, y* et le temps *t*.

Ces équations peuvent être écrite sous forme matricielle $Av = b$, avec :

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \ldots & \ldots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}$$

$$V = \begin{bmatrix} V_x \\ V_y \end{bmatrix}$$

$$b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

Ce système à plus d'équations que d'inconnus, i.e. *25* équations pour chaque pixel, le système est alors surdéterminé. La méthode de *Lucas-Kanade* résout la situation par la méthode des moindres carrés.

Alors nous calculons :

$$A^T Av = A^T b$$

Ou $\quad \mathrm{v} = (A^T A)^{-1} A^T b$

Où A^T, est la matrice transposée de A.

Nous calculons alors :

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix}$$

IV. Segmentation par contours

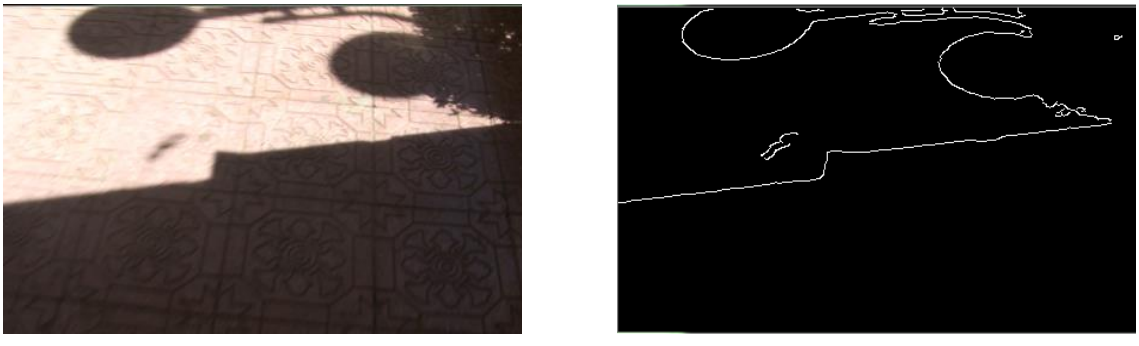
La détection de contours est une étape préliminaire à de nombreuses applications de l'analyse d'images. Les contours constituent en effet des indices riches, au même titre que les points d'intérêts, pour toute interprétation ultérieure de l'image. Les contours dans une image proviennent des :

- Discontinuités de la fonction de réflectance (texture, ombre),
- Discontinuités de profondeur (bords de l'objet),

Les contours sont aussi caractérisés par des discontinuités de la fonction d'intensité dans les images.

La détection de contours dans les images a débuté de façon extrêmement empirique par des opérateurs locaux qui, soit estimaient un gradient, soit convoluaient l'image par des masques caractéristiques des contours [*Haralick* et *Shapiro*, 1985]. Dans les années 80, des approches plus systématiques ont été mises en place par *Marr* [*Marr* et *Hildreth*, 1980], puis Canny [*Canny*, 1986], pour obtenir des contours plus significatifs.

Cette méthode détecte les changements de luminosité les plus subtiles dans une image, elle amène souvent à une « sur détection » des obstacles qui sont en réalité des régions du background (détection des ombres par exemple *(fig.9)*). L'utilisation de cette méthode a pour rôle d'introduire un biais qui favorise la détection.



a) **b)**

Figure 9 **: Méthode de Canny pour la détection d'obstacles.**

Les ombres sont détectés comme étant des obstacles : il s'agit d'une « sur détection ».

Contours de Canny

Le filtre de *Canny* (ou détecteur de *Canny*) (1986) [16] [17]est utilisé en traitement d'images pour la détection des contours. L'auteur l'a conçu pour être optimal suivant trois critères clairement explicités :

- Bonne détection : faible taux d'erreur dans la signalisation des contours,
- Bonne localisation : minimisation des distances entre les contours détectés et les contours réels,
- Clarté de la réponse : une seule réponse par contour et pas de faux positifs

Les étapes de détection de contours pour un filtre de Canny sont comme la suite :

i. Réduction du bruit

La première étape est de réduire le bruit de l'image originale avant d'en détecter les contours. Ceci permet d'éliminer les pixels isolés qui pourraient induire de fortes réponses lors du calcul du gradient, conduisant ainsi à de faux positifs.

Un filtrage gaussien 2D est utilisé, dont l'opérateur de convolution est le suivant :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

ii. Gradient d'intensité

Après le filtrage, l'étape suivante est d'appliquer un gradient qui retourne l'intensité des contours. L'opérateur utilisé permet de calculer le gradient suivant les directions *X* et *Y*, il est composé de deux masques de convolution, un de dimension 3×1 et l'autre 1×3 :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad ; \quad G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

La valeur du gradient à un point est approximée par la formule :

$$|G| = |G_x| + |G_y|$$

iii. Direction des contours

Les orientations des contours sont déterminées par la formule :

$$\theta = \pm \arctan\left(\frac{G_y}{G_x}\right)$$

Nous obtenons finalement une carte des gradients d'intensité en chaque point de l'image accompagnée des directions des contours.

iv. Suppression des non-maxima

La carte des gradients obtenue précédemment fournit une intensité en chaque point de l'image. Une forte intensité indique une forte probabilité de présence d'un contour. Toutefois, cette intensité ne suffit pas à décider si un point correspond à un contour ou non. Seuls les points correspondant à des maximas locaux sont considérés comme correspondants à des contours, et sont conservés pour la prochaine étape de la détection.

Un maximum local est présent sur les extrema du gradient, c'est-à-dire là où sa dérivée s'annule.

v. Seuillage des contours

La différenciation des contours sur la carte générée se fait par seuillage à hystérésis. Cela nécessite deux seuils, un haut et un bas, qui seront comparés à l'intensité du gradient de chaque point. Le critère de décision est le suivant. Pour chaque point, si l'intensité de son gradient est:

- Inférieur au seuil bas, le point est rejeté ;
- Supérieur au seuil haut, le point est accepté comme formant un contour ;
- Entre le seuil bas et le seuil haut, le point est accepté s'il est connecté à un point déjà accepté.

V. Logique floue

L'imprécision des données fournies par le sous-système de vision, due à l'incertitude sur la position de l'obstacle et les mouvements de la caméra, nous incite à utiliser un système de décision en logique floue. Ce choix nous approche du raisonnement humain et nous permet de manipuler l'incertitude de nos informations.

Nous allons présenter une brève introduction à la logique floue, qui permettra au lecteur non initié à cette théorie, d'acquérir les notions nécessaires pour comprendre les termes utilisés par la suite. Il est à noter qu'il ne s'agit pas ici de présenter les différents aspects de cette théorie, puisqu'ils sont nombreux et parfois s'avèrent complexes.

V.1. Définition

La logique floue est une extension de la logique booléenne créée par *Lotfi Zadeh* en 1965 en se basant sur sa théorie mathématique des ensembles flous,

qui est une généralisation de la théorie des ensembles classiques [18] [19]. En introduisant la notion de degré dans la vérification d'une condition, permettant ainsi à une condition d'être dans un autre état que vrai ou faux, la logique floue confère une flexibilité très appréciable aux raisonnements qui l'utilisent, ce qui rend possible la prise en compte des imprécisions et des incertitudes.

V.2. Eléments de base de la logique floue

Les éléments de base de la logique floue sont :

- Les variables linguistiques
- Les fonctions d'appartenance
- Les déductions aux inférences

i. Variables linguistiques

La description d'une certaine situation, d'un phénomène ou d'un procédé contient en général des qualificatifs flous tels que :

- Peu, beaucoup, énormément
- Rarement, fréquemment, souvent
- Froid, tiède, chaud
- Petit, moyen, grand
- etc.....

Exemple : la variable linguistique « température » peut appartenir aux ensembles flous « froid », « tiède » ou « chaud ».

ii. Fonctions d'appartenance

Au lieu d'appartenir à l'ensemble « vrai » ou à l'ensemble « faux » de la logique binaire traditionnelle, la logique floue admet des degrés d'appartenance à un ensemble donné. Le degré d'appartenance à un ensemble flou est matérialisé par un nombre compris entre *0* et *1*. Une valeur précise de la fonction d'appartenance liée à une valeur de la variable est notée μ et appelée « *facteur d'appartenance* ».

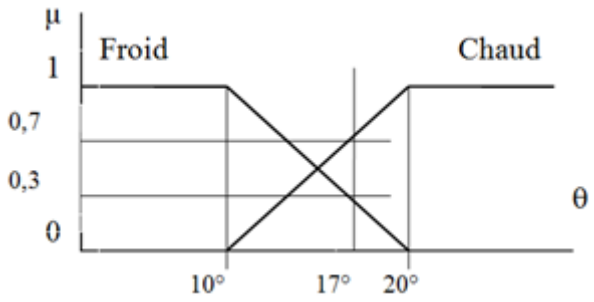


Figure 10 : **Exemple de fonction d'appartenance**

D'après ce graphique, nous pouvons constater que pour une valeur $\theta = 17°$, le facteur d'appartenance à l'ensemble « froid » vaut $\mu froid=0,3$ et le facteur d'appartenance à l'ensemble « chaud » vaut $\mu chaud=0,7$.

Les fonctions d'appartenance peuvent théoriquement prendre n'importe quelle forme. Toutefois, elles sont souvent définies par des segments de droites, et dites « *linéaires par morceaux* ».

iii. Déductions aux inférences

Plusieurs valeurs de variables linguistiques sont liées entre elles par des règles et permettent de tirer des conclusions.

Les règles peuvent alors être exprimées sous la forme générale :

Si *condition 1* ***alors*** *action 1 ;*

Si *condition 2* ***alors*** *action 2 ;*

Si .....

⋮

Si *condition n* ***alors*** *action n.*

Les conditions peuvent dépendre de plusieurs variables liées entre elles par des opérateurs *OU* ou *ET*.

Exemple : ***Si*** température froide ***ET*** hygrométrie importante ***alors*** ouvrir la vanne d'admission d'air chaud.

V.3. Composantes d'un contrôleur flou

Le schéma présentant les différentes composantes d'un système flou, est comme suit:

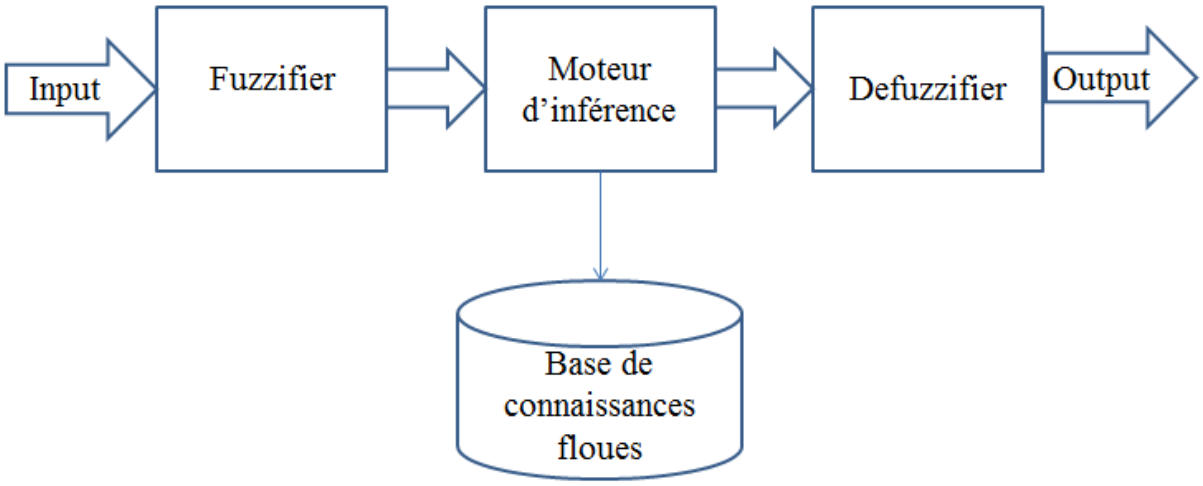


Figure 11 **: schéma synoptique d'un contrôleur flou**

***Fuzzifier* :** Prend des valeurs nettes en entrées et les transforme en des valeurs linguistiques, par les fonctions d'appartenances définies auparavant.

Base de connaissances : contient l'ensemble des règles définit par l'expert sous la forme **Si** (prémisses) **Alors** (conclusion).

***Moteur d'inférence* :** Activation des règles appropriés en fonctions des variables d'entrées et leur degrés d'appartenance.

***Defuzzifier* :** Transforme les valeurs floues obtenues par le moteur d'inférence en des valeurs nettes, en utilisant les fonctions d'appartenance.

VI. Conclusion

Dans ce chapitre nous avons exposé les différents algorithmes que nous avons utilisés dans notre approche.

Trois algorithmes que nous utilisons pour la segmentation, ont été détaillés, à savoir la soustraction du *background* par une mixture de gaussiennes, le flux de mouvement et la détection des contours par la méthode de *Canny*.

Ensuite nous avons présenté quelques notions de la logique floue. Cela avait pour but d'introduire le lecteur à cette théorie qui a été utilisée pour l'implémentation du système de décision dans notre projet.

Chapitre 3 : Approche adoptée

I. Introduction

Dans ce chapitre notre approche sera présentée en détails, une approche qui se distingue par l'utilisation d'une seule caméra pour acquérir l'information de l'environnement .D'ailleurs, nous avons vu que cela limite notre capacité à estimer la profondeur de la scène, ce qui implique directement d'éliminer toute notion de hauteur ou de distance, mais d'un autre côté, l'utilisation d'une seule caméra présente une multitude d'avantages : D'abord le faible coût, ensuite la portabilité du système, autrement dit son aptitude à être implémenté sur un système embarqué équipé d'une caméra, ou sur un téléphone mobile.

Pour combler les différentes lacunes liées à l'utilisation d'une seule caméra, nous faisons appel à un système de contrôle en logique floue, ce choix nous permet de prendre une décision à partir d'un ensemble de données incertaines ou/et imprécises.

Dans le reste du chapitre nous allons voir l'architecture générale du système proposé, nous verrons en détails le sous-système de vision ainsi que le sous-système de décision.

II. Hypothèses préétablies

Un ensemble de suppositions a été établi sur la nature de l'environnement et la position de la caméra. Bien que ces suppositions semblent restreindre l'utilisation du système, elles nous ont permis de développer une plateforme dans des conditions relativement « idéales », sur laquelle nous pouvons tester la faisabilité du système et d'obtenir des résultats favorables et encourageants. Ces suppositions sont comme suit:

i. Le chemin de navigation est plat, ainsi, des scènes incluant des fosses ou des escaliers ne seront pas traités.

ii. Les obstacles éventuellement présents sur une scène sont mis à terre, la position des obstacles en dessus de la terre seront généralement mal-estimés.

iii. La caméra est supposée fixée dans une position connue. Cette position sera fixée à *1 mètre* de hauteur.

III. Architecture générale du système

Tout d'abord nous commençons par introduire l'architecture générale du système. En effet, le système peut être divisé en deux parties majeures. Le premier est un sous-système de vision (*SSV*) qui traite la séquence vidéo trame par trame, en la redimensionnant, puis segmente le background du foreground et estime la position de chaque obstacle. Le second est un sous-système de décision (*SSD*) implémenté en logique floue, qui reçoit les positions des obstacles et renvoie la direction à suivre et la vitesse du mouvement. Ce qui peut être schématisé comme suite :

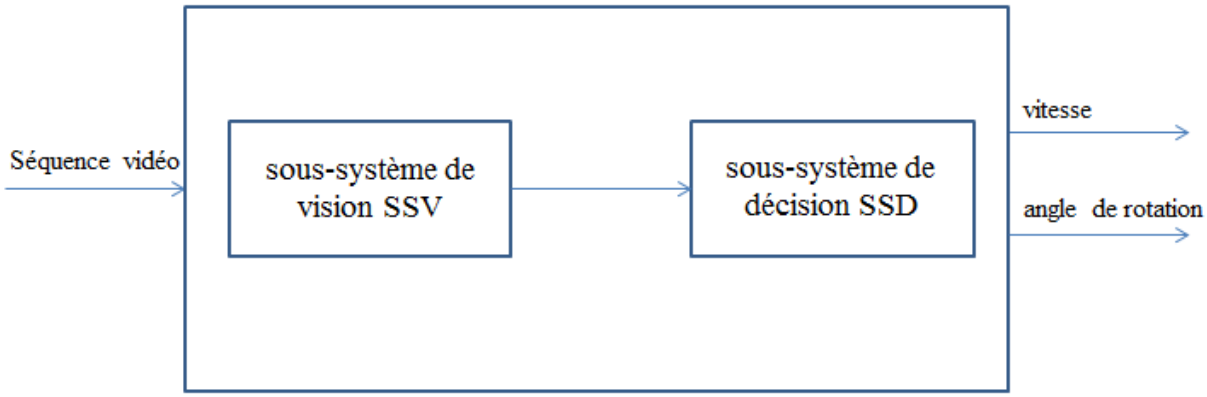


Figure 12 : **schéma synoptique de l'architecture générale du système.**

Dans la suite de ce chapitre nous allons décortiquer chaque partie du système, en expliquant les différents algorithmes utilisés et les enjeux à lesquelles nous devions répondre.

III.1. Sous-système de vision

Le sous-système de vision est constitué d'un ensemble des étapes que nous allons exposer une par une dans cette partie, nous allons présenter les algorithmes utilisés et les différents traitements mis en jeu :

i. Prétraitement

Après avoir acquis la vidéo de la scène, la première étape consiste à redimensionner chaque trame pour réduire ses dimensions. Nous réduisons la taille

de la trame tout en respectant le compromis entre la richesse d'information fournie par des grands dimensions et la réduction du temps de calcule donnée par des petites dimensions, surtout que nous travaillons sur une tâche qui exige le temps réel, alors nous avons choisi une dimension standard de *240*180* pixels.

ii. Segmentation

La détection des obstacles sur un chemin, peut être traitée comme un problème de segmentation, en vue de distinguer les pixels qui appartiennent au chemin, de ceux qui ne sont pas, *i.e.* qui présentent un obstacle.

Mais nous devons tout d'abord définir l'appartenance dans notre cas, et les critères selon lesquelles nous allons qualifier un pixel comme appartenant au chemin ou non, car les chemins peuvent être diversifiés et leurs textures peuvent être compliquées. Toutefois l'indice de couleur seul ne peut pas refléter nécessairement la présence d'un obstacle, *i.e.* l'obstacle peut avoir la même couleur que le chemin, ou encore, le chemin n'est pas nécessairement d'une couleur unifiée et peut contenir des textures complexes, et les ombres peuvent être facilement confus pour des obstacles. La seule distinction légitime –à notre avis- qui peut différencier un chemin des obstacles est en termes de profondeur. Les obstacles ont généralement une hauteur (qu'elle soit positive ou négative) qui les distingue du chemin traversable, or cette hauteur est perdue dans la projection du monde réel tridimensionnel sur une image bidimensionnelle.

Pour ce faire, nous avons utilisé une combinaison de trois méthodes de segmentation, qui sont expliquées dans le deuxième chapitre, à savoir : La segmentation par soustraction du background, la segmentation à base de mouvement et la segmentation par détection de contours.

iii. Post-Traitement

Après l'étape de segmentation, nous avons pour chaque trame de la séquence vidéo, trois masques (un masque pour chaque méthode) qui sont des images binaires où les pixels noirs sont ceux qui appartiennent au *background*, alors que les pixels blancs présentent des obstacles éventuels. Pour chaque masque, nous y appliquons des opérations morphologiques, principalement l'érosion, pour réduire les pixels qui n'appartiennent pas à leurs voisinages (pixels non connectés) et qui sont dû généralement au bruit

vi. Division de l'image

Marr [20] a défini la vision par ordinateur comme : « *le processus qui permet -étant donné un ensemble d'images- de créer une représentation complète et précise de la scène et ses propriétés* ». Cette définition est connue sous le nom de la « *vision générale* » parce que les informations extraites à partir de la scène doivent être aussi générales que possibles.

Or, la reconstruction complète (la plus générale) d'une scène nécessite une puissance de calcul considérable, et souvent, pour certaines tâches, il n'est pas nécessaire d'acquérir toutes les informations sur une scène donnée, mais juste ce qui est nécessaire pour le problème à résoudre. En vision biologique par exemple, les

systèmes de vision des fourmis et les abeilles acquiescent la direction aux endroits importants à partir du phénomène de la polarisation du ciel bleu [21]. Ils emploient les propriétés de leur environnement pour trouver leur chemin, cela suggère que *« la perception est intimement liée à la physiologie du sujet et les tâches qu'il doit accomplir »* [22].

Dans ce travail, nous ne cherchons pas à trouver la position exacte des obstacles dans le monde réel, leurs formes ou les relations qui peuvent exister entre ces objets, plutôt nous cherchons seulement à les éviter et à trouver les régions les moins encombrées d'obstacles.

De cette perspective, et inspirés de [12] [23] [24], nous proposons de diviser l'image en cinq régions, chaque région sera présentée par une barre et fait correspondance avec une région dans le champ de vue de la caméra.

Utilisons l'hypothèse 3 (caméra fixée dans une position connue), nous pouvons ainsi établir une relation empiriquement déduite entre la géométrie de la scène et celle de son image 2D. En prenant l'image d'une scène plane ne contenant pas d'obstacles, il est possible de présenter les dimensions de chaque région en termes de pixels.

Or, si le champ de vision contient des obstacles, nous ne pourrons pas faire de correspondance en termes de dimensions entre le monde réel et son image bidimensionnelle, cela revient au fait que les pixels d'un obstacle vont « *cacher* » ceux du *background*. Cela est résolu par le fait qu'on cherche la position du ***premier*** obstacle, et que distance entre la caméra et le premier obstacle appartient nécessairement au background.

Ainsi avec la caméra fixée dans une position préétablie, nous avons calculé les dimensions suivantes :

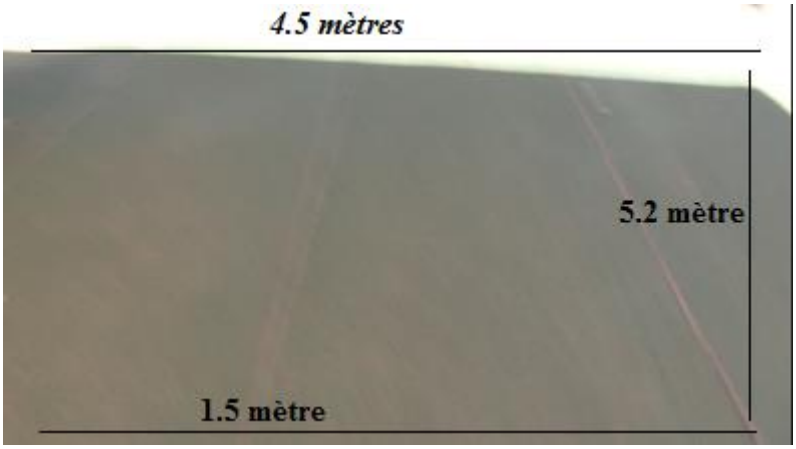


Figure 13 **: mesures prises sur une trame**

Ensuite nous avons défini les cinq régions suivantes :

Tableau 1: Régions définies pour le système et leurs variables linguistiques

Région	*Variable linguistique correspondante*
A gauche de l'utilisateur	Extrême gauche
A gauche de l'utilisateur	Gauche
En face de l'utilisateur	Centre
A droite de l'utilisateur	Extrême droite
A droite de l'utilisateur	Droite

Chaque région a une largeur maximale d'environ un mètre, et elle peut être considérée comme un chemin franchissable par l'utilisateur.

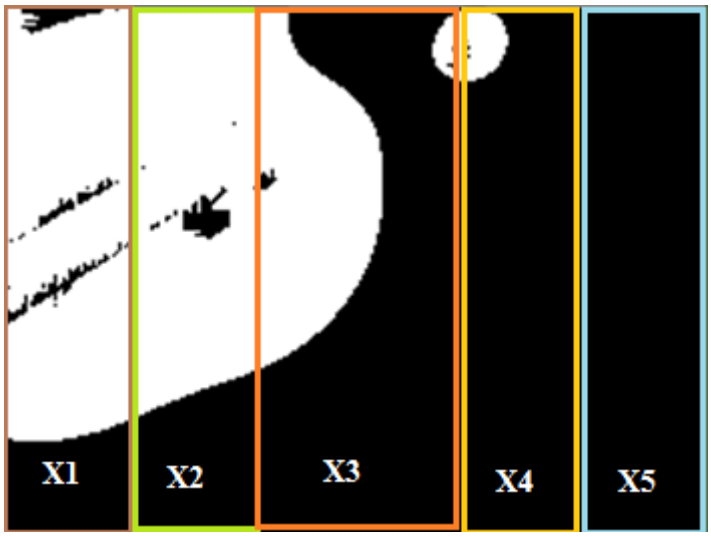


Figure 14 **: Division de l'image**

Pour le nombre de barres choisi, nous avons divisé les régions gauche et droite en des sous-régions. Cette division a pour but d'affiner la granularité d'analyse des régions gauche et droite. En effet, nous définissons un indice appelé « *indice de traversabilité* » pour la région gauche et la région droite, il est déduit à partir de leurs sous régions et donne une information globale sur chaque région (plus de détails sur cet indice et comment il est calculé dans la partie du système de décision).

iv. Régions connectées

Pour chaque barre, nous cherchons la position du ***premier*** obstacle rencontré, nous procédons par scanner l'image du bas vers le haut en cherchant un obstacle. Un obstacle est défini comme une région de pixels blancs qui sont connectés par une connectivité au moins de *4*, *i.e.* tout rassemblement d'au moins *4* pixels est un obstacle. Chaque obstacle est présenté par un rectangle, le plus pe-

tit rectangle contenant tous les pixels de l'obstacle, et la position de l'obstacle est la position suivant l'axe y du côté inferieur de ce rectangle.

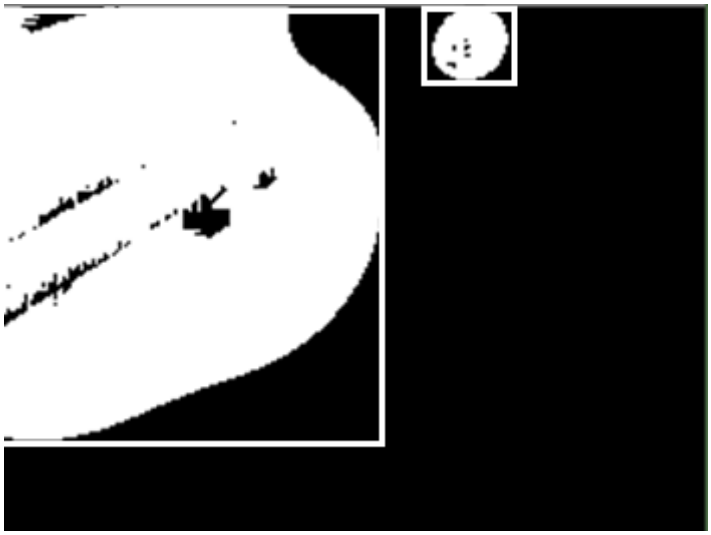


Figure 15 : **Le plus petit rectangle contenant l'obstacle**

v. Combinaison des résultats des méthodes de segmentation

Comme expliqué auparavant, nous utilisons trois méthodes pour la segmentation *background/foreground* à savoir la mixture de gaussiennes, le flux optique et le contour de *Canny*. Cette approche nous permet de détecter le maximum d'obstacles.

Certes, cela peut engendrer une « *surestimation* » d'obstacles, qui conduira à détourner parfois des obstacles qui n'existent pas. Mais dans un cadre où le risque d'ignorer des obstacles peut engendrer des graves dégâts, nous estimons le pire des cas, et nous cherchons à combiner les résultats de plusieurs méthodes au lieu de ne considérer qu'une.

La méthode de détection de contours est la plus sensible au changement de luminance dans une image. Elle détecte tous les contours dans une texture. Cela amènes à segmenter des régions comme étant des obstacles, alors qu'elles font partie du background.

La méthode de mixture des gaussiennes, peut tolérer certains obstacles s'ils restent '*longtemps*' dans le champ de vue. La méthode refait l'apprentissage dans un laps de temps, ainsi les objets qui font partie de la scène une période de temps suffisante, risquent d'être considérés comme des objets de l'arrière-plan.

Quoique la méthode du flux optique présente les meilleurs résultats, elle omet généralement les objets statiques dans la scène tels que les murs.

En effet, nous utilisons la combinaison des trois méthodes, et nous procédons ainsi par un *vote*, où chaque méthode nous fournit la position de l'obstacle le plus proche dans chaque barre. Si deux méthodes au moins « *sont d'accord* » sur une réponse, nous la choisissons. Si les réponses des trois méthodes diffèrent l'une de l'autre, et si par exemple la méthode 1 dit que l'obstacle est critique, alors que la méthode 2 dit qu'il est loin, et la troisième *dit* qu'il est proche, alors nous pouvons

rien conclure sur ce cas, et par principe de « *la sécurité d'abord* », nous choisissons le cas le plus échéant, et nous concluons que l'obstacle est critique.

En fait, trois réponses en accord valent mieux qu'une seule, et pour refléter cela, nous associons les votes des trois méthodes avec un indice dit de *fiabilité*. Cet indice sera traduit par un poids dans notre système de décision, ce poids est ensuite associé aux règles en question. Par conséquent, si les trois règles sont *en accord*, nous avons un degré de fiabilité de *100%*, deux règles avec *66%* et une seule règle avec *33%*.

III.2. Sous système de décision

Dans cette partie, nous allons exposer les différentes composantes de notre système de décision. Nous allons voir les différentes variables du système, la base des règles utilisée et l'architecture que nous avons adoptée pour réduire le nombre de règles.

i. Architecture

Le système de décision implémente un contrôleur en logique floue, il a cinq variables d'entrées (une variable pour chaque barre dans l'image) et deux variables de sortie étant la vitesse et l'angle de rotation. Le schéma synoptique du contrôleur est le suivant :

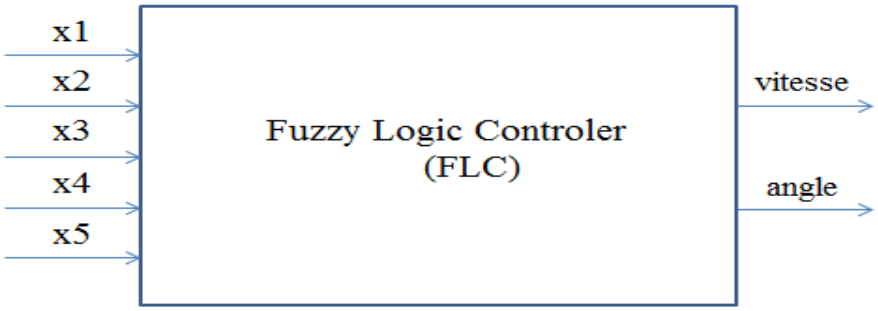


Figure 16 : **Système de décision et ses variables**

ii. Les variables du système

a. Variables d'entrée

En amont, le système a cinq variables, chaque variable présente la position du plus proche obstacle détecté sur une barre donnée. Nous avons divisé l'image en cinq barres, ainsi nous avons cinq variables linguistiques, qui ont le même univers de discours (l'ensemble des valeurs possibles que peuvent prendre une variable). Dans notre cas l'univers de discours est *[-10,180]* avec *180* est la hauteur de l'image en pixel. Ces variables ont aussi les mêmes fonctions d'appartenance.

Chaque variable d'entrée peut appartenir à un ou plus de l'un des cinq ensembles flous suivant : ***Critique***, ***Proche***, ***Moyen***, ***Loin*** ou ***Libre***.

Les valeurs d'appartenance de chaque variable à ces ensembles sont données par la fonction d'apparenance dans la (*figure 17*) (la même fonction pour les cinq variables).

Nous considérons un obstacle comme étant *critique*, s'il est à une distance plus proche qu'il ne soit évité, à moins d'un demi-mètre de l'utilisateur, alors la région concernée ne peut être utilisée pour la navigation. C'est-à-dire, si nous sommes déjà dans une région critique (dans la barre centrale on détecte un obstacle comme étant *critique*), nous devons s'arrêter immédiatement. Si une barre de gauche ou de droite est *critique* cette région devient non pénétrable, (à voir les règles formelles plus loin). Si l'obstacle est à plus d'un demi-mètre de l'utilisateur, et à moins de deux mètres, il est qualifié alors comme « *proche* ». Entre plus de deux mètres et moins de quatre mètres, l'obstacle est « *moyen* ». Entre 4 et 5,2 mètres, l'obstacle est déclaré comme « *loin* ». Si la barre ne contient aucun obstacle, on dit qu'elle est « *libre* ».

Position de l'obstacle	*Ensemble flou*
Moins de 0.5m	Critique
Entre 0.5m et 2m	Proche
Entre 2m et 4m	Moyen
Entre 4m et 5.2m	Loin
Aucun obstacle	Libre

Tableau 2 : Ensemble flous pour la position de l'obstacle

Les mesures aux obstacles sont définies en termes de pixels, (correspondance faite dans la partie de la division de l'image), et nous avons défini les la fonction d'appartenance suivante illustrés dans le graphe suivant :

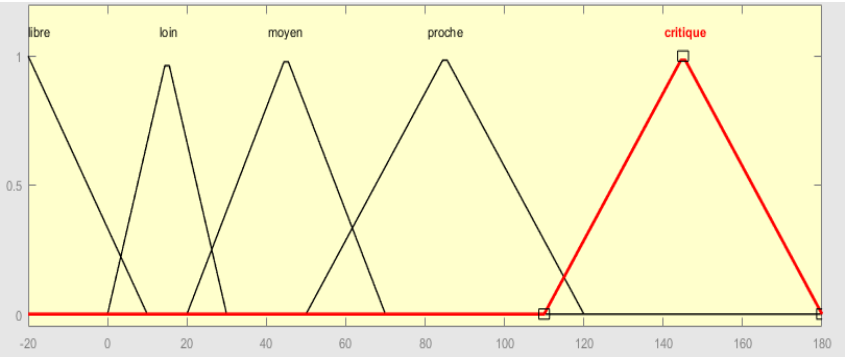


Figure 17 : **Fonction d'appartenance des variables d'entrée.**

En abscisse la postions de l'obstacle en pixel, en coordonnée le degré d'appartenance à un ensemble flou.

b. Variables de sortie

En aval, le système de décision répond par deux variables, qui sont la vitesse de navigation, et l'angle de rotation.

La variable « *vitesse* » a pour univers de discours *[-10,100]*. Cette vitesse n'est pas définie par une unité précise, mais plutôt définie pour être interprétée selon l'état final du système et ses spécificités.

Cette variable peut appartenir à un ou plusieurs ensembles flous suivants : ***stop, slow, medium, fast***. L'appartenance d'une variable à ces ensembles est déterminée par la fonction d'appartenance suivante :

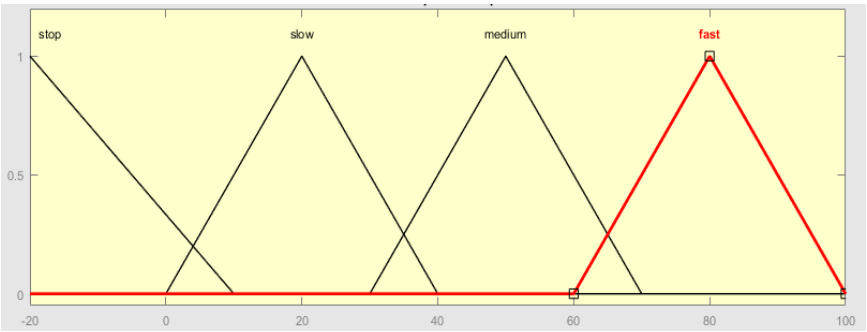


Figure 18 : Fonction d'appartenance de la variable vitesse

La variable de sortie « *direction* » indique le degré de rotation que l'utilisateur doit suivre pour éviter l'obstacle. Cette variable à pour univers de discours [0,100] (un degré relatif interprété selon l'état final du système) et peut appartenir à l'un (ou plusieurs) des ensembles flous suivants : ***left, forward, right.*** L'appartenance d'une variable à ces ensembles est donnée par la fonction d'appartenance suivante :

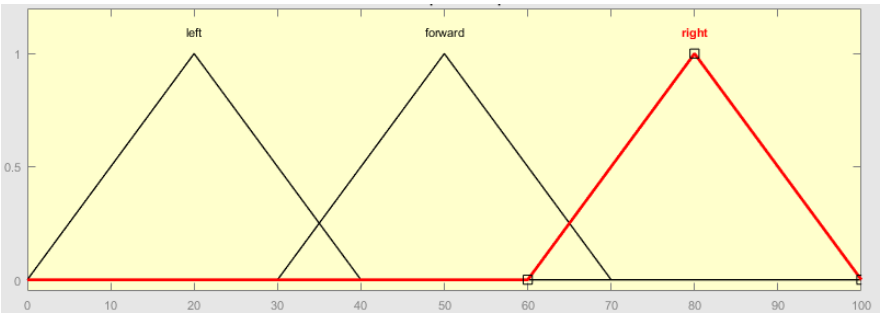


Figure 19 **: Fonction d'appartenance de la variable direction**

iii. Problème de dimensions

Soit m le nombre des termes linguistiques (nombre d'ensembles flous auxquels une variable peut appartenir), et n le nombre de variables. Ainsi, pour une définition exhaustive de l'ensemble des règles à appliquer à ces variables, nous avons m^n combinaisons possibles. En fait, le nombre des règles augmente exponentiellement avec le nombre de variables.

Etant donné que nous avons cinq variables linguistiques et nous avons défini cinq termes linguistiques pour chacune, nous devons citer : $5^5 = 3125$ règles ! Ce problème est connu dans la littérature sous le nom de *curse of dimensionality,* et plusieurs approches ont été proposées pour le résoudre [25] [26] [27].

vi. Structure hiérarchique

Pour réduire le nombre de règles à introduire, nous proposons une architecture hiérarchique pour le système de décision. L'idée est de décomposer le contrôleur principal en des sous-contrôleurs où chacun va manipuler un sous ensemble de variables à la fois. Ainsi, pour les cinq variables d'entrée que nous avons, au lieu de les manipuler toutes à la fois, nous allons manipuler deux variables avec un sous-contrôleur 1, qui va donner une variable en sortie. Un sous-contrôleur 2, va traiter deux autres variables, et donne une variable en sortie, et un troisième sous-contrôleur, va combiner les résultats des deux premiers avec la cinquième variable.

L'affectation des variables aux sous-contrôleurs, n'est pas arbitraire, nous affectons au contrôleur *FC1* les variables *x1* et *x2*, qui présentent respectivement les régions extrême-gauche et gauche. Nous aurons en sortie un indice qui reflète l'encombrement de la région gauche d'obstacles. De même, au sous-contrôleur 2, nous affectons les deux variables *x4* et *x5* qui désigne respectivement les barres de droite et extrême droite. Le sous-contrôleur 3, manipule la variable indiquant la barre de centre, et les sorties des deux sous-contrôleurs 1 et 2.

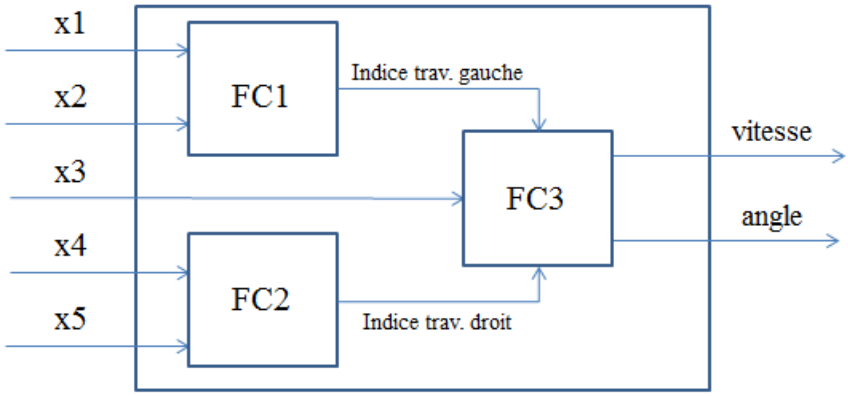


Figure 20 **: architecture hiérarchique du contrôleur flou**

Avec cette architecture, le nombre des règles augmente linéairement, puisque le nombre des règles total est la somme de nombre des règles des sous-contrôleurs.

Les variables d'entrée *x1…x5,* sont distribuées sur les trois sous-contrôleurs. Nous avons alors pour le *FC1* deux variables d'entrée avec cinq termes linguistiques. Aussi, nous obtenons $5^2=25$ combinaisons pour citer les différentes règles du *FC1*, pour *FC2,* nous avons de même *25* règles, et pour FC3 nous avons *5*3*3 = 45* règles. Alors pour tout le système nous avons besoin de citer 95 règles. Nous remarquons bien que cela nous a aidés à réduire le nombre de règles à citer de *3125* à *95*.

L'architecture hiérarchique introduit de nouvelles variables intermédiaires dans le système. Par conséquent, nous devons définir des variables de sorties pour les sous contrôleurs FC1 et FC2, qui seront ensuite les entrées du FC3. Le problème qui se pose est la sémantique des variables intermédiaires, puisque c'est un état « intra-système ». Alors nous devons définir une « sémantique » de ces variables intermédiaires pour que nous puissions définir leurs fonctions d'appartenance, et les utiliser par la suite dans les règles.

Nous introduisons un nouveau paramètre, étant la *traversabilité d'une région*. À partir des deux variables de la région gauche (la gauche et l'extrême gauche) et la région droite (la droite et l'extrême droite) nous définissons un indice dit de traversabilité, et qui indique si une région est encombrée d'obstacles.

La variable « *indice de traversabilité* »

Nous définissons une variable appelée *indice de traversabilité*, définie sur l'univers de discours *[0,1]* et peut appartenir à l'un (ou plus) des ensembles flous suivants : ***traversable, « traversable moyen »*** et ***« non traversable ».***

La fonction d'appartenance de cette variable est la suivante :

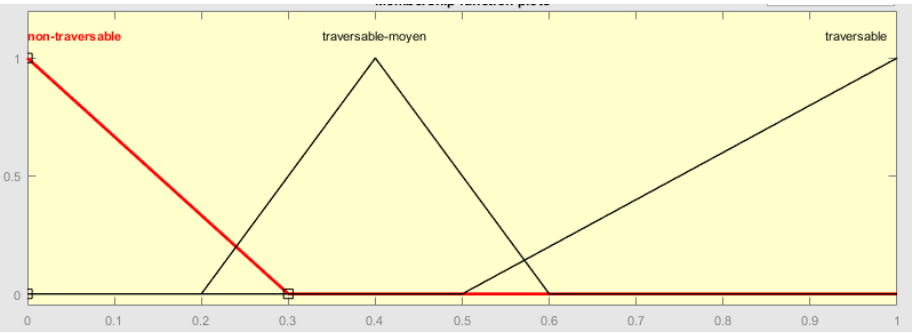


Figure 21 **: Fonction d'appartenance de la variable "indice de traversabilité"**

iv. Base des règles

La base des règles permet de regrouper les connaissances de l'expert sur un domaine spécifique. Dans notre cas, l'expert doit citer les différentes règles qui permettront la navigation dans l'environnement, tout en sachant les postions des obstacles dans chaque région.

Pour cela nous devons citer 95 règles pour exprimer tous les cas possibles, or par intuition des règles peuvent être éliminées, *i.e.* des règles qui ne peuvent pas se réaliser par exemple.

Nous définissons pour chaque sous-contrôleur un ensemble des règles, les sous-contrôleurs 1 et 2 (des régions gauche et droite respectivement), ont presque les mêmes règles, il suffit de remplacer les noms des variables *x1*, *x2* par *x5* et *x4*, alors nous présentons ici juste la base de règles de FC1, il va de même pour FC2.

a. Base des règles de FC1

La première règle exige que si la variable *x2* (présentant la barre *x2* dans l'image) contient un obstacle à une distance « *critique* », alors la région gauche devient non traversable. Nous ne nous soucions pas de ce qui est la variable *x1* (présentant l'extrême-gauche). Bien que *x1* soit libre, nous ne pouvons pas utiliser cette direction car l'obstacle dans *x2* bloque ce chemin. Ainsi, nous avons évité de tester les autres valeurs de *x1*, et nous avons éliminé quatre règles.

Le reste des règles est cité d'une manière exhaustive. Les règles reflètent la connaissance de l'expert et sont exprimées en termes linguistiques. Elles sont interprétables par l'humain comme par la machine, par exemple, si l'obstacle est qualifié comme « *moyen* » pour *x1*, et « *moyen* » pour *x2*, nous déduisons que la région est « *traversable moyen* ».

```
"IF x2 IS critique THEN output IS non_traversable"

"IF x1 IS libre AND x2 IS loin THEN output IS traversable"
"IF x1 IS libre AND x2 IS libre THEN output IS traversable"
"IF x1 IS libre AND x2 IS moyen THEN output IS traversable"
"IF x1 IS libre AND x2 IS proche THEN output IS traversable_moyen"

"IF x1 IS LOIN AND x2 IS libre THEN output IS traversable"
"IF x1 IS LOIN AND x2 IS LOIN THEN output IS traversable"
"IF x1 IS LOIN AND x2 IS moyen THEN output IS traversable_moyen"
"IF x1 IS LOIN AND x2 IS proche THEN output IS traversable_moyen"

"IF x1 IS moyen AND x2 IS libre THEN output IS traversable"
"IF x1 IS moyen AND x2 IS LOIN THEN output IS traversable"
"IF x1 IS moyen AND x2 IS moyen THEN output IS traversable_moyen"
"IF x1 IS moyen AND x2 IS proche THEN output IS non_traversable"

"IF x1 IS proche AND x2 IS libre THEN output IS traversable_moyen"
"IF x1 IS proche AND x2 IS LOIN THEN output IS traversable_moyen"
"IF x1 IS proche AND x2 IS moyen THEN output IS non_traversable"
"IF x1 IS proche AND x2 IS proche THEN output IS non_traversable"

"IF x1 IS critique AND x2 IS libre THEN output IS traversable_moyen"
"IF x1 IS critique AND x2 IS LOIN THEN output IS traversable_moyen"
"IF x1 IS critique AND x2 IS moyen THEN output IS non_traversable"
"IF x1 IS critique AND x2 IS proche THEN output IS non_traversable"
```

Figure 22 : Base de règles de FC1

b. Base des règles de FC3

La base de règles de FC3 est la base des règles « *principale* ». Elle prend la sortie des deux sous-contrôleurs des régions gauche et droite et les combine avec l'état de la barre centrale pour donner le résultat final du système.

La première règle dans cette base exige que si la barre centrale est qualifiée comme critique, *i.e.* contient un obstacle très proche que ne nous pouvons l'éviter (à moins d'un demi mètre), alors nous ne pouvons pas contourner l'obstacle même si la région gauche ou droite sont libres, dans ce cas nous devons s'arrêter.

Si la barre centrale est *libre* ou contient un obstacle *loin* (à plus de 4 mètres), alors on peut continuer tout droit avec une grande vitesse puisqu'il n'y a aucun danger d'heurter un obstacle. En plus, si la barre centrale est *libre*, alors ce n'est pas la peine de de déclencher des calculs inutiles et nous continuons notre chemin, puisqu'aucune destination n'est prédéfinie.

```
"if central is CRITICAL then velocity is stop"

"if central is LIBRE then direction is direct and velocity is high"

"if central is LOIN then direction is direct and velocity is high"

"if central is MOYEN and right is traversable and left is traversable then direction is droit and velocity is high"
"if central is MOYEN and right is traversable and left is traversable_moyen then direction is droit and velocity is high"
"if central is MOYEN and right is traversable and left is non_traversable then direction is droit and velocity is high"
"if central is MOYEN and right traversable_moyen and left is traversable then direction is gauche and velocity is high"
"if central is MOYEN and right traversable_moyen and left is traversable_moyen then direction is direct and velocity is medium"
"if central is MOYEN and right traversable_moyen and left is non_traversable then direction is direct and velocity is medium"
"if central is MOYEN and right non_traversable and left is traversable then direction is gauche and velocity is high"
"if central is MOYEN and right non_traversable and left is traversable_moyen then direction is direct and velocity is medium"
"if central is MOYEN and right non_traversable and left is non_traversable then direction is direct and velocity is medium"


"if central is PROCHE and right is traversable and left is traversable then direction is droit and velocity is high"
"if central is PROCHE and right is traversable and left is traversable_moyen then direction is droit and velocity is high"
"if central is PROCHE and right is traversable and left is non_traversable then direction is droit and velocity is high"
"if central is PROCHE and right traversable_moyen and left is traversable then direction is gauche and velocity is high"
"if central is PROCHE and right traversable_moyen and left is traversable_moyen then direction is droit and velocity is medium"
"if central is PROCHE and right traversable_moyen and left is non_traversable then direction is droit and velocity is medium"
"if central is PROCHE and right non_traversable and left is traversable then direction is gauche and velocity is high"
"if central is PROCHE and right non_traversable and left is traversable_moyen then direction is gauche and velocity is medium"
```

Figure 23 : Base de règles de FC3

Ensuite, si la barre centrale déclare un obstacle à une distance « *moyenne* » alors nous cherchons à évaluer les alternatives : si les deux régions *(gauche et droite)* sont traversables, alors il est préférable de passer par la région droite (un choix complétement aléatoire, vu que nous devons choisir une direction), sinon, si une seule région est traversable nous la suivons.

Si les deux régions (*gauche et droite*) sont « *non traversable* » alors ce n'est pas la peine de changer de direction, mais plutôt nous continuons dans la même direction avec une vitesse moyenne.

L'autre cas, si un obstacle est détecté sur la barre centrale comme étant « *proche* », et si les deux régions sont « *traversables »,* alors la région *droite* sera choisie à une grande vitesse. Si une seule région est « *traversable »*, elle sera poursuivie à une grande vitesse. Si l'une des deux régions est « *traversable moyen* » nous la suivons avec une vitesse moyenne.

La règle par défaut dans le cas où aucune règle n'est activée, est « *slow forward* » qui signifie de rester dans la direction, avec une petite vitesse. Cette règle sera activée par exemple si les régions gauche et droite sont « *non traversable* » et un obstacle devant est « *proche* ».

IV. Conclusion

Dans ce chapitre, nous avons vu en détails l'approche que nous avons adoptée. Notre approche se distingue par l'utilisation d'une seule caméra qui cause

des aléas. En effet, dans la projection de l'espace tridimensionnel sur le capteur bidimensionnel de la caméra, l'information sur la profondeur de la scène est perdue.

Nous avons divisé notre système en deux sous-systèmes. Le premier est un sous-système de de vision, qui est responsable de la détection des obstacles dans la séquence vidéo. Le deuxième est un sous-système de décision, qui est responsable de l'évitement des obstacles et qui était implémenté en logique floue. La logique floue a pour but de remédier aux imprécisions des mesures fournies par le sous-système de vision, et qui sont dues à notre incapacité d'estimer la profondeur à partir de l'image bidimensionnelle.

La croissance exponentielle du nombre des règles dans le sous-système de décision, reconnue sous le nom de « *curse of dimensionnality*» présentait une entrave inévitable donné les dimensions de notre système. Cependant, il nous semblait qu'une architecture hiérarchique du système est apte de corriger cette problématique. Nous avons alors divisé notre contrôleur en trois sous contrôleurs. Chaque contrôleur accomplit un nombre de tâches et est connecté aux autres contrôleurs.

Chapitre 4 : Implémentation et tests du système

I. Introduction

Dans ce chapitre nous allons voir l'implémentation de l'approche présentée dans le *chapitre 3*.

En fait, nous avons développé une application en *C++* sous *Visual Studio*. Nous avons utilisé la bibliothèque *OpenCV* pour implémenter le sous-système de vision, et *FuzzyLite* pour le sous-système de décision. Nous allons présenter une brève introduction de ces deux bibliothèques ci-après.

Ensuite nous allons voir les expériences que nous avons menées pour tester notre système, nous allons exposer les différentes conditions sous lesquelles nous l'avons testé. Ensuite nous allons voir les différents résultats que nous avons obtenus, et les différents critères utilisés pour évaluer la performance du système.

Pour atteindre le temps réel dans notre système, nous avons utilisé la programmation parallèle, et la programmation GPU, nous allons donner le principe de ces techniques, et exposer leurs utilisations dans notre projet.

II. Bibliothèques utilisées

II.1. *OpenCV*

OpenCV (***Open*** *Source* ***C****omputer* ***V****ision*) est une bibliothèque proposant un ensemble de plus de *2500* algorithmes de vision par ordinateur, accessible à travers l'*API* pour les langages *C*, *C++,* et *Python*. Elle est distribuée sous une licence *BSD* (libre) pour les plateformes *Windows*, *GNU/Linux*, *Android* et *MacOs*.

Initialement écrite en *C* en *2010* par des chercheurs de la société Intel, *OpenCV* est aujourd'hui développée, maintenue, documentée et utilisée par une communauté de plus de *40 000* membres actifs. C'est la bibliothèque de référence pour la

vision par ordinateur, aussi bien dans le monde de la recherche que celui de l'industrie.

Afin de mieux présenter son étendue et ce qu'elle permet de faire, jetons un œil aux principaux modules accessibles au travers de son *API C*.

Module	*Fonctions*
core	Cette bibliothèque permet de manipuler les structures de base, réaliser des opérations sur des matrices, dessiner sur des images, sauvegarder et charger des données dans des fichiers *XML*…
imgproc	Les fonctions et structures de ce module ont trait aux transformations d'images, au filtrage, à la détection de contours, de points d'intérêt…
features2d	Ce module concerne principalement l'extraction de descripteurs selon deux approches courantes (*SURF et StarDetector*), utiles à la caractérisation d'images.
objdetect	Cette bibliothèque permet de faire de la reconnaissance d'objets dans une image au moyen de l'algorithme *Adaboost* (*Viola & Jones, 2001*). Utilisée pour l'apprentissage et la reconnaissance de formes.
video	Sert à segmenter et suivre les objets en mouvement dans une vidéo
highgui	Une bibliothèque haut-niveau pour ouvrir, enregistrer et afficher des images et des flux vidéo. Celle-ci contient aussi un certain nombre de fonctions permettant de réaliser des interfaces graphiques simples
calib3d	Ce module contient des fonctions permettant de reconstruire une scène en 3D à partir d'images acquises avec plusieurs caméras simultanément.

Tableau 3: Modules de la bibliothèque OpenCV

II.2. *FuzzyLite*

FuzzyLite est une bibliothèque de la logique floue multiplateforme écrite en *C++* en *2010* par *Juan Rada-Vilela*, et distribuée sous une licence *Apache*.

Cette bibliothèque fournit la conception et les options des contrôleurs en logique floue avec une approche orientée objet. Elle peut être facilement incorporée dans l'application.

Parmi les fonctionnalités de *FuzzyLite* nous citons :

Fonctionnalité	*Exemples*
Les contrôleurs	Mamdani, Takagi-Sugeno et Tsukamoto.

Les termes linguistiques	Triangle, Gaussienne, Trapézoïde, Rectangle
Les T-Normes	Minimum, algebraic product,bounded difference, drastic product, einstein product, hamacher product
Les S-Normes	Maximum, algebraic sum, bounded sum, normalized sum, drastic sum, einstein sum, hamacher sum.
Defuzzifiers	Centroid, bisector, smallest of maximum, largest of maximum, mean of maximum, weighted average, weighted sum

Tableau 4 : Fonctionnalités de la bibliothèque *Fuzzylite*

Les classes principales de la bibliothèque sont données dans la figure ci-après (*fig.24*).

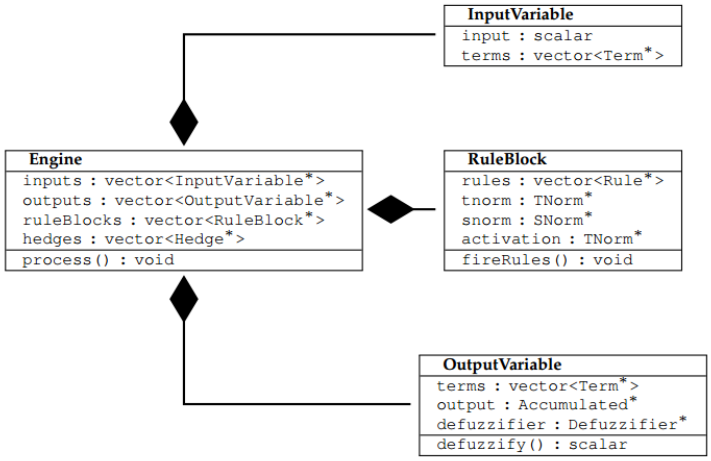


Figure 24 **: Diagramme des classes de *FuzzyLite***

III. Développement de l'application

III.1. Environnement de l'implémentation

Nous avons développé l'application en *C++* avec *Visual Studio 2010*, sous un système *Windows 7,* et nous avons utilisé les bibliothèques *OpenCV* et *FuzzyLite* présentés plus haut.

Côté matériel, nous avons utilisé une machine *Dell XPS M1510* avec la configuration :

Processeur	Core 2 Duo 2.5 GHz
RAM	4GB
Carte graphique	256 Mb Nvidia GForce 8600 GT

Tableau 5 : Configuration de la machine d'implémentation

III.2. Architecture de l'application

Le « *flux* » de l'application est d'une nature séquentielle. Chaque trame de la vidéo suit un ensemble de traitements dans un ordre précis. Le code de l'application est constitué d'un ensemble de procédures (fonctions) qui ont l'arborescence de la figure suivante.

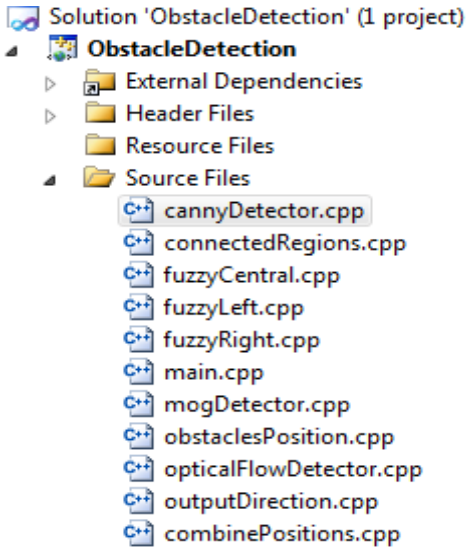


Figure 25 **: Arborescence de l'application sous *Visual Studio***

Chaque fonction accomplit une ou un ensemble de tâches explicitées dans le tableau suivant.

Fonction	*Tâche*
readframe	Lit une trame à la fois à partir de la séquence vidéo, ou éventuellement directement d'une caméra
resizeframe	Réduit les dimensions de la trame d'une taille quelconque à 240*180.
mogDetector	Reçoit une trame en entrée et y applique la méthode de soustraction du *background,* qui retourne un masque où les pixels noirs sont ceux du background et les pixels blancs sont ceux qui désignent des obstacles. Nous appliquons à ce masque par la suite une opération d'érosion pour réduire les pixels isolés et qui sont

	généralement dues au bruit.
cannyDetector	Prend une trame en entrée, la transforme en niveau de gris, puis y applique un filtre gaussien pour la réduction du bruit, ensuite détecte les contours par la méthode de *Canny*, et retourne une image binaire.
OFDetector	Reçoit une trame en entrée. Nous calculons les vecteurs de mouvement de chaque pixel avec la trame précédente, puis nous traçons ces vecteurs en blanc sur une image noire. Ainsi, nous avons en blanc les pixels qui ont eu un mouvement de trame en trame, et qui appartiennent généralement à un obstacle. Cette méthode retourne une image binaire.
connectedRegions	Prend en entrée une image binaire, et calcule puis retourne les coordonnées des rectangles contenants les régions connectées avec une connectivité de 4.
obstaclesPosition	Scanne l'image du bas vers le haut, cherchant la position du premier obstacle dans chaque barre.
combinePositions	Cette méthode prend en entrée les positions des obstacles pour chaque masque des trois méthodes de segmentation, et les combinent afin d'avoir la position d'obstacle dans chaque barre, avec un poids de fiabilité. (Cette opération est détaillée dans le 3éme chapitre).
fuzzyLeft &fuzzyRight	Prennent en entrée les positions des obstacles dans les deux barres x1, x2 (barres de la région gauche) et *x4,x5* (barres de la région droite) respectivement, et le poids de fiabilité. Définissent les fonctions d'appartenances et les règles utilisées, et retournent la valeur du paramètre « indice de traversabilité ».
fuzzyCentral	Prend en entrée les indices de traversabilité de *fuzzyLeft* et *fuzzyRight*, et la position de l'obstacle dans x3. Définit les fonctions d'appartenances et les règles utilisées et retourne le degré de rotation, et la vitesse.
outputDirection	Prend la valeur de la direction et la vitesse obtenus du « *fuzzyCentral* » et les délivre sous une forme exploitable par l'utilisateur finale. Nous utilisons des directives vocales, qui indiquent la vitesse et la direction à suivre.

Tableau 6 : Fonctions de l'application

Pour visualiser le « *flow* » de l'application, aussi la hiérarchie des fonctions présentées auparavant, nous dressons le diagramme suivant.

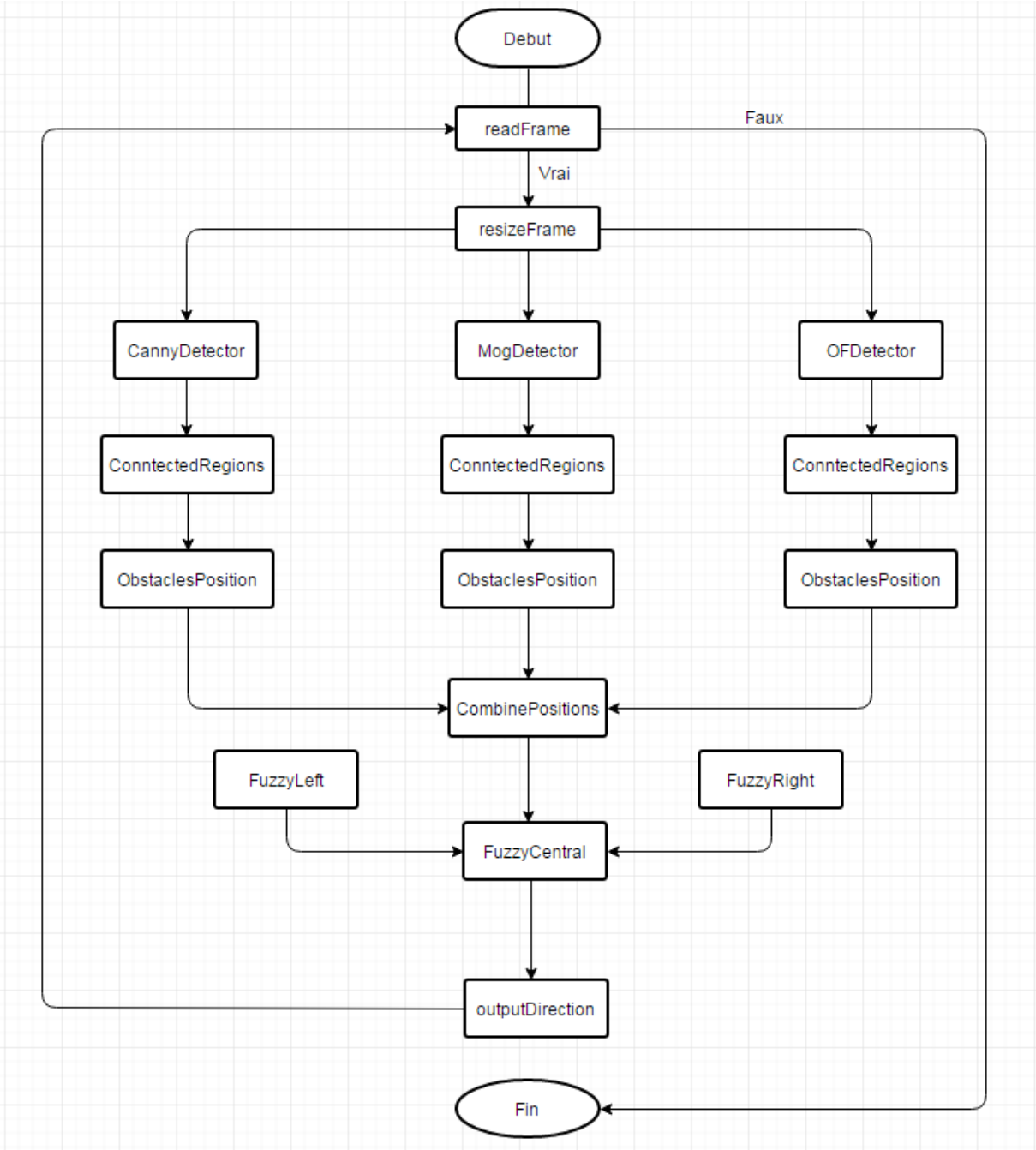


Figure 26 : **Diagramme "*flow chart*" de l'application**

V. Tests de l'application

V.1. Environnements des tests

Afin de tester l'application, nous avons utilisé 24 vidéos de longueurs entre 14 et 90 secondes, sous différentes conditions. Les critères pris en considération sont :

- Des scènes qui ne contiennent pas d'obstacles et d'autres qui sont totalement occupées d'obstacles.
- Des scènes d'intérieur (*indoor*) avec des arrière-plans (*background*) unifiés et/ou avec des motifs (*pattern*) qui se répètent et une luminosité invariante. Alors que d'autres scènes sont filmées en plein air (*outdoor*), qui sont exposées aux changements de luminosité (contenant des ombres par exemple) et qui contiennent des trajectoires divers (différents types de trottoir par exemple).
- Des vidéos filmées avec différentes caméras, qui diffèrent (les vidéos) par leurs dimensions, nombre de trames par seconde, résolution…

Le tableau suivant présente ces conditions en détails :

Type caméra	*Nb. Vidéos*	*Dimensions*	*Résolution*	*Nb. Trames /sec*	*Indoor/ Outdoor*	*Encombrement par les obstacles*
Canon	10	1440*1080	20 MP	25	6 indoor 4 outdoor	2 vidéos très occupées d'obstacles. 2 vidéos ne contenants aucun obstacle. 6 vidéos contenants peu d'obstacles.
Samsung S5	3	480*270	5 MP	29	3 outdoor	1 vidéo ne contenant aucun obstacle. 2 vidéos contenant quelques obstacles.
Samsung GT	6	320*240	3.2 MP	15	6 indoor	2 vidéos sans obstacles. 4 vidéos contiennent quelques obstacles.
Sony Xperia Z1	5	640*480	5 MP	29	2 indoor 3 outdoor	1 vidéo sans obstacles 3 vidéos avec peu d'obstacles. 1 vidéo très occupée d'obstacles

Tableau 7:Les différents paramètres des expériences

V.2. Résultats

Avant d'exposer les résultats des tests effectués à notre application, nous allons définir les critères selon lesquelles nous avons évalué le système.

i. Taux d'exactitude

Le premier critère que nous avons défini est le *taux d'exactitude*. Pour cela nous avons calculer le nombre de directives correctes. Puis nous divisons ce nombre par le nombre total des directives (nombre de directions annoncées durant une vidéo).

Les taux d'exactitude varient notament en fonction des types des scènes filmées (indoor/outdoor).

Nombre de vidéos	*Type de vidéo*	*Taux d'exactitude*
14	indoor	87.41%
10	outdoor	63.17%

Tableau 8 : Résultats en taux d'exactitude

Ces résultats s'expliquent par le fait que les chemins des scènes *outdoor* varient brusquement. Ce qui exige à la méthode d'apprentissage du *background* utilisée de refaire l'apprentissage. Tandis que pour la durée nécessaire à la reconstruction de modèl, nous nous retrouvons devant un chemin qui est tout a fait nouveau, et qui est détecté comme un obstacle.

Ce problème peut être remédié par l'utilisation d'une connaissance à priori sur des chemins qui sont fréquemment abordés. Cette connaissance peut être obtenue par un apprentissage avec les réseaux de neurons par exemple. Une amélioration qui sera appréciée dans les perspectives de notre projet.

ii. Taux d'optimisation

Nous avons évalué un autre critère que nous avons appelé *taux d'optimisation*, et il est calculé par :

$$\frac{nombre\ de\ directives\ optimales}{nombre\ total\ des\ directives}$$

Nous définissons une directive optimale comme étant une directive qui n'est pas seulement une correcte, mais « *la plus correcte* » sachant qu'une directive correcte est une directive qui empêche d'heurter un obstacle. Par exemple, une directive qui suggère de tourner à gauche, alors que le chemin devant est libre, est une

directive correcte, mais pas optimale, car même s'elle évite une collision, elle déclenche un changement de direction inutile. Ainsi, nous caractérisons les directives optimales comme suit:

- Une directive optimale exige le minmum de changement de direction.
- Une directive optimale choisit la direction la moins encombrée d'obstacles.

Pour ce critère, nous avons obtenu les résultats suivants :

Nombre de vidéos	*Type de vidéo*	*Taux d'optimisation*
14	Indoor	72.15%
10	Outdoor	43.61%

Tableau 9 : Résultats en taux d'optimisation

iii. Temps d'exécution

Une tâche crcuciale pour notre système est de s'éxecuter en temps réel. Nous avons testé l'application sur des vidéos enregistrées, ainsi, la tâche que nous devons accomplir en termes de temps d'éxécution est de traiter la vidéo dans un temps égal à sa longueur.

Les résultats que nous avons obtenu avant d'optimiser l'application en temps de calcul sont fournies dans le tableau suivant :

Type de caméra	$\frac{temps\ de\ traitement}{longueur\ de\ la\ vidéo} \times 100$
Canon	413%
Samsung S5	317%
Samsung GT	221%
Sony Xperia Z1	342%

Tableau 10 : Résultats en temps d'exécution

Nous remarquons que le temps de traitement croît proportionnelement avec la dimension de la trame et le nombre de trames par secondes.

Pour réduire le temps de calculs et atteindre le temps réel, nous avons utilisé plusieurs techniques à savoir le programmation parallèle, la programmation GPU et la diminution du nombre de trames à traiter par seconde.

V.3. Temps réel

i. Programmation parallèle

La programmation parallèle consiste à diviser un programme en plusieurs tâches séquentielles s'exécutant simultanément, habituellement appelées *threads* et peuvent être de plusieurs types selon le système d'exploitation ou la machine virtuelle.

La programmation parallèle est devenue un outil indispensable, notamment pour des applications à haut débit. Un algorithme parallèle consiste à partitionner le problème étudié en tâches élémentaires afin que ces tâches soient exécutées simultanément dans plusieurs processeurs en vue que le temps d'exécution soit plus rapide que celui de l'algorithme séquentiel.

Plusieurs bibliothéques sont destinées à la programmation parallèle, entre autres nous pouvons citer *PTHREADS, INTEL THREADING BUILDING BLOCKS, OPENMP…*

Nous avons utilisé *OPENMP* qui est une bibliothèque supportée par plusieurs langages (*C,C++ et Fortran*) et disponible sur plusieurs plate-formes *(Linux, Windows, OS X, ...). OpenMP* regroupe des directives de compilation et des fonctions. Le compilateur *gcc* supporte *OpenMP* depuis la version 4.2, en ajoutant simplement une option sur la ligne de commande et en incluant le fichier d'en-têtes ***omp.h***. La gestion des différentes fonctions est assurée par la libraire ***libgomp***.

ii. GPU

La programmation ***GPU***, aussi appelée ***G****eneral-****p****urpose* ***c****omputing on* ***g****raphics* ***p****rocessing* ***u****nits (****GPGPU****)* en anglais, est l'utilisation du processeur graphique dans le but d'effectuer des calculs normalement effectués par le processeur central *(CPU)* de l'ordinateur. En raison de l'architecture particulière des *GPU*, les calculs pouvant être parallélisés.

Le calcul par le GPU permet de paralléliser les tâches et d'offrir un maximum de performances dans de nombreuses applications, le *GPU* accélère les portions de code les plus lourdes en ressources de calcul, le reste de l'application restant affecté au *CPU*. Les applications des utilisateurs s'exécutent aussi bien plus rapidement.

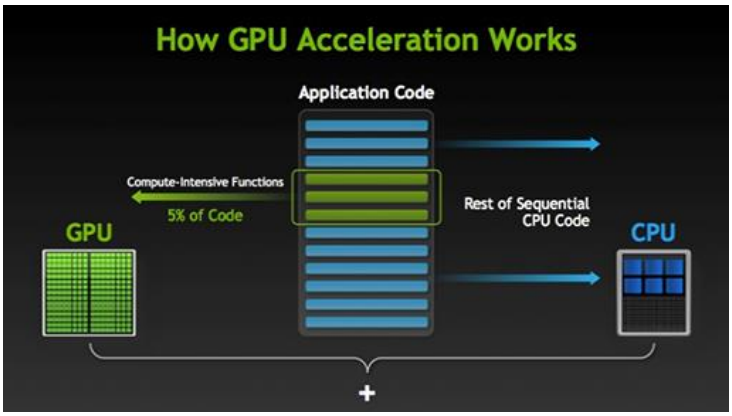


Figure 27 **: Répartition du travail entre le *CPU* et le *GPU***

Pour comprendre les différences fondamentales entre un *CPU* et un *GPU*, il suffit de comparer leur manière de traiter chaque opération. Les *CPU* incluent un nombre restreint de cœurs optimisés pour le traitement en série, alors que les GPU intègrent des milliers de cœurs conçus pour traiter efficacement de nombreuses tâches simultanées.

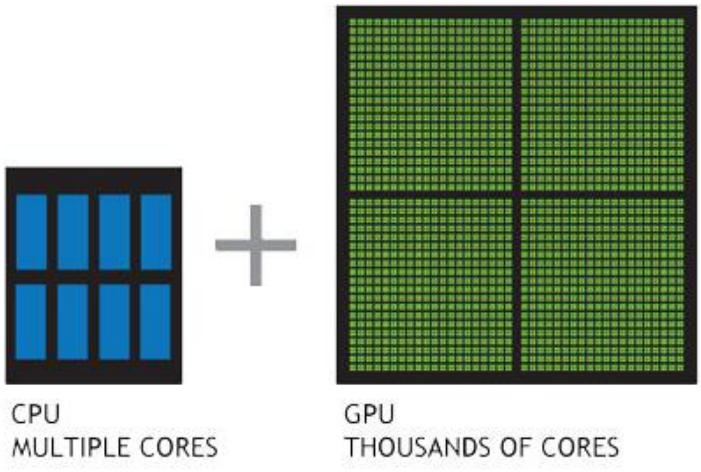


Figure 28 **: Les *GPU* incluent des milliers de cœurs pour traiter efficacement des tâches parallèles.**

Plusieurs techniques permettent d'utiliser les *GPU* :

- Programmation en *CUDA*;
- *Toolbox GPUmat* pour *MATLAB*;
- *Toolbox Jacket* pour *MATLAB*;
- Packages *CUDALink* et *OpenCLLink* de *Mathematica*.

OpenCV utilise la programmation en *CUDA*, nous avons utilisé les fonctions *GPU* implémentées dans *OpenCV*. Ces fonctions sont plus rapides que les fonctions implémenté sur le *CPU*.

iii. Temps de calcul

Nous avons réduit le nombre de trames à traiter par secondes. Le traitement est appliqué à une trame sur six (5 trames seront sautées). Le nombre de trames par seconde étant grand (*>20*), nous n'avons pas besoin de toute l'information contenue dans ces trames.

La première opération que nous avons fait pour réduire le temps de calcul était de réduire le nombre de trames. Ensuite, nous avons utilisé la programmation parallèle sur des fonctions qui peuvent s'éxecuter simultanément, à titre d'exemple, les méthodes de segmentation sont éxecutées chacune sur un thread.

Et finalement nous avons utilisé les fonctions *GPU* implémentées dans *OpenCV*. Ces fonctions sont plus rapides que les fonctions implémenté sur le *CPU*.

Après avoir appliqué ces techniques, nous avons évalué notre système en termes de temps d'éxecution et nous avons obtenu les résultats suivants :

Type de caméra	$\frac{temps\ de\ traitement}{longueur\ de\ la\ vidéo} \times 100$
Canon	110%
Samsung S5	97%
Samsung GT	85%
Sony Xperia	99%

Tableau 11 : Temps de calcul réduit après utilisation *OpenMP*, *GPU*

Ces résultats suggèrent que l'atteinte du temps réel est automatiquement déterminée par les dimensions initiales de la séquence vidéo. Ainsi nous avons obtenu le temps réel avec des séquences vidéo qui ont des dimensions inférieures ou égales à 640*480, Ces résultats peuvent être améliorer avec une carte graphique plus puissante (nous avons utilisé une carte graphique avec 256 MB).

VI. Conclusion

Ce chapitre a été consacré à l'implémentation de notre approche. Ainsi on a vu en termes de programmation les différents outils et techniques mises en jeux.

Dans un premier lieu, nous avons présenté les bibliothèques OpenCv et FuzzyLite, que nous avons utlisées pour implémenter respectivement le système de vision et le système de décision.

Nous avons exposé par la suite les différentes conditions sous lesquelles nous avons testé notre système. Puis nous avons donné les différents résultats que nous avons obtenus.

Le temps réel a été une tâche cruciale pour notre système. Nous avons présenté les techniques que nous avons adoptées pour l'atteindre, à savoir la programmation parallèle et la programmation GPU.

Conclusion générale & Perspectives

Dans ce projet, nous avons développé une application pour assister les personnes déficientes visuelles dans leur locomotion. Notre approche se distingue par l'utilisation d'une seule caméra pour acquérir l'information sur l'environnement. Cela avait pour but de concevoir un système avec le maximum de portabilité et le moindre coût.

L'utilisation d'une seule caméra a des inconvénients. En effet, dans la projection de l'espace tridimensionnel sur le capteur bidimensionnel de la caméra, l'information sur la profondeur de la scène est perdue.

Nous avons divisé notre système en deux sous-systèmes. Le premier est un sous-système de de vision, qui est responsable de la détection des obstacles dans la séquence vidéo. Le deuxième est un sous-système de décision, qui est responsable de l'évitement des obstacles et qui a été implémenté en logique floue. La logique floue a pour but de remédier aux imprécisions des mesures fournies par le sous-système de vision, et qui sont dues à notre incapacité d'estimer la profondeur à partir de l'image bidimensionnelle.

Une fatalité que nous avons pu esquiver est la croissance exponentielle du nombre des règles dans le sous-système de décision, reconnue sous le nom de « *curse of dimensionnality* ». Nous avons pu surmonter cette difficulté par l'architecture hiérarchique de notre système qui l'imbrique en trois sous contrôleurs. Chaque contrôleur accomplit un nombre de tâches et alimente son successeur.

Des tests ont été menés pour examiner la performance de notre système. Un examen qui a été conduit en termes de trois critères à savoir le taux d'erreur, le

taux d'optimisation et le temps d'exécution. Plus encore, les vidéos utilisées pour ces tests proviennent de différents fournisseurs de caméras et de Smartphones, contiennent un nombre et formes variés d'obstacles et ont lieu aux espaces ouverts aussi bien qu'aux espaces fermés. Une diversité qui nous a permis d'enrichir l'environnement des tests, de détecter les défaillances éventuelles et aussi de procurer une certaine standardisation de notre système.

Notre motivation était de développer un système intelligent tout en minimisant les ressources matérielles recourues. Nous avons ciblé de développer un programme portable, autrement dit, implémentable sur un Smartphone ou sur un système embarqué quelconque. Par conséquent, nous avons dû consacrer une importante partie de notre travail à l'atteinte du temps-réel, d'où, nous avons utilisé la programmation parallèle et la programmation GPU dont les résultats se présentaient prometteurs et encourageants.

La locomotion des personnes déficientes visuelles, ou des personnes à mobilité restreinte, s'avère une série de tâches qui présentent plusieurs interférences techniques et introduit une intelligence considérable. Aussi, il s'agit d'un domaine névralgique susceptible de mettre la vie des gens en péril et de convaincre les gens à lui accorder une grande confiance pour que ses produits soient commercialisables, cette fiabilité est traduite par le taux d'erreur qui tend vers zéro. Pour y arriver, nous apprécions les perspectives de ce travail, de doter le système d'une phase d'apprentissage au niveau du sous-système de vision. Un apprentissage supervisé qui aura une connaissance à priori sur les environnements possibles pour la navigation. Cela peut améliorer les résultats du système, dans les environnements fréquemment parcourus.

Dans son intégralité le projet se voit comme un système autonome qui se charge intégralement de la locomotion. Comme le travail présent attaque la première partie informatique soft, Nous proposons de l'étendre et le combiner à une étude mécatronique pour instaurer une chaise roulante équipée d'un système embarqué. Puisque notre système est portable, il sera accueilli évidement par cette chaise roulante. Le système peut être ainsi utilisé dans les environnements fermés, comme les hôpitaux ou les universités, par tout individu intéressé.

Références

[1] «http://www.who.int/about/who-we-are/fr/,» [En ligne].

[2] «Wearable Obstacle Avoidance Electronic Travel Aids for Blind: A Survey Dimitrios Dakopoulos and Nikolaos G. Bourbakis, Fellow, IEEE».

[3] «Confédération Française pour la Promotion Sociale des Aveugles et Amblyopes : LES BE-SOINS DES PERSONNES DÉFICIENTES VISUELLES.».

[4] «Eric Schwitzgebel et Michael S. Gordon, « How Well Do We Know Our Own Conscious Ex-perience? - The Case of Human Echolocation »».

[5] «Neural Correlates of Natural Human Echolocation in Early and Late Blind Echolocation Experts : Lore Thaler, Stephen R. Arnott, Melvyn A. Goodale.».

[6] «The NavBelt - A Computerized Multi-Sensor Travel Aid for Active Guidance of the Blind J. Borenstein».

[7] «S. Meers and K. Ward, "A substitute vision system for providing 3D perception and GPS nav-igation via electro-tactile stimulation," presented at the 1st Int. Conf. Sens. Technol., Palmer-ston North, New Zealand, Nov. 21–23, 2005».

[8] «"Background subtraction techniques: a review - Systems, Man and Cybernetics, 2004 IEEE International Conference on - Piccardi.pdf."».

[9] «C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time track-ing of the human body. IEEE Trans. PAMI, 1997.».

[10] «C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. IEEE Proc. CVPR, 1999.».

[11] «Gibson. 1.1.. The Perception of the Visual World (Riverside Press, Cambridge. 1950).».

[12] «Gibson, J.J.. On the analysis of change in the optic array. Scandinavian I. Psychol. 18 (1977)».

[13] «Generalized image matching by the methode of differences, Lucas, Bruce David. Carnegie-Mellon University PH.D. 1985 ».

[14] «An Iterative Image Registration Technique with an Application to Stereo Vision. Bruce D. Lucas Takeo Kanade, From Proceedings of Imaging Understanding Workshop, pp. 121-130 (1981)».

[15] «Pyramidal Implementation of the AÆne Lucas Kanade Feature Tracker Description of the algorithm, Jean-Yves Bouguet, Intel Corporation Mi-croprocessor Research Labs 2010».

[16] «Canny, J., A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986».

[17]] «F. Mai, Y. Hung, H. Zhong, and W. Sze. A hierarchical approach for fast and robust ellipse extraction. Pattern Recognition, 41(8):2512–2524, August 2008».

[18] « Advances in Fuzzy Systems-Applications and Theory Vol 6. Fuzzy Sets, Fuzzy Logic and Fuzzy Systems. Selected Papers by Lotfi A. Zadeh, G.J. Klir and B. Yuan (eds), 433-448. Singapore: World Scientific, 1996.».

[19] «Advances in Fuzzy Set Theory and Applications, M. Gupta, R. Ragade and R. Yager (eds.), 3-18. Amsterdam: North-Holland Publishing Co., 1979».

[20] «D. Marr. Vision: A Computational Investigation into the Human Representation and Processing of».

[21] «R. Wehner. The polarization-vision project: Championing organismic biology. In K. Schildberger».

[22] «Y. Aloimonos. Active Vision Revisited. In Active Perception. Lawrence Erlbaum Associates, 1993.».

[23] «Patrern Recognition. 4-10 November 1978. Kyoto. Japan. 27. Nakayama. K. and laomis. J.M.. Optical velocity patterns. velocity-sensitive neurons and space perception. Per-ception 3 (1974) 63-80.».

[24] «: Confédération Française pour la Promotion Sociale des Aveugles et Amblyopes : LES BESOINS DES PERSONNES DÉFICIENTES VISUELLES.».

[25] «Avoiding Exponential Parameter Growth in Fuzzy Systems, Mustafa K. Güven and Kevin M. Passino».

[26] «Modeling of hierarchical fuzzy systems, Ming-Ling Lee, Hung-Yuan Chung∗, Fang-Ming Yu».

[27] «A Hierarchical Fuzzy System with High Input Dimensions for Forecasting For-eign Exchange Rates, France Cheong».