République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie d'Oran

Mohamed BOUDIAF

Faculté de génie électrique

Département d'électronique

Mémoire

Présenté par

M. Houcine ZERFA

Pour obtenir

Le Titre de Magister en Automatique Robotique Productique

Sujet de mémoire :

Conception, Réalisation et Commande Floue d'un Robot Mobile

Soutenu le 19 Mars 2013 devant la commission d'examen composée de :

Président	: M. Abdelhamid MIDOUN	Professeur/USTO-MB
Encadreur	: M. Wahid NOUIBAT	Maître de conférences/USTO-MB
Examinateur	: M. Abdelhamid LOUKIL	Maître de conférences/USTO-MB
Examinateur	: M. Mohamed OUSLIM	Maître de conférences/USTO-MB

Année universitaire 2012-2013

Remerciements

Ce travail a été effectué au sein du département d'électronique, magistère en Automatique Robotique Productique. Je tiens avant tout à remercier mon directeur de projet Monsieur Wahid NOUIBAT. Ses conseils et son optimisme m'ont été d'une aide précieuse tout au long de ce projet. Je le remercie pour la liberté qu'il m'a laissée dans mes recherches et pour tout le temps qu'il m'a consacré lors de l'écriture de ce mémoire. Merci également à Monsieur Zoubir Ahmed FOUATIH de m'avoir aidé. Ses conseils lors de nos entrevues ont toujours été fructueux. Je tiens également à remercier les membres de mon jury, Monsieur Abdelhamid MIDOUN pour l'honneur qu'il me fait de le présider, Monsieur Abdelhamid LOUKIL et Monsieur Mohamed OUSLIM pour l'intérêt qu'ils ont porté à mon travail en acceptant d'être examinateurs, leurs remarques et suggestions m'ont permis d'améliorer la qualité de mon travail. Je suis très reconnaissant envers Djamel BENYELHA, Moussa RIGHI et Akram ADNANE, de m'avoir aidé. Je remercie également mes coéquipiers de travail qui partageaient le même laboratoire. Remerciements à tous ceux qui ont contribués de près ou de loin à l'élaboration de ce travail.

A la mémoire de mon frère Noureddine, à mes parents, à mon frère Mohammed, pour la patience et le dévouement dont ils ont fait preuve.

Sommaire

Table des figures

Chapitre 1

Chapitre 2

Chapitre 3

Chapitre 4

Liste des algorithmes

Notations et abréviations

θ	◊	Angle de braquage d'un robot uni-cycle
n	◊	Nombre d'impulsion fourni par un codeur optique
α	◊	Résolution d'un codeur optique
Δt	◊	Intervalle de temps
Δd	◊	Distance parcourue par une roue
R	◊	Rayon d'une roue
R0	◊	Repère de référence (X, Y)
Rr	◊	Repère du robot (Xr, Yr)
M	◊	Origine du repère du robot (Xr, Yr)
ΔD	◊	Déplacement élémentaire du robot
$\Delta\theta$	◊	Orientation élémentaire du robot
Δdd	◊	Déplacement élémentaire de la roue droite du robot
Δdg	◊	Déplacement élémentaire de la roue gauche du robot
E	◊	Entraxe du robot (la distance entre les deux roues du robot)
PMC	◊	Perceptron Multicouche
RBF	◊	Radial Basis Function (fonctions de base radiale)
ANFIS	◊	Adaptive network based fuzzy inference system
$\mu(x)$	◊	Degré d'appartenance
U	◊	Univers de discourt (référence)
Γ	◊	Fonction demi trappez droit
L	◊	Fonction demi trappez gauche
S	◊	Fonction sigmoïde
Λ	◊	Fonction triangle
c	◊	Centre d'une fonction gaussienne
σ	◊	Écart type d'une fonction gaussienne
$\cap$	◊	Intersection (minimum)
U	◊	Union (maximum)
Σ	◊	Addition
w_i	◊	Poids synaptique d'un neurone
in	◊	Entrée
out	◊	sortie
hid	◊	caché
N	◊	Négatif
P	◊	Positif

Z	◊	Environ zéro
PV	◊	Petite vitesse
GV	◊	Grande vitesse
MV	◊	Moyenne vitesse
φ	◊	Angle flou
V	◊	Vitesse
cap1	◊	Capteur frontale
cap2	◊	Capteur droit
cap3	◊	Capteur gauche
Arg	◊	Argument (calcul de l'angle)
Mod	◊	Module (distance entre deux points dans l'espace à 2D)
k_{ij}	◊	Paramètres des conséquents
$\prod_i$	◊	Produit des degrés d'appartenances
N_i	◊	Degrés normalisés
y_i	◊	Sortie i du neurone i de la couche 4 du réseau ANFIS
y	◊	Sortie totale du réseau ANFIS
e	◊	erreur
E_T	◊	Erreur totale
Δc	◊	Variation du centre de la gaussienne
Δa	◊	Variation de l'écart type de la gaussienne
η	◊	Pas d'apprentissage
δE_T	◊	Gradient de l'erreur totale
Yd	◊	Sortie désirée
VRML	◊	Virtual Reality Modeling language
PID	◊	Proportionnel Intégrateur Dérivateur
PWM	◊	Pulse Width Modulation (modulation de largeur d'impulsion)
T0CKI	◊	Timer 0 with an External Clock (Timer 0 avec une horloge externe)
T1CKI	◊	Timer 1 with an External Clock (Timer 1 avec une horloge externe)
RF	◊	Radio fréquence
TX	◊	Transmitter (Émetteur)
RX	◊	Receiver (Récepteur)
IDE	◊	Environnement de développement intégré
IHM	◊	Interface Homme Machine
ADCON1	◊	Analogical digital control register 1
AD	◊	Résultat de la conversion analogique – numérique
Gs	◊	Gain statique
τ	◊	constante de temps (le retard)
ω	◊	Vitesse de rotation
V	◊	Vitesse de translation
V_{droite}	◊	Vitesse de translation de la roue droite
V_{gauche}	◊	Vitesse de translation de la roue gauche
D	◊	Distance entre le robot et le point d'arrivée

A.

Préface

Ce document présente la conception, la réalisation et la commande floue d'un robot mobile. La problématique plus spécifique de la navigation et un état de l'art des approches existantes sont exposés au début de ce mémoire. **Le premier chapitre** est une introduction générale à la robotique mobile à roues. Il présente les aspects de perception, de localisation et de contrôle des robots autonomes. **Le second chapitre** résume toutes les notions de base nécessaires à la compréhension du domaine de la logique floue, ainsi que les outils nécessaires à la mise en œuvre d'un système d'inference flou, nous présentons aussi, dans ce chapitre, le système d'optimisation neuro-flou (ANFIS), ainsi que le navigateur flou de notre robot avant et après optimisation. La réalisation de la plateforme de simulation, ainsi les différents résultats de simulation de la navigation réactive floue de notre robot sera l'objet du **troisième chapitre**. Enfin **le dernier chapitre** présente la conception et la réalisation (mécanique, électronique et informatique) de notre robot nommé ARP. Nous présentons aussi l'implémentation du navigateur flou présenté déjà au deuxième chapitre, et nous effectuons des tests pratiques sur les différents comportements réactifs du robot ARP, des résultats sont finalement présentés. Nous terminons le mémoire par une conclusion générale sur l'ensemble de cette étude et nous proposons des perspectives à notre travail.

B.

Problématique et état de l'art

La robotique mobile cherche depuis des années à rendre une machine mobile autonome face à son environnement pour qu'elle puisse sans intervention humaine accomplir les missions qui lui sont confiées. Le spectre des missions que les roboticiens veulent voir accomplir par leurs machines est immense : exploration en terrain inconnu [Yang], manipulation d'objets [D. Hong], assistance aux personnes handicapées [C. Biihler], transport automatisé, etc. De grand progrès ont été accomplis dans tous les domaines de la robotique : perception et modélisation de l'environnement [Yang], commande automatique des actionneurs, navigation et planification de mouvements, ordonnancement de tâches, gestion de l'énergie, etc.

Dans le cadre de la navigation d'un robot mobile dans son environnement c'est-à-dire, la capacité d'aller d'une position initiale A à une position finale B de manière autonome, le problème général de navigation consiste à déterminer pour le robot un mouvement lui permettant de se déplacer entre ces deux configurations données tout en respectant un certain nombre de contraintes et de critères. Ceux-ci découlent de plusieurs facteurs de natures diverses et dépendent généralement des caractéristiques du robot, de l'environnement et du type de tâche à exécuter. En l'occurrence, les contraintes relatives au robot concernant sa géométrie, sa cinématique et sa dynamique, les contraintes émanant de l'environnement concernant essentiellement la non-collision avec les obstacles encombrant l'environnement et la prise en compte d'interactions de contact avec le robot. L'évitement d'obstacles dépend de la géométrie de l'environnement et est commun à toutes les tâches robotiques.

Historiquement, le problème de navigation a été initialement abordé dans le cadre d'un système de robots se déplaçant dans un environnement contenant des obstacles fixes et soumis à la seule contrainte de non-collision. Cette instance du problème général a fait l'objet de plusieurs travaux. Deux grandes familles de méthodes permettent à un robot de se déplacer d'une configuration A à une configuration B tout en évitant les obstacles : Les **méthodes réactives** [Valentino][Brooks 3] (champs de potentiels [Khatib][Arkin 3], réseaux de neurones [Lebedev][Belker][Wang][T. Bräunl] et logique floue [W.L. Xu] [Motlagh]

[Foudil] [Ouadah], etc.), ce type d'approches, pour résoudre le problème de navigation d'un robot mobile autonome, est de rendre la configuration finale (cible) du robot attractive pour celui-ci, de manière à ce qu'il cherche à s'en rapprocher. Et inversement rendre les obstacles répulsifs, pour qu'il s'en écarte et reste à bonne distance. Dans cette approche, on ne cherche pas à anticiper la trajectoire du robot, on le laisse librement décider à chaque instant dans quelle direction il doit se rendre, à partir des informations recueillies concernant son environnement local. On fait l'hypothèse que contrôler son comportement général plutôt que de chercher à calculer une trajectoire, permettra tout de même au robot d'atteindre son but au final. L'autre famille celles des **méthodes de planification de trajectoire** [Steven M 1] (décomposition cellulaire [Buhmann][Arleo][Thrun][Latombe], Roadmaps [Laumond][Steven M 2], etc.), qui permettent au robot de suivre "du mieux possible" une trajectoire de référence donnée, ces méthodes sont généralement basées sur des cartes qui permettent au robot de prendre en compte des buts à long terme en utilisant des informations mémorisées sur la structure de son environnement.

C.

Objectifs du projet

Il s'agit de la conception, de la réalisation et la commande floue d'un robot mobile. L'objectif est de mettre en place un navigateur flou réactif qui permet au robot de se déplacer d'une configuration A à une configuration B dans un environnement inconnu de terrain plat a deux dimension (x, y) d'une dizaine de mètre carré tout en assurant la non collision avec les obstacles. Le navigateur flou réactif doit gérer la vitesse et l'orientation du robot.

Un des problèmes inhérent aux méthodes réactives par logique floue est qu'elles souffrent de problèmes d'adaptation à des situations différentes. Cependant il existe des méthodes dites d'apprentissage on-line qui permettent au robot de modifier lui même ses comportements au fur et à mesure qu'il explore son environnement [Er M.J]. Le problème de ces méthodes est qu'elles demandent un temps d'apprentissage plus ou moins long avant que le robot ne puisse naviguer efficacement, et que ce type d'apprentissage nécessite généralement un calculateur embarqué sur le robot de capacité mémoire et calcul importants. Pour remédier à ce genre de problème, ils existent d'autres méthodes d'apprentissage dites off-line qui permettent de collectionner des données de référence et déduire ensuite le modèle qui sera

implémenté dans le robot, cette méthode est rapide et simple au niveau de l'implémentation et ne requiert pas de grands moyens matériels. En effet, il faut donc optimiser notre navigateur flou réactif par une méthode d'apprentissage neuro-flou off-line, le but de cet apprentissage est d'affiner en quelque sorte la sortie du navigateur flou réactif.

Afin de bien répondre à notre objectif, le navigateur flou réactif réalisé doit être testé en toute sécurité, il faut donc réaliser une plateforme de simulation. Le principal inconvénient de la simulation est que l'environnement est moins réaliste, et donc que les algorithmes évolués ainsi sont peu susceptibles de fonctionner en environnement réel. C'est pourquoi il faudra également réaliser un robot réel et effectuer des expériences sur ce robot afin de valider notre approche. Les expérimentations consistent tout d'abord à implémenter le navigateur flou réactif déjà optimisé. Il s'agit ensuite de définir une mission de navigation que le robot doit exécuter, et d'évaluer dans quelle mesure le robot remplit sa mission. Certains critères de réalisation doivent être prises en compte, le robot doit en outre faire preuve d'une autonomie suffisante pour pouvoir explorer le terrain et trouver son chemin, et doit être capable de se localiser en permanence dans l'environnement qu'il explore (le robot mobile doit être équipé d'un système de localisation et de détection d'obstacles).

Chapitre 1

Sommaire partiel 1:

1

Introduction à la robotique mobile

1.1 Notions de robotique mobile :

Bien souvent, quand on parle de robotique mobile, on sous entend robots mobiles à roues. Ce sont en effet les systèmes les plus étudiés, parce qu'ils sont plus simples à réaliser que les autres types de robots mobiles, ce qui permet d'en venir plus rapidement à l'étude de leur navigation. Ce type de robots est notamment très souvent utilisé pour l'étude des systèmes autonomes.

L'appellation " robot mobile " désigne généralement un véhicule équipé de capacités de perception, de décision et d'action qui lui permettent d'agir de manière autonome ou semi-autonome dans un environnement complexe, parfois évolutif, partiellement connu ou inconnu, et d'exécuter les tâches programmées sans intervention humaine ou avec une intervention réduite.

Il existe deux principaux modes de fonctionnement pour un robot mobile : télé-opéré et autonome. En mode télé-opéré [H. Zerfa], une personne pilote le robot à distance. Elle donne ses ordres via une interface de commande (joystick, clavier…), et ceux-ci sont envoyés au robot via un lien de communication (internet, satellite …). Le robot doit donc obéir aux ordres de l'opérateur qui perçoit l'environnement autour du robot, par différents moyens (camera, radar…), de manière à donner des ordres adaptés au robot. A l'inverse, en mode autonome [Bilgic] [R. SIEGW] le robot doit prendre ses propres décisions. Cela signifie qu'il doit être capable à la fois de percevoir correctement son environnement, mais également de savoir comment réagir en conséquence, suivant le niveau d'autonomie. C'est à lui d'envisager son parcours et de déterminer avec quels mouvements il va atteindre son objectif.

Cette notion d'autonomie prise en exemple ci-dessus, que nous pourrions qualifier de décisionnelle, ne doit pas être confondue avec celle d'autonomie énergétique (capacité du robot à gérer efficacement son énergie, à la préserver, voire à se ravitailler), même si ces deux notions sont étroitement liées : idéalement une des préoccupations principales d'un robot mobile totalement autonome (du point de vue décisionnel), serait en effet de pouvoir gérer de lui-même ses réserves d'énergie.

Voyons maintenant les différents types de robots mobiles à roues.

1.1.1 Classes des robots mobiles à roues :

Pour déplacer un robot mobile sur une surface, il faut au moins deux degrés de liberté, donc deux moteurs. Et c'est aussi la combinaison du choix des roues et de leur disposition qui confère à un robot son mode de locomotion propre [R. SIEGW] [T. Bräunl] (voir figure 1.1), on rencontre principalement trois types de robot :

- Robot uni-cycle : Actionné par deux roues indépendantes et possédant éventuellement un certain nombre de roues folles assurant sa stabilité ;

- Robot tricycle (et de type voiture qui partagent des propriétés cinématiques proches) : Constitué de deux roues fixes de même axe et d'une roue centrée orientable placée sur l'axe longitudinal du robot. Le mouvement est conféré au robot par deux actions (la vitesse longitudinale et l'orientation de la roue orientable) ;

- Robot omnidirectionnel : Un robot mobile est dit omnidirectionnel si l'on peut agir indépendamment sur les vitesses (vitesse de translation selon les axes x et y et vitesse de rotation autour de z). D'un point de vue cinématique il n'est pas possible avec des roues fixes ou des roues centrées orientables. On peut en revanche réaliser un robot omnidirectionnel en ayant recours à un ensemble de trois roues décentrées orientables ou de trois roues suédoises disposées aux sommets d'un triangle équilatéral.

1.1.2 L'holonomie :

En robotique mobile, une plateforme est dite holonome lorsque le nombre de degrés de libertés contrôlables est égal au nombre total de degrés de liberté. Pour un robot se déplaçant sur un plan, il y a 3 degrés de liberté (deux translations et une rotation). A partir d'une position donnée, une plateforme holonome devra donc pouvoir se déplacer en avant, sur le coté et tourner sur elle-même. Cette capacité permet de contrôler très simplement le robot car tous les déplacements imaginables sont réalisables, Exemple (le cas des robots omnidirectionnels).

La relative simplicité mécanique des robots non-holonomes face aux robots holonomes à un prix. Les algorithmes d'exécution de navigation, de commandes et de planification de mouvements sont plus complexes. Mais leur facilité de construction présente un intérêt économique évident.

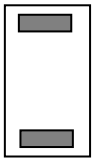
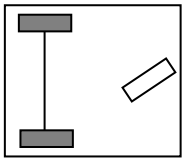
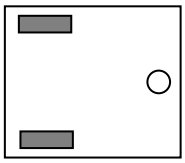
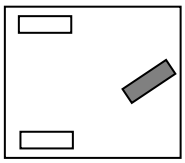
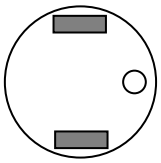
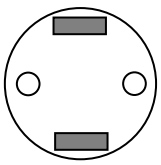
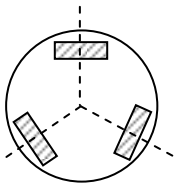
	1. Uni-cycle, avec deux roues motrices différentielles exemple : Cye personal robot.
	2. Tricycle, deux roues motrices en arrière, et une roue dirigée à l'avant exemple : Piaggio minitrucks.
	3. Uni-cycle, deux roues motrices différentielles en arrière, et une roue libre à l'avant exemple : Pygmalion et Alice.
	4. Tricycle, deux roues libres en arrière, une roue de traction dirigée à l'avant exemple : Neptune, Hero-1.
	5. Uni-cycle, deux roues motrices différentielles centrées, et un point de contacte a l'avant exemple : Nomad Scout, smartRob.
	6. Uni-cycle, deux roues motrices différentielles centrées, et deux points de contacte avant et arrière exemple : EPFL Khepera, Hyperbot Chip.
	7. Omnidirectionnel, trois roues suédoises motorisées exemple : Tribolo EPFL, Palm Pilot Robot Kit.

Figure 1.1 : Classes des robots mobiles à roue [R. SIEGW].

1.1.3 La localisation en robotique mobile, techniques et capteurs :

Quelque soit le type de robot, holonome ou pas, la connaissance de sa position dans son environnement est importante pour s'y déplacer. Souvent, la localisation se restreint aux robots mobiles navigants sur un plan 2D. Localiser le robot revient alors à déterminer trois paramètres : deux coordonnées cartésiennes pour la position et un angle pour l'orientation. De façon plus formelle, la tâche de localisation consiste à calculer la transformation de passage d'un repère lié au robot à un repère lié à l'environnement.

Le système de localisation est l'ensemble constitué par les capteurs et les logiciels de traitement de données utilisé par le véhicule pour estimer de manière autonome son déplacement ou sa situation dans l'espace. Classiquement, on distingue deux types de solutions [Pruski]:

- La localisation relative qui permet au véhicule de naviguer à l'estime (Dead reckoning) en utilisant uniquement les mesures de ses mouvements propres délivrées par les capteurs proprioceptifs ;

- La localisation absolue qui fait appel aux mesures des capteurs extéroceptifs pour estimer la situation du véhicule dans un repère lié à l'environnement.

Les capteurs [Bilgic] [Borenstein] [T. Bräunl] constituent les éléments fondamentaux du système de localisation. On peut les classer en fonction du type de localisation associé:

- Les capteurs extéroceptifs (capteur d'environnement) délivrent une information relative à l'environnement (reconnaissance, modèle) ou aux interactions entre le robot et son environnement (position, force). Exemple de capteurs extéroceptifs : (Camera, Sonars, infrarouge, Détecteur de lumière, Boussole, GPS, Détecteur de chaleur…) voir figure 1.2;

Figure 1.2 : Capteurs extéroceptifs (ultrason et infrarouge).

- Les capteurs proprioceptifs [G. Frappier] Il s'agit des capteurs qui délivrent les informations sur l'état (interne) du robot. Pour un robot mobile, il peut s'agir également des capteurs de vitesse, d'accélération..., Exemple de capteurs

proprioceptifs [T. Bräunl]: (Gyrocompas, Accéléromètres, Inclinomètre, Odomètre…) voir figure 1.3.

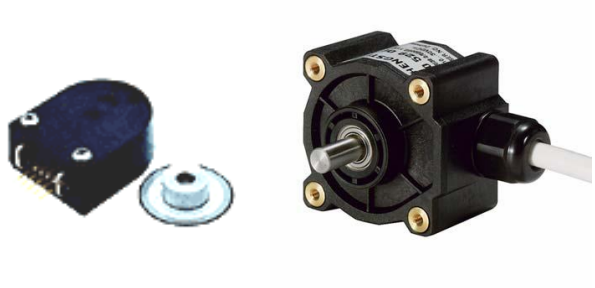


Figure 1.3 : Capteurs proprioceptifs (odomètres).

1.1.3.1 Mesure des mouvements dans un plan :

Parmi les différentes méthodes et technologies de mesure de mouvement nous nous intéresserons dans ce chapitre aux codeurs optiques (odomètres) [T. Bräunl]. Les systèmes optiques sont les plus répandus en robotique mobile. Le codeur optique rotatif fonctionne selon le principe du balayage photoélectrique d'un disque sur lequel a été déposé un réseau radial plus ou moins complexe de trous (ou de traits opaques). Lorsque ce disque effectue une rotation, le flux lumineux émis par une diode électroluminescente est modulé et les cellules photoélectriques qui captent la lumière à la sortie du disque délivrent un signal carré.

Le codeur incrémental est le capteur le plus utilisé pour mesurer les variations de position et de l'orientation d'un robot mobile à roues car il est peu onéreux et facile à interfacer. Monté sur l'axe du moteur ou sur l'axe de la roue, il délivre des informations de rotation élémentaire qui, par intégration, donnent une mesure du mouvement global voir figure 1.4.

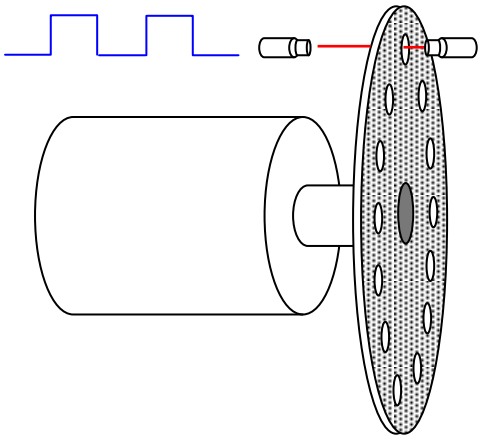


Figure 1.4 : Principe du codeur incrémentale.

1.1.3.1.1 L'Odométrie :

L'odométrie est certainement la méthode de localisation relative [N. MORET] la plus couramment employée pour les robots disposant d'une structure de locomotion à roues. Son principe consiste à déduire une position, de façon incrémentale, à partir de la vitesse et de la géométrie des roues. Cependant, l'odométrie est fragile. Tout d'abord, elle nécessite une connaissance précise de la géométrie du robot (le diamètre des roues doit être déterminé, mais d'autres dimensions, comme l'entraxe[1] ou le point de contact avec le sol, doivent également être prises en compte). Ces paramètres sont généralement difficiles à obtenir de façon précise et, dépendent fortement de la nature du sol. De plus, les cas pathologiques sont rarement détectables (les glissements) sont par exemple dramatiques pour l'estimation de la position et ils ne peuvent en général pas être mesurés directement. Selon le type de terrain ils peuvent rendre quasiment inutile l'odométrie en tant que méthode de localisation.

L'odométrie permet de déterminer la position *(x, y)* et le cap *(θ)* d'un robot mobile navigant sur un sol plat, par rapport au repère de référence qui était celui du robot dans sa configuration initiale. Cette technique est basée sur l'intégration des mouvements élémentaires des roues mesurés grâce à des codeurs incrémentaux. Connaissant le rayon *(R)* de la roue et le nombre *(n)* d'impulsions délivrées par le codeur de résolution *(α)* durant un intervalle de temps *(Δt)*, il est possible de calculer la distance *(Δd)* parcourue par cette roue : $\Delta d = R*n*\alpha$. *(1)*

1.1.3.1.2 Modèle d'évolution d'un robot à roues différentielles :

Le modèle d'évolution d'un robot terrestre [M. ALDON] exprime son déplacement dans le repère de référence *(R0)* en fonction du mouvement de ses roues. Considérons le cas d'un robot de type uni-cycle muni de deux roues motrices indépendantes, diamétralement opposées et dont l'axe commun porte l'origine *(M)* du repère *(Rr)* attaché au robot (figure 1.5). Ce système de locomotion (roues différentielles) est très répandu pour les robots d'intérieur en raison de sa manœuvrabilité et de sa simplicité de commande. Dans ce cas, le déplacement élémentaire *(ΔD)* et la rotation élémentaire *($\Delta\theta$)* du robot dans le plan s'expriment en fonction des déplacements élémentaires des roues droite et gauche, respectivement *(Δdd)* et *(Δdg)*, par:

- $\Delta D = (\Delta dd + \Delta dg)/2$ *(2)*
- $D\theta = (\Delta dd - \Delta dg)/E$ *(3)*

Avec E l'entraxe du véhicule.

[1] Entraxe : Distance qui sépare les deux roues d'un robot mobile uni-cycle.

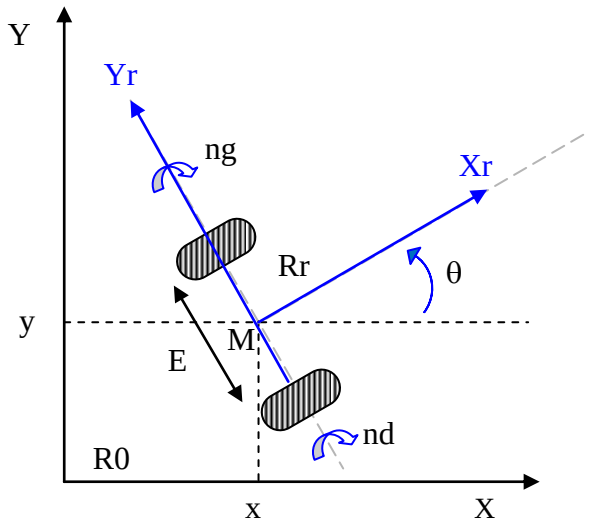


Figure 1.5 : Paramètres d'un robot à roues différentielles.

Désignons par (x_k, y_k, θ_k), la configuration du véhicule à l'instant (k), et par $(\Delta D_k, \Delta\theta_k)$ les composantes du déplacement élémentaire mesuré entre les instants (k) et $(k + 1)$ (voir figure 1.6). Le cap à l'instant $(k + 1)$ est :

- $\theta_{k+1} = \theta_k + \Delta\theta_k$ *(4)*

Pour estimer les coordonnées (x_{k+1}) et (y_{k+1}) du robot à l'instant $(k+1)$, il faut émettre des hypothèses sur la trajectoire du robot entre les deux instants d'échantillonnage. Selon les hypothèses choisies, les équations obtenues sont plus ou moins complexes.

Les formules les plus simples sont obtenues en considérant que le robot se déplace en ligne droite [M. ALDON] de (ΔD_k) dans la direction définie par (θ_k), puis effectue une rotation sur place de $(\Delta\theta_k)$:

- $x_{k+1} = x_k + \Delta D_k * \cos\theta_k$ *(5)*
- $y_{k+1} = y_k + \Delta D_k * \sin\theta_k$ *(6)*

En faisant l'hypothèse que la trajectoire du robot est localement assimilable à un arc de cercle de longueur (ΔD_k), tangent en (M_k) à la direction définie par (θ_k), et en (M_{k+1}) à la direction définie par le nouveau cap (θ_{k+1}), on obtient:

- $x_{k+1} = x_k + \Delta D_k * [(\sin(\Delta\theta_k / 2)) / (\Delta\theta_k/2)] * \cos[\theta_k + (\Delta\theta_k/2)]$ *(7)*
- $y_{k+1} = y_k + \Delta D_k * [(\sin(\Delta\theta_k / 2)) / (\Delta\theta_k/2)] * \sin[\theta_k + (\Delta\theta_k/2)]$ *(8)*

Cependant, lorsque la trajectoire est assimilable à une ligne droite, $(\Delta\theta_k)$ est infiniment petit. Il faut alors simplifier ces équations pour éviter les problèmes numériques.

Pour s'affranchir de ces problèmes, on préfère utiliser une troisième hypothèse [M. ALDON] qui consiste à approximer l'arc de cercle par sa corde. On obtient alors le modèle suivant qui est l'un des plus utilisés :

- $x_{k+1} = x_k + \Delta D_k * cos\,[\theta_k + (\Delta\theta_k/2)]$ *(9)*
- $y_{k+1} = y_k + \Delta D_k * sin\,[\theta_k + (\Delta\theta_k/2)]$ *(10)*

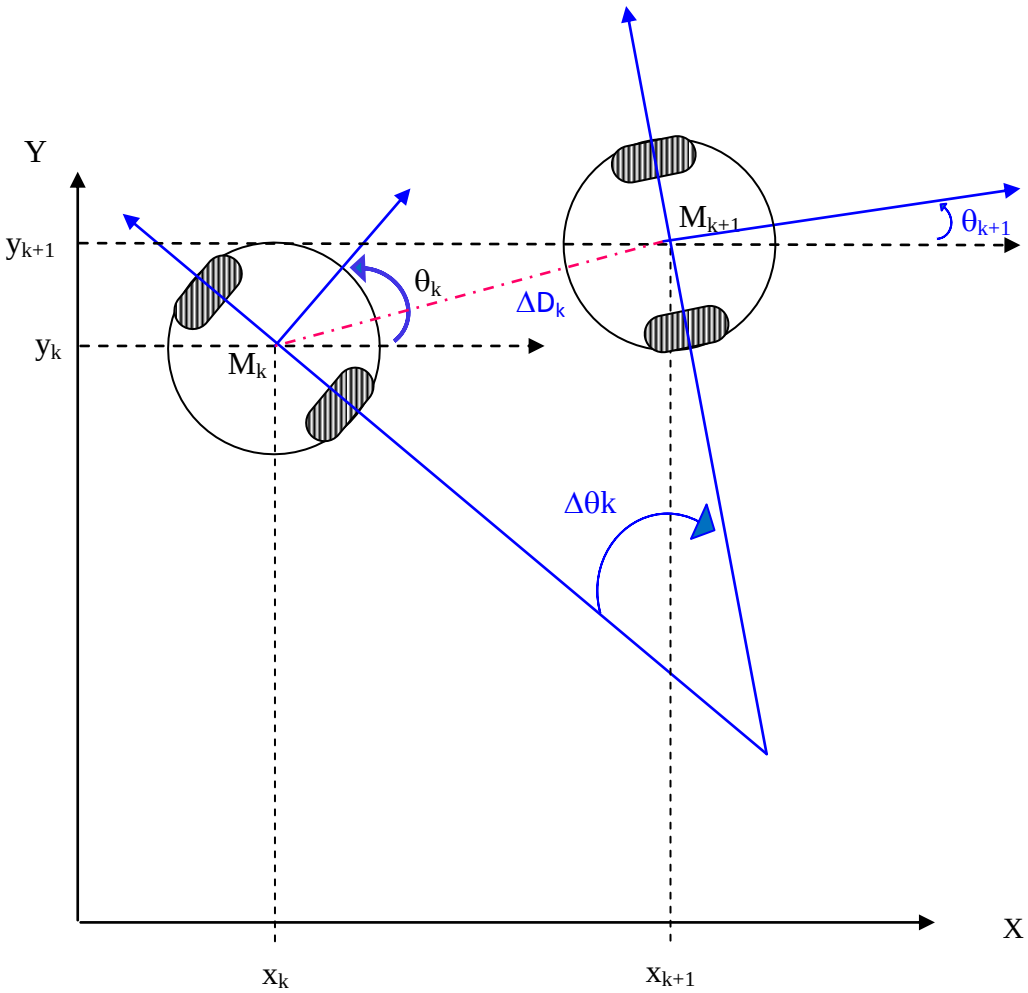


Figure 1.6: Modèle d'évolution d'un robot à roues différentielles.

L'odométrie permet d'obtenir une estimation en temps réel de la position et du cap. Cependant, dès que la distance parcourue augmente, des erreurs d'origines diverses s'accumulent. Elles résultent de l'imprécision introduite par les codeurs sur la mesure des déplacements angulaires des roues et d'une modélisation approximative de l'évolution du robot mobile. En effet, nous avons vu que les équations cinématiques utilisées sont généralement simplifiées. Par ailleurs, elles ne prennent pas en compte un certain nombre de phénomènes physiques difficilement modélisables tels que le

dérapage des roues. Le modèle d'évolution est basé sur des hypothèses qui ne sont jamais rigoureusement vérifiées. On suppose généralement que :

- Les deux roues ont le même rayon *(R)* ;
- Les paramètres géométriques *(R)* et *(E)* sont constants et parfaitement connus.

En pratique, le rayon *(R)* peut varier et être différent pour les deux roues (gonflage différent des pneus, déformations dues à la charge, usure). Si la surface de roulement n'est pas parfaitement plane, les coordonnées dans le repère robot *(Rr)* des points de contact des roues avec le sol varient, et dans ce cas, *(E)* n'est pas constant. Les phénomènes de patinage et de glissement se traduisent par le fait que le vecteur de vitesse du point *(M)* n'est pas toujours dirigé suivant l'axe *(Xr)* du repère robot, contrairement à l'hypothèse admise dans le modèle d'évolution.

Cependant, une analyse détaillée de l'influence des différentes sources d'erreur montre que les erreurs dues à la simplification du modèle sont généralement négligeables par rapport à celles provoquées par le bruit de mesure des déplacements des roues. L'erreur due au modèle est d'autant plus faible que le mouvement élémentaire $(\Delta D_k, \Delta\theta_k)$ effectué entre deux mesures est petit.

1.2 Navigation et architectures de contrôle:

1 .2.1 La navigation:

Dans la littérature classique, le problème de navigation est souvent formulé comme un problème de calcul de chemin orienté vers un but, problème où la représentation de l'environnement à partir des informations sensorielles apparait comme essentielle.

Pour exécuter une tâche de navigation autonome, un robot mobile doit mettre en œuvre un certain nombre de fonctionnalités, le robot doit se localiser dans l'environnement, et doit aussi percevoir les différents obstacles autour de lui et calculer un chemin sans collision pour atteindre son but (voir figure 1.7). On peut constater que les deux familles de capteurs présentées précédemment (proprioceptifs pour la localisation et extéroceptifs pour l'évitement d'obstacles) sont plus complémentaires que concurrentes. C'est pour cette raison qu'un système de navigation sera généralement basé sur ces deux familles de capteurs.

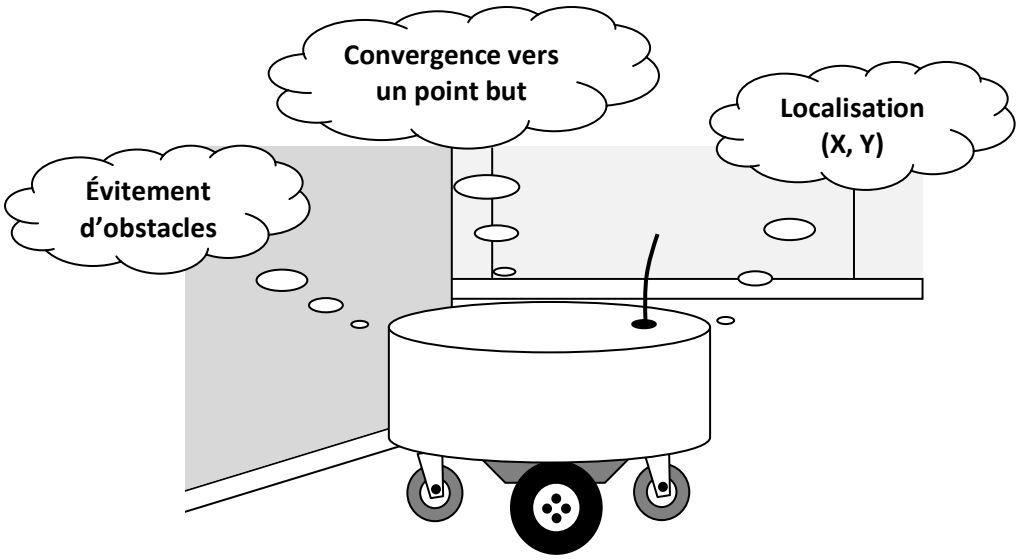


Figure 1.7 : Robot mobile en phase de navigation.

La navigation [T. Bräunl] peut être soit prédéterminée (la planification est effectuée avant que le robot n'entame son mouvement), soit réactive (la modification de la trajectoire s'effectue en temps réel au cours du déplacement, à partir du retour sensoriel). Nous nous intéressons dans cette section seulement à la navigation réactive.

1.2.1.1 La navigation réactive :

La navigation réactive [Song] [P. Reignier] utilise des actions réflexes pour guider le robot et se différencie essentiellement par le type de perceptions utilisées pour déclencher ces actions. Elle n'utilise que les valeurs courantes des capteurs, pour décider de l'action à effectuer. Le principe consiste à engendrer progressivement des incrémentations de déplacements pour permettre de se rapprocher du but en utilisant une information locale sur les contraintes imposées par l'environnement. Cette information peut être issue, par exemple, de capteurs de distances ou de capteurs tactiles ou encore, directement, par un algorithme de calcul de distances. On reconnaît à cette méthode l'avantage d'être rapides pour l'obtention d'une solution, elle permet de réaliser des tâches de bas niveau, comme l'évitement des obstacles imprévus, essentielles à la sécurité d'un robot. Il est à noter que ce type de navigation ne requiert aucune modélisation spatiale (aucune phase de planification). On peut citer deux sous classes :

1.2.1.1.1 La Navigation vers un but visible (objet) :

Cette capacité de base permet de se diriger vers un objet visible depuis la position courante du robot. Elle est en général réalisée par une remontée sur la perception de l'objet, comme dans l'exemple célèbre des véhicules de Valentino Braitenberg [Valentino] qui utilise deux capteurs de lumière pour atteindre ou fuir une source lumineuse. Cette stratégie utilise des actions réflexes, dans lesquelles chaque perception est directement associée à une action. C'est une stratégie locale, c'est-à-dire fonctionnelle uniquement dans la zone de l'environnement pour laquelle le but est visible.

1.2.1.1.2 La Navigation vers un point dans l'espace:

Cette capacité permet d'atteindre un but qui n'est pas un objet matériel directement visible, mais un point de l'espace caractérisé par la configuration spatiale d'un ensemble d'objets remarquables, ou amers, qui l'entourent ou qui en sont voisins. La stratégie de navigation consiste souvent à se diriger dans la direction qui permet de reproduire cette configuration. Cette capacité semble utilisée par certains insectes, comme les abeilles [Cartwright], et a été utilisée sur divers robots [Gaussier] [Lambrinos] [Gourichon]. Cette stratégie utilise également des actions réflexes et réalise une navigation locale qui requiert que les amers caractérisant le but soient visibles.

1.2.2 Architectures de contrôles :

Un robot est donc composé d'un ensemble de modules [Mataric], chacun étant responsable d'une ou de plusieurs capacités. L'un des premiers défis à résoudre est de déterminer comment relier efficacement les différents modules. Pour ce faire, il faut élaborer une architecture décisionnelle qui dicte les responsabilités de chacun des modules et comment l'information circule entre ces derniers. Depuis les débuts de la robotique, beaucoup d'architectures ont été proposées. Elles peuvent être généralement classées en trois grandes catégories : délibérative, comportementale et hybride.

1.2.2.1 Architecture délibérative (hiérarchique) :

Les architectures délibératives [Collin] sont les premières à avoir été proposées. Comme son nom l'indique, les architectures de ce type sont basées sur des processus complètement planifiés. Par exemple, afin d'exécuter un déplacement, un robot basé sur ce type d'architecture calcule un plan complet, lui disant d'avancer de x mètres, ensuite de tourner de y degrés, et ainsi de suite. Lorsqu'un changement dans l'environnement est perçu, l'exécution est suspendue et un nouveau plan est généré.

Ce type d'architecture souffre de plusieurs lacunes importantes. Premièrement, puisque les capteurs sont imprécis et que l'environnement est dynamique et partiellement observable, il est très difficile de tout prévoir à l'avance. Pour ces raisons, il n'est pas d'une très grande utilité de tout planifier à l'avance, puisque les plans seront constamment à refaire. Un autre problème avec ce type d'architecture est que la génération de plans précis demande beaucoup de ressources (temps de calcul et mémoire).

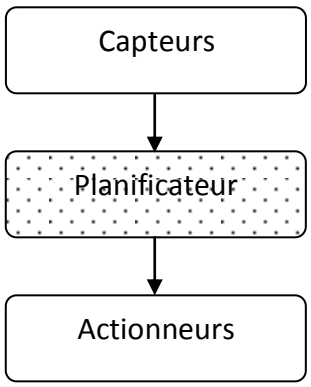


Figure 1.8 : Architecture délibérative.

1.2.2.2 Architecture comportementale (réactive) :

L'architecture comportementale [Brooks 1] [Brooks 2] [Rodney] est inspirée par le comportement des insectes. L'idée générale est de développer plusieurs petits modules simples et indépendants les uns des autres et, une fois regroupés, un comportement plus intelligent émerge sans qu'il ait été spécifiquement programmé. Ce

type d'architecture est complètement à l'opposé des architectures délibératives et ne fait aucune place à des processus raisonnés.

Un exemple d'architecture comportementale est illustré à la figure 1.9. Celle-ci est composée de plusieurs modules simples appelés comportements et d'un module d'arbitration.

Dans sa conception, chaque comportement se limite à une seule fonctionnalité pour le contrôle du robot. Les comportements sont tous indépendants les uns des autres. Par exemple, on peut avoir des comportements pour l'évitement d'obstacles, le suivi de chemin, le suivi d'objets de couleur ou la manipulation d'objets. Les comportements sont exécutés parallèlement à une certaine fréquence.

Lors d'une itération, chaque comportement calcule une ou plusieurs commandes motrices qui sont envoyées à un module d'arbitration. Ce dernier fusionne l'ensemble des commandes reçues et calcule les commandes finales devant être envoyées à chacun des actionneurs du robot.

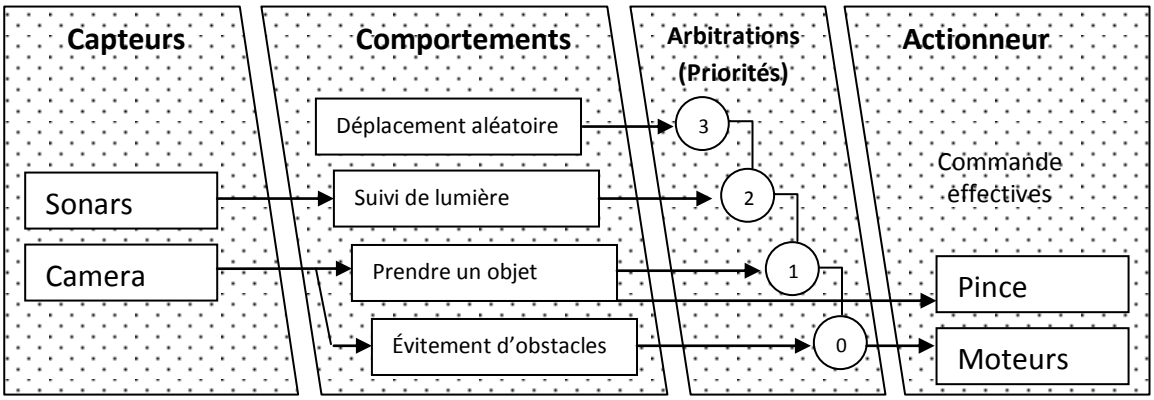


Figure 1.9 : Exemple d'architecture comportementale.

En guise d'exemple, examinons l'intégration d'un comportement d'évitement d'obstacles et d'un comportement de suivi de chemin. En présence d'un obstacle sur un côté (voir figure 1.10), le comportement d'évitement d'obstacles ne fait que tourner vers le côté opposé. Indépendamment, le comportement de suivi de chemin aligne et dirige en ligne droite le robot vers la cible courante. Comme nous pouvons l'apercevoir à la figure 1.11, ce comportement ignore la présence d'obstacles.

Afin de diriger le robot vers la cible sans collision, les capacités de ces deux comportements doivent être fusionnées par un mécanisme d'arbitration.

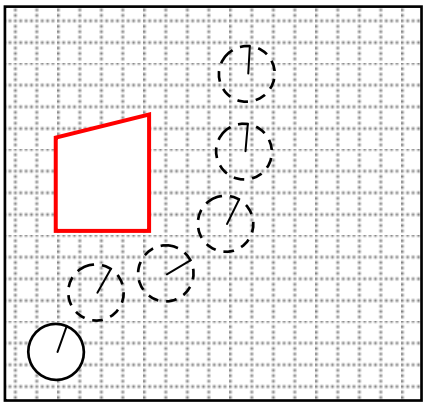


Figure 1.10 : Évitement d'obstacle

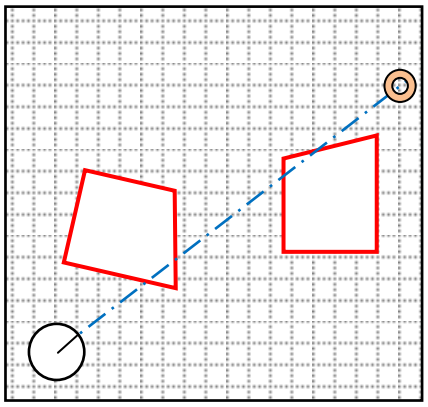


Figure 1.11 : Suivi de chemin

L'approche classique consiste à utiliser une arbitration par priorité. Dans le cas présent, l'évitement d'obstacles a priorité sur le suivi de chemin. Lorsqu'aucun obstacle n'est détecté, le module d'évitement d'obstacles ne génère aucune commande et laisse le plein contrôle au module de suivi de chemin. Par contre, quand un obstacle est détecté, le module d'évitement substitue la commande du module de suivi de chemin par une commande de rotation dans le sens opposé à l'obstacle perçu. Lorsque l'obstacle est complètement évité et sorti du champ de vision des capteurs, le comportement de suivi de chemin reprend le plein contrôle du robot.

En contrepartie, les architectures comportementales ont de la difficulté à réaliser des tâches structurées, puisqu'elles ne contiennent aucun processus délibératif. En effet, les tâches complexes requièrent la capacité du robot à prédire les conséquences futures de ses actions afin de sélectionner celles qui conviennent le mieux pour la réalisation de ses activités. En d'autres mots, ces tâches complexes ont besoin d'être planifiées.

1.2.2.3 Architecture hybride :

Les limitations des deux types d'architecture précédentes justifient l'émergence récente des architectures hybrides [Arkin 1] [Arkin 2] [Arkin 3], tentant de combiner les avantages des architectures délibératives et comportementales. Elles sont généralement décomposées en plusieurs niveaux.

Dans la partie supérieure, on place les modules de type délibératif. Dans la partie inférieure, on retrouve les modules de type comportemental. La figure 1.12 illustre un exemple d'architecture hybride où un planificateur s'occupe des capacités délibératives.

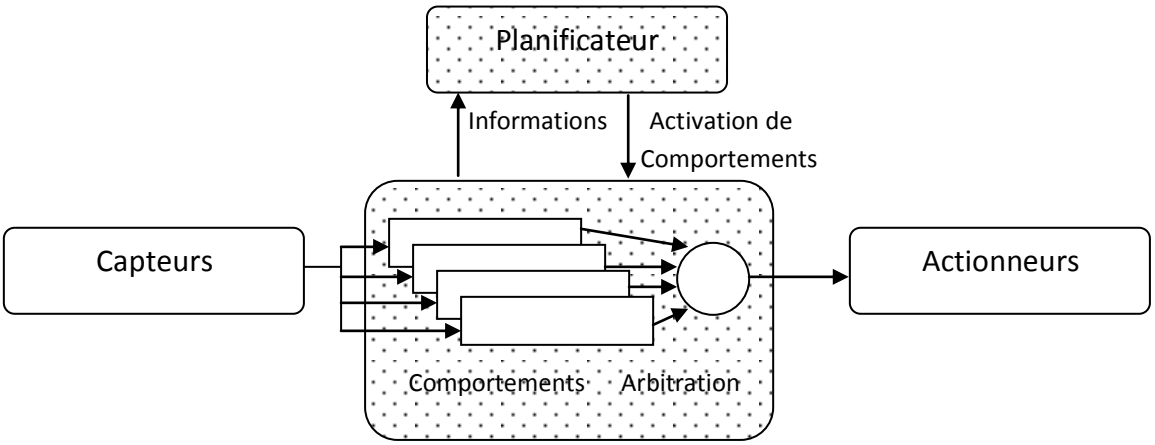


Figure 1.12 : Architecture hybride.

1.3 Conclusion :

Dans ce chapitre nous avons résumé toutes les notions de base nécessaires à la compréhension du domaine de la robotique mobile, ainsi que les techniques de localisation et de navigation d'un robot autonome, nous avons aussi présenté un aperçu sur les architectures de contrôle. Nous allons, dans le prochain chapitre, présenter le contrôleur flou réactif adopté et le système neuro-flou ANFIS pour l'optimisation de ce contrôleur flou.

Chapitre 2

Sommaire partiel 2:

2

Navigateur flou et optimisation neuro-flou pour la navigation du robot ARP

Dans ce chapitre, nous présentons le navigateur flou adopté à notre système de navigation réactive du robot ARP. Ce navigateur envoie des consignes de vitesse et angle de braquage pour assurer la convergence du robot ARP vers un point but dans son espace de navigation. Le réglage des différents paramètres de ce navigateur peut nous fournir de bons résultats pour la navigation réactive du robot ARP. De manière à augmenter l'adaptabilité de notre système de navigation, nous nous sommes tournés vers une technique hybride, association du flou et du neuronal. Il s'agit donc, d'optimiser et identifier certains paramètres du navigateur flou. Mais d'abord, nous allons, à travers quelques notions de base et de rapides rappels théoriques, définir la logique floue et les réseaux de neurones pour pouvoir ensuite présenter notre navigateur flou et la méthode neuro-flou adoptée.

2.1 Introduction :

Alors que les ordinateurs ne traitent que des nombres, les individus utilisent essentiellement des concepts liés entre eux par des règles logiques. Ces concepts, qui possèdent un fort contenu sémantique, sont matérialisés par des mots, plus ou moins vagues. La logique floue [BUHLER] [ZINSER] [Ahmad] se propose de formaliser l'usage des termes vagues, dans le but de les rendre manipulables par les ordinateurs.

La logique floue est née de la constatation que la plupart des phénomènes ne peuvent pas être représentés à l'aide de variables booléennes qui ne peuvent prendre que deux valeurs (0 ou 1). Déterminer si une personne est de petite ou grande taille est facile pour n'importe lequel d'entre - nous, et cela sans nécessairement connaître sa taille exacte. Un ordinateur, lui, devrait non seulement connaître la taille exacte mais également posséder un algorithme divisant immanquablement une population en deux ensembles rigides: les grands et les petits. Supposons que la limite soit 1m70, une personne qui mesure 1m69, est elle vraiment petite ? N'est-elle pas ni vraiment grande, ni vraiment petite mais tout simplement d'une taille moyenne ?

Pour répondre à ce type de question, la logique floue considère la notion d'appartenance d'un objet à un ensemble non plus comme une fonction booléenne mais comme une fonction qui peut prendre toutes les valeurs entre 0 et 1.

2.2 Logique booléenne, logique floue et fonction d'appartenance :

Pour illustrer très concrètement le principe fondamental de la logique floue [Cirstea] [Ahmad], nous allons prendre l'exemple précédent. Nous souhaitons évaluer les tailles des personnes. Grande taille ou petite taille ?

Dans le cadre de la logique booléenne, il est nécessaire d'introduire un seuil correspondant à la valeur limite de la taille qui va déterminer si la taille est grande ou petite (figure 2.1).

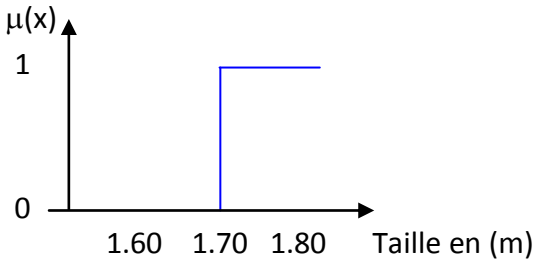


Figure 2.1 : Logique classique.

On s'aperçoit immédiatement que la logique booléenne est difficilement applicable à notre problème. Son utilisation en devient même complètement absurde : un simple degré d'écart entre deux valeurs de taille peut faire basculer d'un état à l'autre.

La logique floue, qui introduit une infinité de valeurs entre vrai et faux, permet de déterminer un degré d'appartenance à l'un ou l'autre état (figure 2.2). La logique floue comble donc les lacunes de la logique booléenne en introduisant la notion de continuité entre les états.

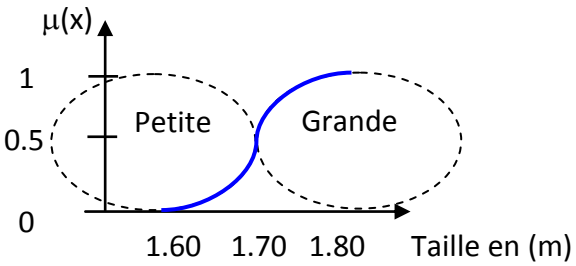


Figure 2.2 : Logique floue.

Par exemple, si nous constatons que la taille de 1.71 m, nous pouvons dire qu'elle est plus grande que petite sans pour autant affirmer qu'elle est vraiment grande!

2.2.1 La théorie des sous-ensembles :

Soit une variable x (la taille) et un univers de référence ou de discourt U (l'intervalle de tailles entre 1.60 et 1.80 m), Un sous-ensemble flou A est défini par une fonction d'appartenance μ(x) qui décrit le degré avec lequel l'élément x appartient à A.

Théorie Classique :

$$\mu(x) = \begin{cases} 1 \text{ si } x = A \\ 0 \text{ sinon} \end{cases}$$

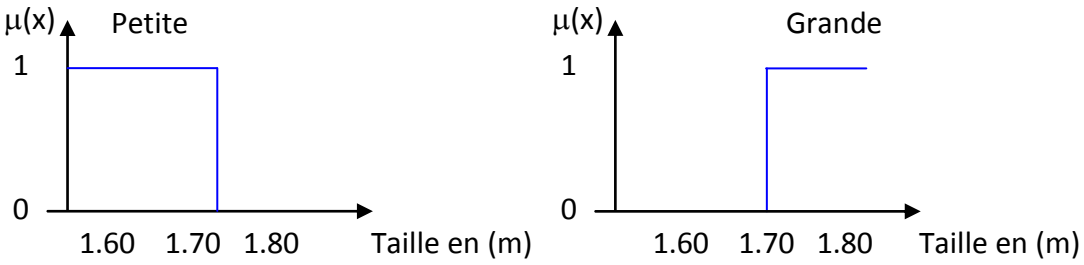


Figure 2.3 : Théorie classique.

Ici la taille de 1.65 m appartient nécessairement au sous-ensemble (petite), avec un degré 1 c.-à-d. 100%.

Théorie Floue :

μ(x) = [0 1]

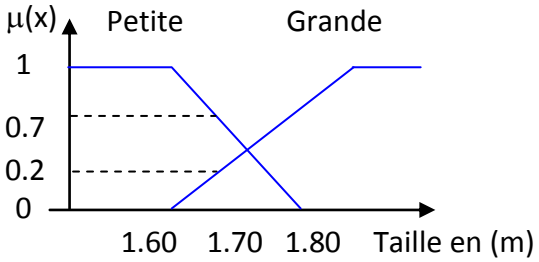


Figure 2.4 : Théorie floue.

Ici la taille de 1.65 m appartient en même temps au sous-ensemble flou (Petite) et au sous-ensemble flou (Grande), respectivement avec un degré de 0,7 et de 0,2.

Dans le jargon de cette nouvelle théorie, on parle de variable linguistique (taille) et de valeurs linguistiques (Petite et Grande) [BUHLER] [ZINSER].

2.2.2 Types de fonctions d'appartenance :

Il existe plusieurs types de fonctions d'appartenance [Cirstea] et le choix de forme dépend de l'application individuelle. Voir figure 2.5 :

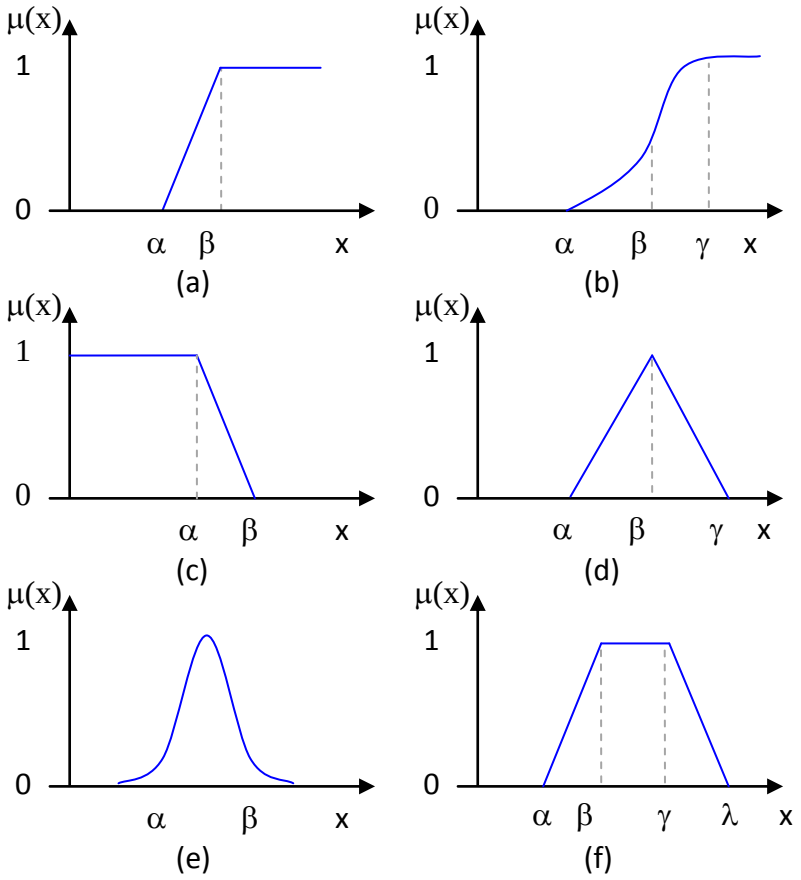


Figure 2.5 : Types de fonctions d'appartenances: (a) fonction – Γ (demi trappez droit); (b) fonction – S (sigmoïde); (c) fonction – L (demi trappez gauche); (d) fonction – Λ (triangle); (e) fonction Gaussienne; (f) fonction -Π (trappez)

Définitions mathématique des fonctions d'appartenance :

Fonction - Γ, Γ : U [0,1]

$$\Gamma(x,\alpha,\beta)=\begin{cases}0 & x<\alpha\\ \dfrac{(x-\alpha)}{(\alpha-\beta)} & \alpha\le x\le\beta\\ 1 & x>\beta\end{cases}$$

Fonction - L, L : U [0,1]

$$L(x,\alpha,\beta)=\begin{cases}1 & x<\alpha\\ \dfrac{(x-\beta)}{(\alpha-\beta)} & \alpha\le x\le\beta\\ 0 & x>\beta\end{cases}$$

Fonction - Λ, Λ : U [0,1]

$$\Lambda(x,\alpha,\beta)=\begin{cases}0 & x<\alpha\\ \dfrac{(x-\alpha)}{(\beta-\alpha)} & \alpha\le x\le\beta\\ \dfrac{(x-\gamma)}{(\beta-\gamma)} & \beta\le x\le\gamma\\ 0 & x>\gamma\end{cases}$$

Fonction Gaussienne avec (c : le centre du gaussien, σ : l'écart-type)

$$f(x)=exp\left(\frac{-0.5(x-c)^2}{\sigma^2}\right)$$

2.2.3 Les opérateurs en logique floue :

Il existe une logique binaire pour les ensembles classiques et des logiques multivalentes (avec degré de vérité) pour les sous-ensembles flous. Les opérateurs sur les sous-ensembles flous sont créés en essayant de respecter un certain nombre de propriétés et de retrouver les opérations de base existant en logique booléenne telles que l'union ou l'intersection [Ahmad].

2.2.3.1 L'opérateur NON (complément) : Il est défini mathématiquement par :

- NON (μ_A (x)) = 1- μ_A (x).

Exemple arithmétique d'opérateur NON :

- On a « A = 0.7 + 1 + 0.6 + 0.2 + 0 » ;
- Alors, « A' = 0.3+ 0 + 0.4+ 0.8 +1 ».

2.2.3.2 L'opérateur ET (intersection) : Il est défini mathématiquement par :

- $\mu_A(x) \cap \mu_B(x) = \mathbf{min}\ (\mu_A(x),\ \mu_B(x))$.

Exemple arithmétique d'opérateur ET :

- On a « A = 0.3+ 0 + 0.4 + 0.8 +1 » ;
- ET « B = 0.2+ 0.3 + 0.1 + 0.2 + 0.4 » ;
- Alors, « A∩ B = 0.2+ 0 + 0.1 + 0.2+ 0.4 ».

2.2.3.3 L'opérateur OU (union) : Il est défini mathématiquement par :

- $\mu_A(x) \cup \mu_B(x) = \mathbf{Max}\ (\mu_A(x),\ \mu_B(x))$.

Exemple arithmétique d'opérateur OU :

- On a « A = 0.3+ 0 + 0.4+ 0.8 + 1 » ;
- OU « B = 0.2 + 0.3 + 0.1 + 0.2 + 0.4» ;
- Alors, « A ∪ B = 0.3 + 0.3 + 0.4 + 0.8 +1 ».

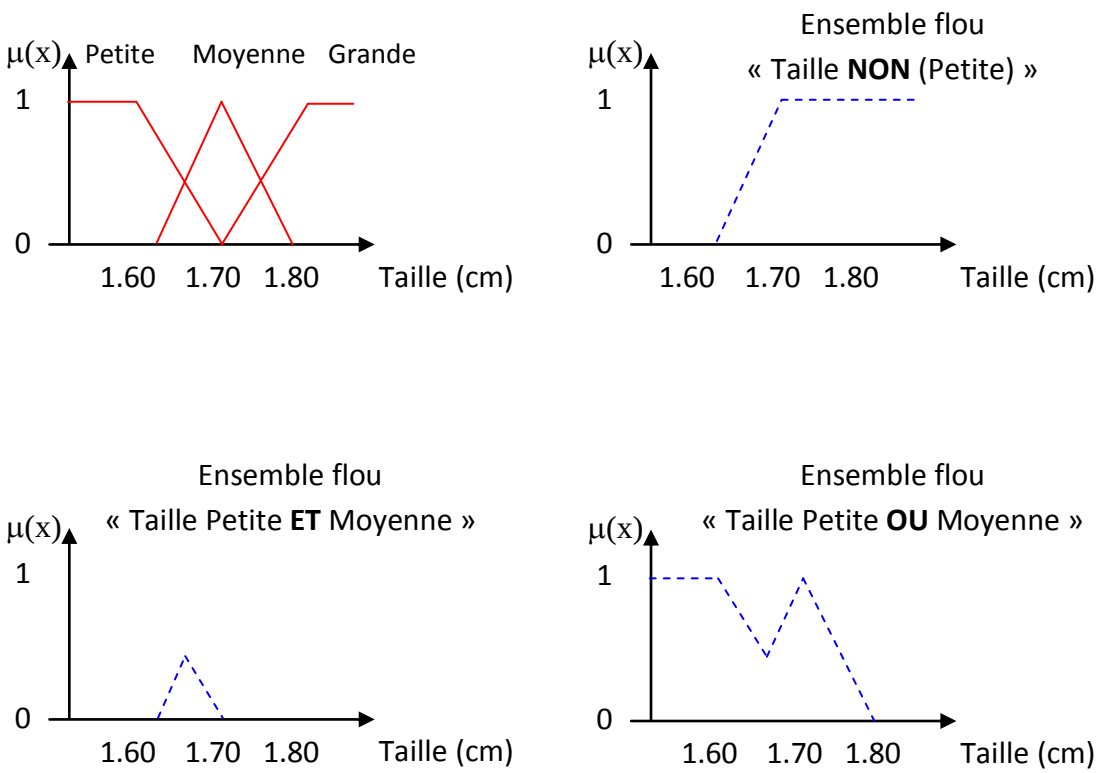


Figure 2.6 : Schéma d'opérations sur ensembles flou.

2.2.4 Principe de fonctionnement d'un système d'inference flou :

Le principe de fonctionnement d'un système d'inference flou [BUHLER] [ZINSER] est simple. Celui-ci se décompose en trois étapes distinctes :

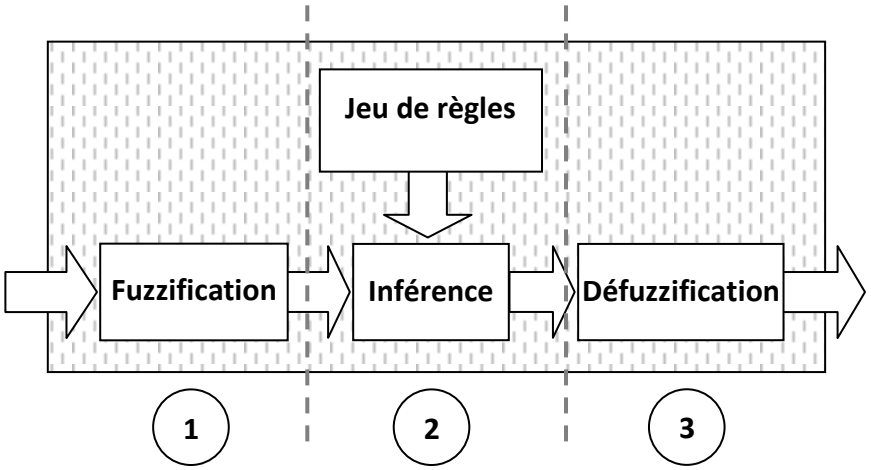


Figure 2.7 : Configuration interne d'un **système d'inference** flou.

a. Fuzzification ou quantification floue :

Cette première étape consiste à déterminer le degré d'appartenance de chaque variable d'entrée à chaque état. Celui-ci est déterminé à l'aide des fonctions d'appartenances présentées précédemment.

b. Établissement des règles liantes les sorties aux entrées :

Après avoir (fuzzifié) les variables d'entrée et de sortie, il faut établir les règles liant les entrées aux sorties afin d'analyser l'état ou la valeur des entrées du système pour déterminer l'état ou la valeur de toutes les sorties.

Exemple : Si la distance (entre robot et obstacle) est très petite, alors vitesse du robot est faible.

b.1 Inférence : Les degrés d'appartenance de chaque variable à chaque état permettent d'appliquer les règles floues qui ont été préalablement définies. Le degré d'appartenance des variables de sortie à chaque état est ainsi obtenu. En d'autres termes l'inférence lie les grandeurs mesurées transformées en variables linguistiques à l'aide de la Fuzzification, à la variable de sortie exprimée comme variable linguistique. On peut distinguer deux genres de règles d'inférences:

- Inférence avec une seule règle
- Inférence avec plusieurs règles

Basée sur le choix d'opérateur *(ET), (OU)* et l'implication floue *(ALORS)*, il existe plusieurs méthodes d'inférence tel que (Mamdani [Mamdani], Sugeno [M. Sugeno 2] [Tak Sug], Larsen, Somme-Prod, Tsukamoto), nous présenterons dans ce chapitre deux méthodes (Mamdani et Sugeno).

c. Défuzzification :

En sortie, le système flou ne peut pas communiquer des valeurs floues qu'il peut seul exploiter. Il lui est donc nécessaire de fournir des valeurs précises, c'est le rôle de la défuzzification. Cette étape s'effectue toujours à l'aide des fonctions d'appartenance. A partir des degrés d'appartenance, on obtient autant de valeurs qu'il y a d'états. Pour déterminer la valeur précise à utiliser, il existe plusieurs méthodes tel que : (le maximum, hauteurs pondérées, le centre de gravité des valeurs obtenues).

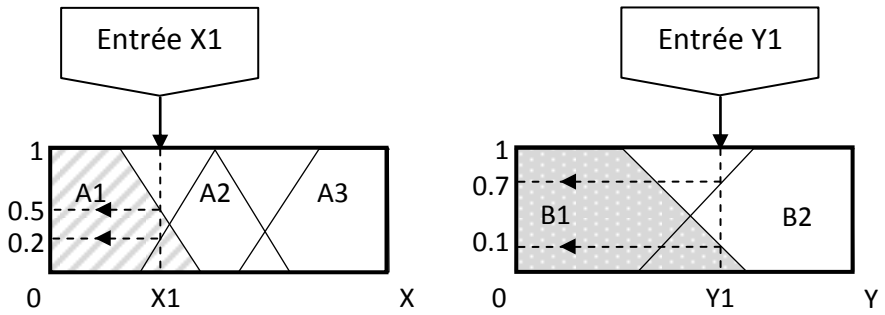
2.2.4.1 Systèmes d'inference flou de Mamdani :

En 1975, première application industrielle. Régulation floue d'une chaudière à vapeur réalisée par le professeur Ebrahim Mamdani [Mamdani] (Mamdani et Assilian, 1975) de l'université de Londres.

On prend un exemple d'un système d'inference flou de Mamdani à deux entrées *(X, Y)* et une sortie *(Z)* :

2.2.4.1.1 Fuzzification (Mamdani):

L'entrée *(X)* est répartie en trois sous ensembles flous *(A1, A2, A3)*, et l'entrée *(Y)* est répartie en deux sous ensembles flous *(B1, B2)*, la sortie *(Z)* est répartie en trois sous ensemble flous *(C1, C2, C3)*. Pour *(X=X1)* et *(Y=Y1)* on détermine le degré d'appartenance de *(X1)* et *(Y1)* :



μ (x1=A1)=0.5
μ (x1=A2)=0.2

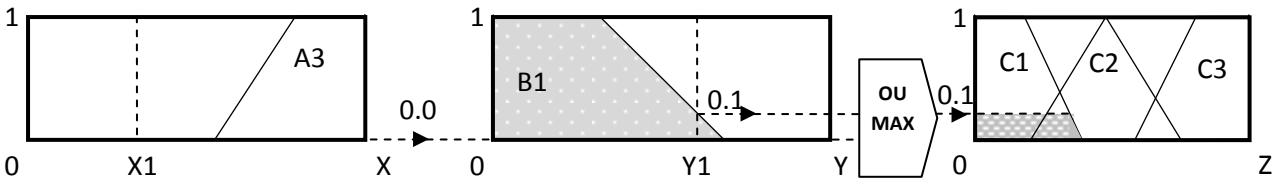
μ (y1=B1)=0.1
μ (y1=B2)=0.7

Figure 2.8 : Étape 1 (fuzzification) du système d'inference flou de Mamdani.

2.2.4.1.2 Établissement des règles floues (Mamdani):

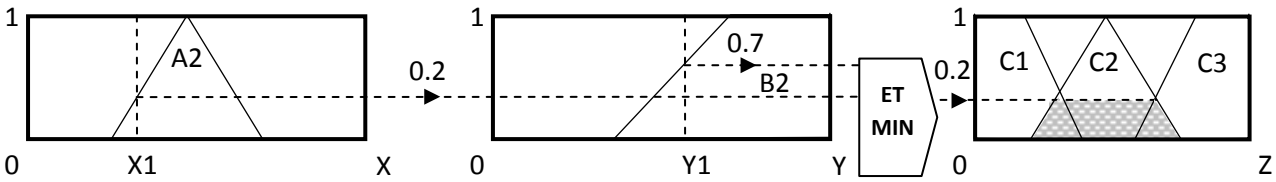
Règle 1 : Si *X* est *A3 (0.0)* **OU** *Y* est *B1 (0.1)* **Alors** *Z* est *C1 (0.1).*

Selon la règle 1, *(X1)* n'appartient pas au sous ensemble flou *(A3)* son degré d'appartenance est nul *(0.0)*, et *(Y1)* appartient au sous ensemble flou *(B1)* avec un degré d'appartenance de *(0.1)*. Le **OU** logique dans la méthode de Mamdani [Mamdani] se traduit par la valeur maximum entre ces deux degrés, le résultat alors est *C1 (0.1).*



Règle 2 : Si *X* est *A2 (0.2)* **ET** *Y* est *B2 (0.7)* **Alors** *Z* est *C2 (0.2).*

Même chose pour la règle 2. Mais cette fois si le **ET** logique dans la méthode de Mamdani [Mamdani] se traduit par la valeur minimum entre ces deux degrés, le résultat alors est *C2 (0.2).*



Règle 3 : Si *X* est *A1 (0.5)* **Alors** *Z* est *C3 (0.5).*

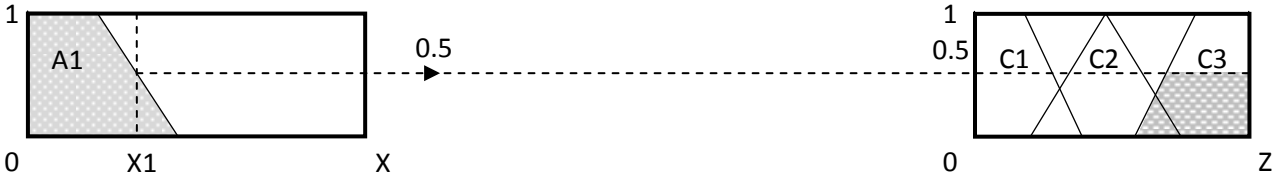
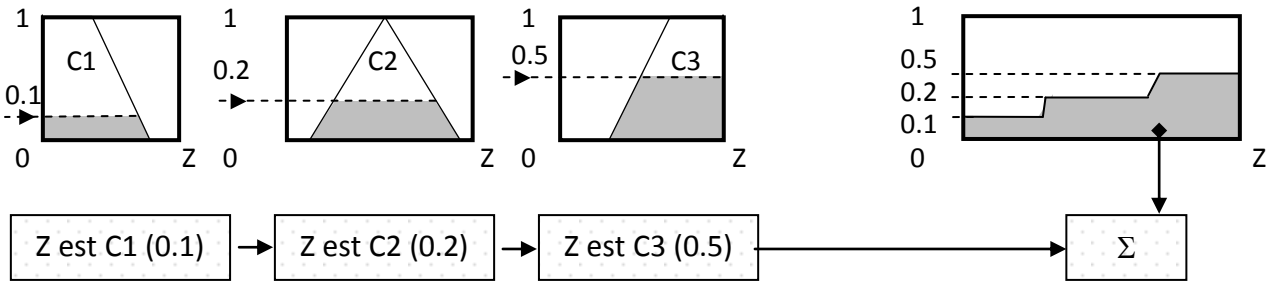


Figure 2.9 : Étape 2 (Règles floues) du système d'inference flou de Mamdani.

2.2.4.1.3 Agrégation des règles et défuzzification (Mamdani):

L'agrégation est le processus d'unification des sorties de toutes les règles floues. En d'autres termes, on assemble les fonctions d'appartenances des résultats des trois règles précédentes *(C1, C2, C3)* multiplié par leurs degrés d'appartenances c'est-à-dire *[C1 (0.1), C2 (0.2), C3 (0.5)]*. On obtient alors un seul sous ensemble flou à la sortie. La defuzzification de Mamdani [Mamdani] se fait par le calcul du centre de gravité de la surface obtenue après l'agrégation et le projeté sur l'axe horizontal (figure 2.10).



Le résultat final est : $Z1 = \dfrac{\sum z.\mu(z)}{\sum \mu(z)}$

Figure 2.10 : Étape 3 (Agrégation et défuzzification) du système d'inference flou de Mamdani.

2.2.4.2 Systèmes d'inference flou de Sugeno :

Pour le même système précédent à deux variables d'entrées et une sortie, les règles floues sont de la forme :

- **SI** *X* est *Ai* **ET** *Y* est *Bi* **Alors** *Z* est *Ci* *(1)*

Où *(Ai)* et *(Bi)* sont des sous ensembles flous, par contre *(Ci)* peut appartenir aussi bien au domaine symbolique (surface / sous ensemble flou) qu'au domaine numérique. L'originalité de la méthode de Sugeno réside dans le fait que la conclusion de chaque règle n'appartient pas au domaine symbolique (surface), mais est définie sous forme numérique comme une combinaison linéaire des entrées. Selon la méthode de Sugeno et son collaborateur Takagi [Tak Sug] [M. Sugeno 2], les règles floues, dans le cas de deux variables, s'expriment donc selon la forme suivante :

- **SI** *X* est *Ai* **ET** *Y* est *Bi* **Alors** $Z = a_i.X + b_i.Y + c_i$ *(2)*

  On parle dans ce cas de système d'inference flou de type **Sugeno d'ordre 1**, où les paramètres a_i, b_i et c_i sont des paramètres à identifier [Tak Sug] [M. Sugeno 1] (ce type de système est utilisé dans le cas d'un apprentissage neuro-flou).

- Lorsque la sortie $Z = c_i$ on parle alors de système d'inference flou de type **Sugeno d'ordre 0** [Tak Sug] [M. Sugeno 2] (pas de paramètres à identifier, pas d'apprentissage).

Dans le système d'inference flou type Sugeno, les étapes d'agrégation et de défuzzification des règles floues se font simultanément:

$$Z_T = \frac{\sum(\mu_i Z_i)}{\sum \mu_i} \qquad (3)$$

Une remarque peut être formulée sur le nom donné à cette étape. En effet, elle est appelée (défuzzification) alors qu'elle ne manipule aucune donnée floue. Ce choix a été dicté afin d'établir une similitude entre ce type de système et le système d'inference de type Mamdani où le cheminement (fuzzification / inférence floue / défuzzification) a été introduit. A la place de (défuzzification), le terme (agrégation) aurait été préférable.

Cette méthode est plus simple à mettre en œuvre et donne aussi de bons résultats en commande floue que la méthode de Mamdani. Elle est utilisée aussi dans le système hybride neuro-flou adaptatif de Jang (Adaptive network based fuzzy inference system) [R. Jang 1] que nous allons présenter par la suite de cette section.

2.3 Les réseaux de neurones :

Les réseaux de neurones artificiels ne sont autres qu'une inspiration des réseaux de neurones biologiques, Les premiers travaux sur les neurones artificiels ont débuté au début des années 1940 et ont été menés par McCulloch et Pitts. Ils décrivent les propriétés du système nerveux à partir de neurones idéalisés (ce sont des neurones logiques [0 ou 1]).

Nous commençons par décrire l'unité de base d'un réseau de neurones artificiel, le neurone formel :

2.3.1 Le neurone formel :

Le neurone formel [T. Bräunl] [P. PREUX] est constitué d'un noyau, d'une liaison synaptique de sortie et de liaisons synaptiques d'entrées qui font le lien avec les neurones environnants. A chaque liaison d'entrée est attaché un poids appelé poids synaptique. Voir figure 2.11. Le noyau réalise deux fonctions:

- La première est celle de la sommation algébrique des entrées (i_i) pondérées par les poids synaptiques (w_i) pour obtenir l'état interne *a (i, w)* du neurone ;

- La deuxième est la fonction d'activation qui détermine la valeur de sortie (o) du neurone en fonction de son état interne. Les fonctions les plus connues sont : (la fonction tangente, sigmoïde, signe,…).

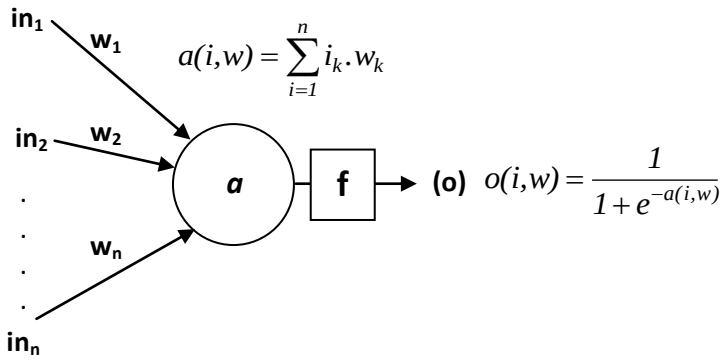


Figure 2.11 : Neurone formel.

2.3.2 Le réseau de neurones PMC :

Le type de réseau le plus courant, communément appelé (Perceptron Multicouche) [T. Bräunl] [P. PREUX] est organisé en couches successives : couche d'entrée, une ou plusieurs couches cachées et couche de sortie. Chaque neurone d'une couche est relié à tous les neurones de la couche suivante voir figure 2.12.

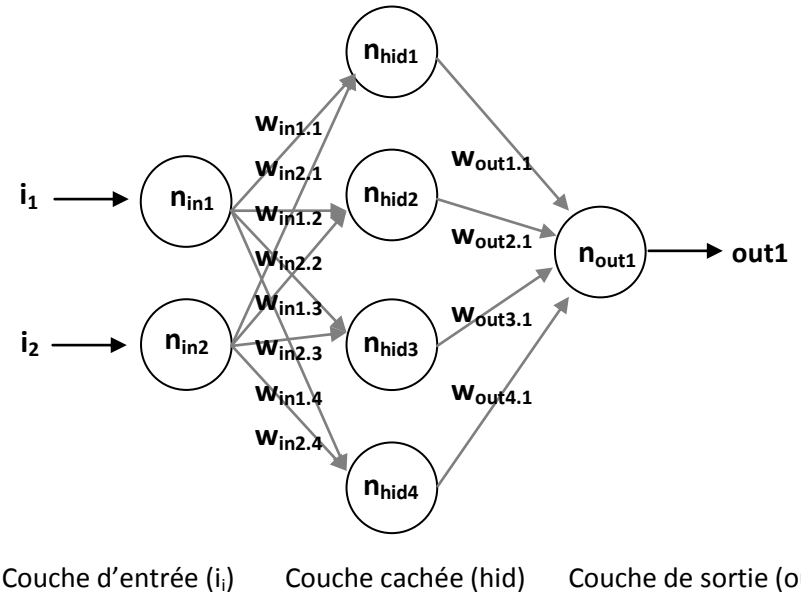


Figure 2.12: PMC à trois couches.

2.3.3 Le réseau de neurones a fonction de base radiale RBF :

Les réseaux de neurones à fonctions de base radiale (Radial Basis Function) sont des réseaux de neurones à une seule couche cachée dont les fonctions d'activation sont des fonctions à base radiale, le plus souvent des gaussiennes. La fonction d'activation du neurone de la couche de sortie est l'identité. Les entrées sont directement connectées aux neurones de la couche cachée voir figure 2.13.

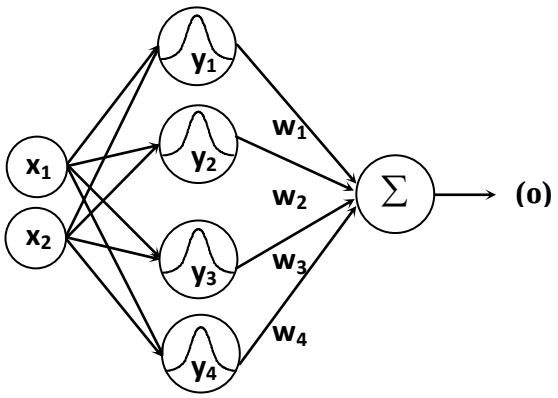


Figure 2.13 : Réseau de neurones de type RBF.

La sortie d'un neurone de la couche cachée peut donc s'interpréter comme le produit des degrés d'appartenance des entrées à une famille de gaussiennes définies par leur centre *cij* et leur écart-type σj. Ces derniers, avec les poids *wi*, sont les paramètres à optimiser en vue de l'apprentissage. La sortie *(o)* est calculée comme une combinaison linéaire des *wi* pondérés par les valeurs des sorties des neurones de la couche intermédiaire.

Remarque : il est intéressant de remarquer qu'il existe une équivalence fonctionnelle entre un réseau de neurones RBF et un système d'inférence flou de Jang [R. Jang 1] [R. Jang 4] bien que provenant d'origines complètement différentes. Il faut dans ce cas choisir des fonctions d'appartenance gaussiennes ainsi qu'un nombre de neurones égal au nombre de règles d'inférence.

2.4 Synthèse Neuro-Flou (identification et optimisation du Navigateur flou du robot ARP) :

Les systèmes hybrides neuro-flou [M. Negnev] [Petru] [Shen] [Lin] [Kadri] combinent les traitements parallèles et les capacités d'apprentissage des réseaux de neurones [T. Bräunl] avec les raisonnements anthropomorphiques et les capacités d'explication des systèmes flous. Le but est d'apprendre à un système d'inference flou à suivre un model de référence (base de données). Le résultat de ses commandes est jugé grâce à une table de performance permettant d'indiquer les corrections nécessaires. Ces corrections sont alors répercutées sur les différentes règles en tenant compte de leur rôle pour le calcul de la commande finale. La topologie du réseau de neurone est fonctionnellement équivalente à celle d'un modèle d'inférence flou, et on peut l'entraîner à :

- Développer des règles floues **Si-Alors** ;
- Trouver les fonctions d'appartenance de variables d'entrées / sorties en partant d'un ensemble de données représentatives.

Nous allons dans un premier temps, présenter le navigateur flou du robot ARP ensuite la technique d'optimisation adoptée (ANFIS).

2.4.1 Synthèse du Navigateur flou réactif du robot ARP :

Le navigateur flou du robot ARP est de type Sugeno [M. Sugeno 2] [Tak Sug], il a pour tâche de contrôler le déplacement linéaire et angulaire, il possède trois entrées et deux sorties, les entrées sont trois capteurs infrarouges qui permettent de relever les distances des obstacles situés dans la zone droite, gauche et frontale. Les sorties sont la vitesse et l'angle de braquage du robot ARP (figure 2.14).

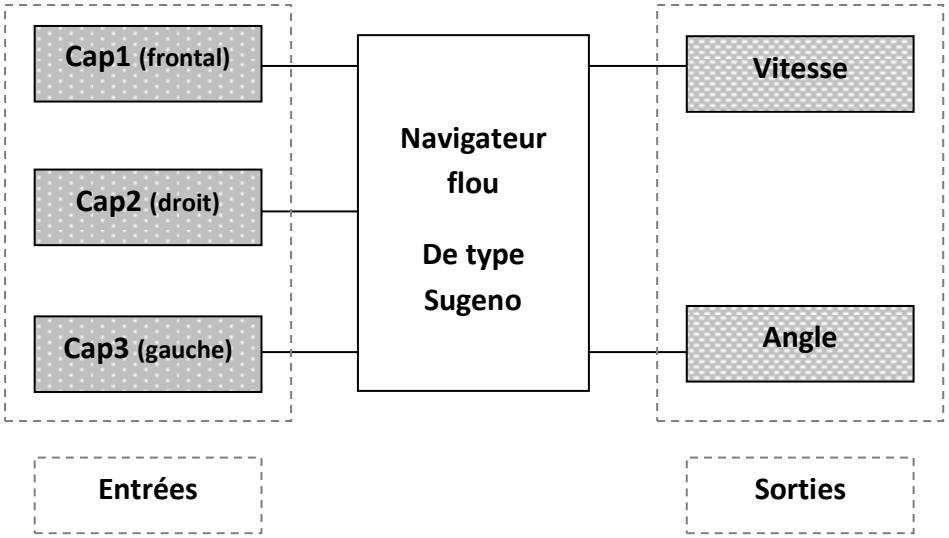


Figure 2.14 : Navigateur flou du robot ARP.

2.4.1.1 Fuzzification (Navigateur flou du robot ARP):

Les entrées : les fonctions d'appartenance des entrées de notre navigateur flou sont des gaussiennes et chaque entrée capteurs est représentée en deux sous-ensembles flous (obstacle **Prés** – obstacle **Loin**). Les fonctions d'appartenance sont symétriques et distribuées de manière équidistante sur un intervalle de [6 à 30] cm pour toutes les entrées. Ce dernier représente l'intervalle de distance captée par des capteurs infrarouges de type GP2D120 que nous verrons dans le chapitre de réalisation du robot ARP.

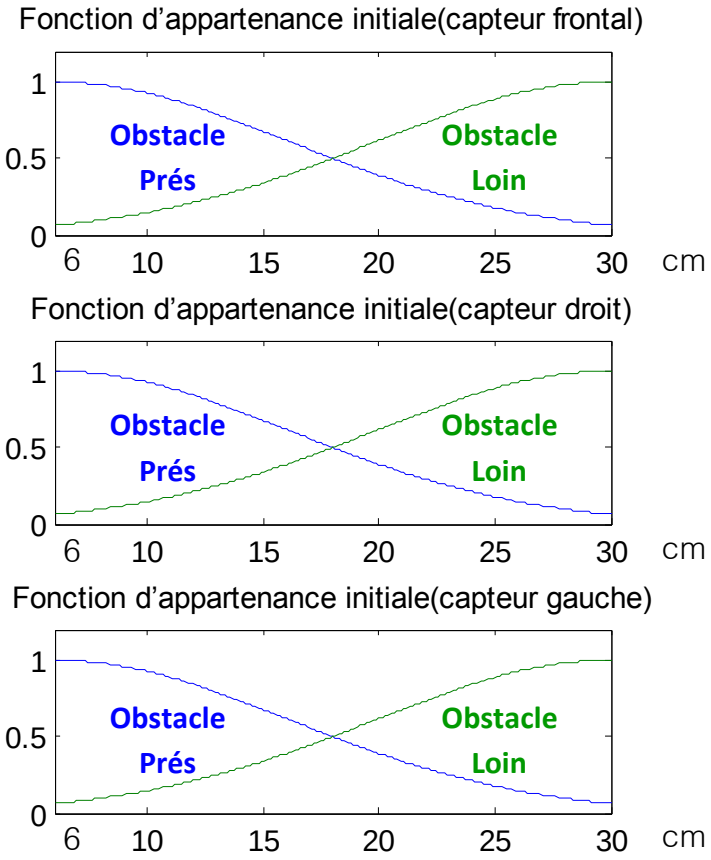


Figure 2.15 : Fonction d'appartenance des trois entrées capteurs.

Le chevauchement entre les sous ensemble flous est nécessaire, sinon le système aura tendance à fonctionner comme un système gouverné par la logique classique, diminuant aussi sa souplesse et sa possibilité de traiter des situations intermédiaires. Des pourcentages de 25% à 50% sont utilisés dans la majorité des cas. Toutefois, le chevauchement ne doit pas dépasser ces limites car trop de chevauchement diminue la possibilité du système à faire la distinction entre deux sous ensemble flous.

Les sorties : Comme nous l'avons déjà vu précédemment, la sortie du navigateur flou de Sugeno (agrégation et défuzzification des règles floues) [M. Sugeno 2] [Tak Sug] se fait numériquement (pas de surfaces au niveau des fonctions d'appartenances), donc les sorties de notre navigateur seront représentées dans des intervalles numériques.

L'angle : La fonction d'appartenance est donnée sous la forme de constante appartenant à l'intervalle [-90 90] °.

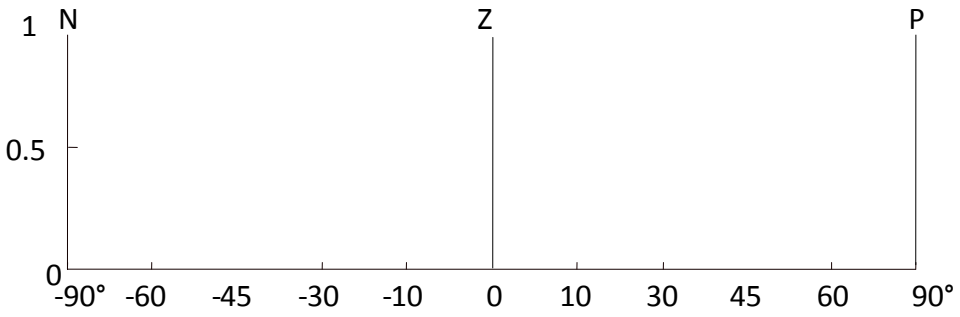


Figure 2.16 : Fonction d'appartenance de la sortie Angle.

Les différents ensembles flous sont caractérisés par des désignations standards, la signification des symboles est indiquée au tableau 1 suivant :

Tableau 1 :

Symboles Angle	Significations
N	Négatif
Z	Environ Zéro
P	Positif

Négatif pour tourner à gauche, **positive** pour tourner à droite et angle **zéro** pour avancer tout droit.

La vitesse : La fonction d'appartenance est donnée sous la forme de constante appartenant à l'intervalle [150 255] avec 255 correspond en réalité a 120 tr/mn.

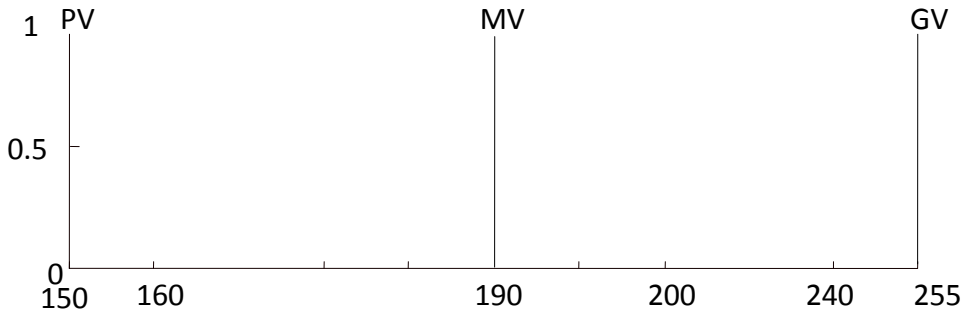


Figure 2.17 : Fonction d'appartenance de la sortie Vitesse.

Tableau 2 :

Symboles Vitesse	Significations
PV	Petite Vitesse
MV	Moyenne Vitesse
GV	Grande Vitesse

2.4.1.2 Base des règles :

La base des règles consiste à déterminer le nombre de règles d'inférence, nous avons trois fonctions d'appartenance (trois capteurs de détection d'obstacle) et chaque fonction est divisée en deux sous ensemble flous (obstacle **Prés** – obstacle **Loin**), on déduit huit règles (il faut s'assurer que chaque sous ensemble de chacune des variables est utilisé au moins une fois) : avec φ =Angle, V=Vitesse, cap1=capteur frontal, cap2= capteur droit, cap3=capteur gauche.

1. Si (cap1 = près) et (cap2 = près) et (cap3 = près) alors (V = PV) et (φ = N) ;
2. Si (cap1 = près) et (cap2 = près) et (cap3 = loin) alors (V = PV) et (φ = P) ;
3. Si (cap1 = près) et (cap2 = loin) et (cap3 = près) alors (V = PV) et (φ = N) ;
4. Si (cap1 = près) et (cap2 = loin) et (cap3 = loin) alors (V = PV) et (φ = N) ;
5. Si (cap1 = loin) et (cap2 = près) et (cap3 = près) alors (V = MV) et (φ = Z) ;
6. Si (cap1 = loin) et (cap2 = loin) et (cap3 = près) alors (V = MV) et (φ = N) ;
7. Si (cap1 = loin) et (cap2 = près) et (cap3 = loin) alors (V = MV) et (φ = P) ;
8. Si (cap1 = loin) et (cap2 = loin) et (cap3 = loin) alors (V = GV) et (φ = Z).

Les règles sont regroupées dans le tableau d'inférence suivant :

Tableau 3 :

Cap1	Cap2	Cap3	Vitesse	Angle
près	près	près	PV	N
près	près	loin	PV	P
près	loin	près	PV	N
près	loin	loin	PV	N
loin	près	près	MV	Z
loin	loin	près	MV	N
loin	près	loin	MV	P
loin	loin	loin	GV	Z

2.4.1.3 Politique de navigation réactive (le module de convergence vers un but) :

Comme nous l'avons déjà vu au premier chapitre (navigation et architecture de contrôle), la politique consistant à se diriger directement vers le point d'arrivée en évitant les obstacles, s'exprime par le fait que, les **sept premières règles** floues sont utilisées pour **le comportement réactif d'évitement d'obstacle** alors que **la huitième règle** permet d'atteindre l'objectif (**le comportement réactif de convergence vers un but**). (θ) désigne l'angle qui sépare l'axe du robot et le point d'arrivée (figure 2.19). On remplace donc l'angle (φ = **zéro**) par (θ). La huitième règle devient alors :

8. Si (cap1 = loin) et (cap2 = loin) et (cap3 = loin) alors (V = GV) et (φ = θ).

L'angle (θ) de la huitième règle ne fait pas partie de la sortie **Angle** du navigateur flou de Sugeno, et se calcul séparément et de la façon suivante :

Dans un repère orthonormé, chaque point est défini par son abscisse et son ordonnée. On peut calculer donc le module et l'argument de chaque point ainsi que la distance entre deux points et l'angle fait par la droite (p1, p2) et l'axe OX par ces deux formules :

- **Arg** = arctg [(y-yr) / (x-xr)]. (4)
- **Mod** = $\sqrt{[(x-xr) * (x-xr) + (y-yr) *(y-yr)]}$. (5)

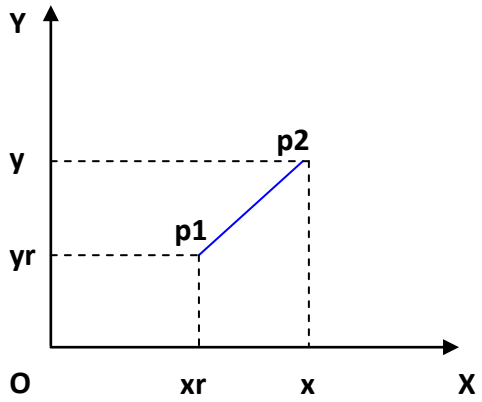


Figure 2.18 : Calcul de la distance et angle qui sépare le robot et le point d'arrivée.

Le module **Mod** traduit la distance parcourue par le robot pour qu'il se déplace du point **P1** pour arriver au point **P2**. L'argument **Arg** est l'angle (θ) de déviation pour que le robot s'oriente vert le point destinataire. Le robot initialement est dirigé vers le sens des **X** croissant : donc il fait un angle **0** avec l'axe **OX**.

Dans le cas où x = xr l'**Arg** est indéfini [voir l'équation (4)]. Pour éviter ce problème, on ajoute a x une distance de 10 cm, x est donc toujours supérieur a xr.

Exemple de navigation : Soit P1(0,0) et P2(500,600)

Mod = $\sqrt{[(500-0) * (500-0) + (600-0) * (600-0)]}$ = 781 cm.

Arg = arctg [(600-0) / (500-0)] = 50 °.

L'opération consiste simplement à faire progresser le robot autour des différents objets situés dans son espace de navigation en modifiant constamment la direction suivie par le robot (figure 2.19). Cette solution assure la navigation du robot à partir d'informations fournies par ses capteurs, Cependant, elle ne garantit pas des chemins optimaux.

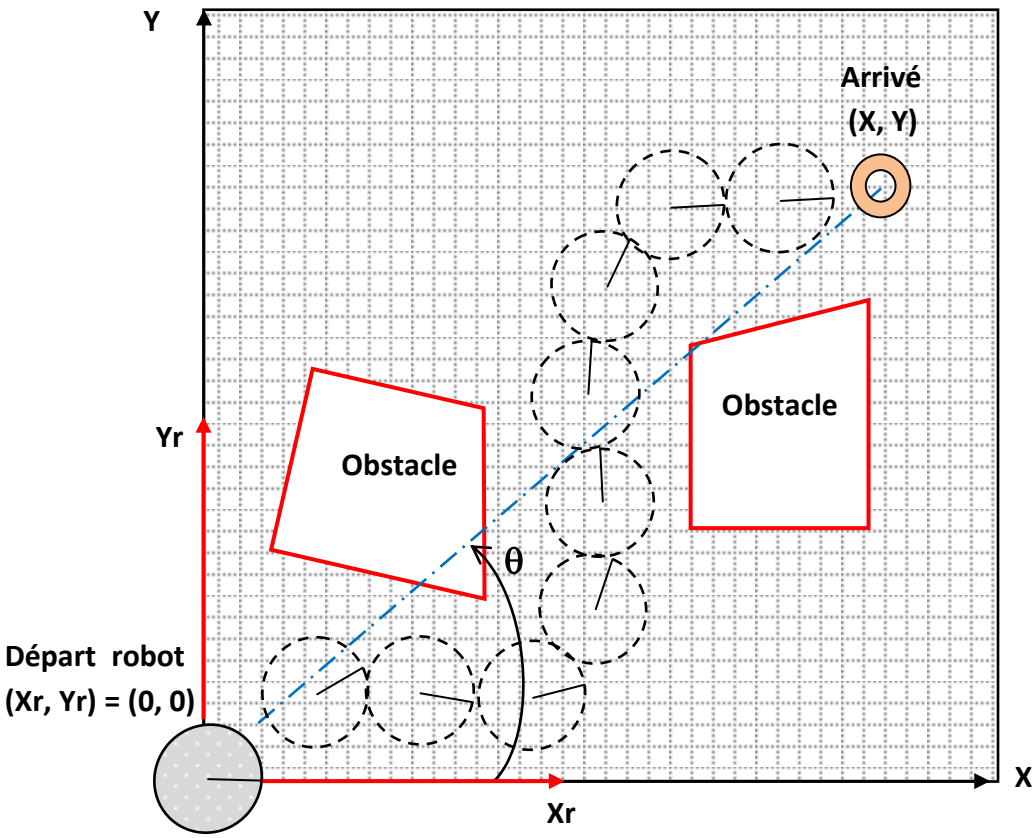
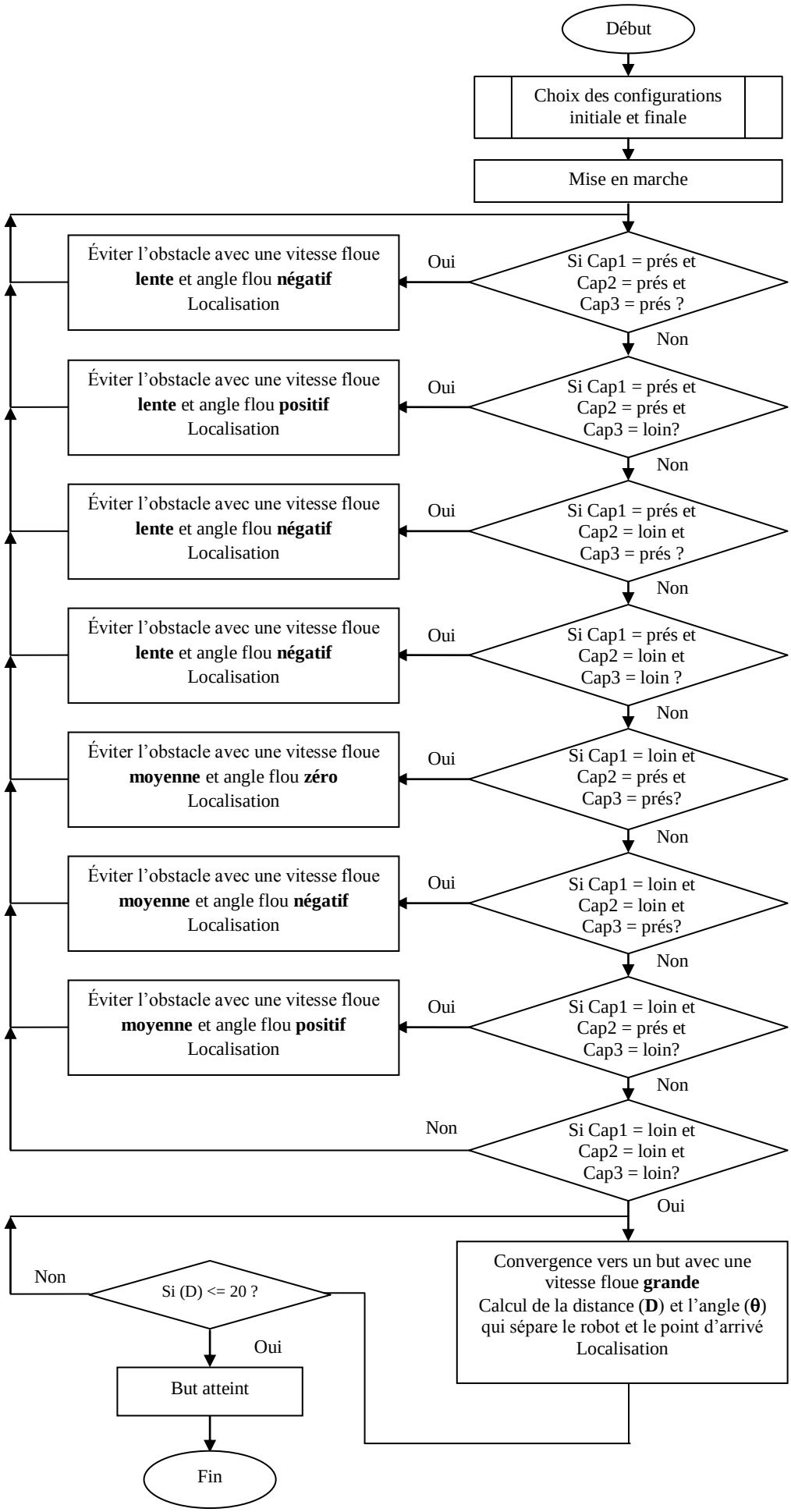


Figure 2.19 : Méthode de navigation réactive floue retenue pour notre robot ARP.

L'organigramme (1) suivant représente notre politique de navigation floue réactive. Avec θ et D sont respectivement l'angle et la distance qui sépare le robot et le point d'arrivée. Le robot s'arrête a une distance de D = 20 cm du point d'arrivée. Sachant que la localisation se fait au niveau du centre du robot et que ce dernier, en pratique, a un entraxe de 24 cm c.-à-d. le centre se trouve à 24/2 = 12 cm et que la distance D est mesurée a partir de ce centre, ce qui fait que le robot s'arrête a une distance de 20 – 12 = 8 cm du point d'arrivée.

Organigramme (1) de la navigation réactive floue du robot ARP :



2.4.2 Synthèse d'Identification et optimisation du navigateur flou du robot ARP :

Dans cette section, on va présenter un système hybride neuro-flou proposé par Jang [R. Jang 1] [R. Jang 4] connue sous le nom de (ANFIS : Adaptive-Network-based Fuzzy Inference System). L'architecture de l'ANFIS appartient à une classe de réseaux adaptatifs RBF fonctionnellement équivalent à un système d'inférence floue de type Sugeno [M. Sugeno 2] [Tak Sug], ce système hybride permet l'identification et l'optimisation de notre navigateur flou du robot ARP.

Dans le model de Jang, les fonctions d'appartenances utilisées sont des gaussiennes (ce qui est le cas aussi de notre navigateur flou), et chaque gaussienne dépend de deux paramètres (c : le centre de la gaussienne, σ : l'écart-type).

$$\mu_i(x) = \exp\left(\frac{-0.5*(x-c)^2}{\sigma^2}\right)$$

On peut aussi utiliser un deuxième modèle mathématique de la fonction gaussienne (fonction cloche):

$$\mu_i(x) = \frac{1}{1+\left|\frac{x-c_i}{a_i}\right|^{2bi}}$$

Le but est de faire varier les paramètres (c et σ) de chaque sous-ensemble flou de chaque entrée jusqu'à ce que la différence entre la sortie réelle et la sortie désirée (enregistré déjà manuellement : base d'apprentissage) soit plus au moins nulle.

Jang a proposé de représenter la base de règles de type Sugeno [M. Sugeno 2] [Tak Sug] par le réseau adaptatif de la figure (2.20). Rappel théorique des règles du système d'inference flou de Sugeno d'ordre 1:

- **SI** *X* est *Ai* **ET** *Y* est *Bi* **Alors** $Z = a_i * X + b_i * Y + c_i$

Pour notre cas les règles deviennent alors :

- **SI** *Cap1* est *près* **ET** *Cap2* est *près* **ET** *Cap3* est *près* **Alors**
  $Vitesse = k_{i1} * Cap1 + k_{i2} * Cap2 + k_{i3} * Cap3 + k_{i0}$ **ET**
  $Angle = l_{i1} * Cap1 + l_{i2} * Cap2 + l_{i3} * Cap3 + l_{i0}$

On remarque que notre système flou est compliqué, pour une seule règle nous avons 8 paramètres à identifier (k_{i0}, $k_{i1,}$ k_{i2} $k_{i3,}$ l_{i0}, $l_{i1,}$ l_{i2} l_{i3}) et pour 8 règles floues nous avons 64 paramètres à identifier. Pour simplifier le processus d'apprentissage nous avons donc

choisi d'optimiser notre navigateur flou au niveau seulement de la sortie floue *Vitesse* les règles deviennent alors :

- **SI** *Cap1* est *près* **ET** *Cap2* est *près* **ET** *Cap3* est *près* **Alors** *Vitesse* = k_{i1}* *Cap1* + k_{i2}* *Cap2* + k_{i3}* *Cap3*+ k_{i0}

  La sortie *Vitesse* est une fonction linéaire de type *Y= f (Cap1, Cap2, Cap3).*

Le modèle ANFIS [R. Jang 1] [R. Jang 4] [M. Negnev] est basé sur l'utilisation de réseaux multicouches où chaque cellule réalise une fonction propre respectant les paramètres qui lui ont été fournis. L'application des huit règles floues de notre navigateur flou présenté précédemment peut donc être représentée de la manière suivante (voir figure 2.20) :

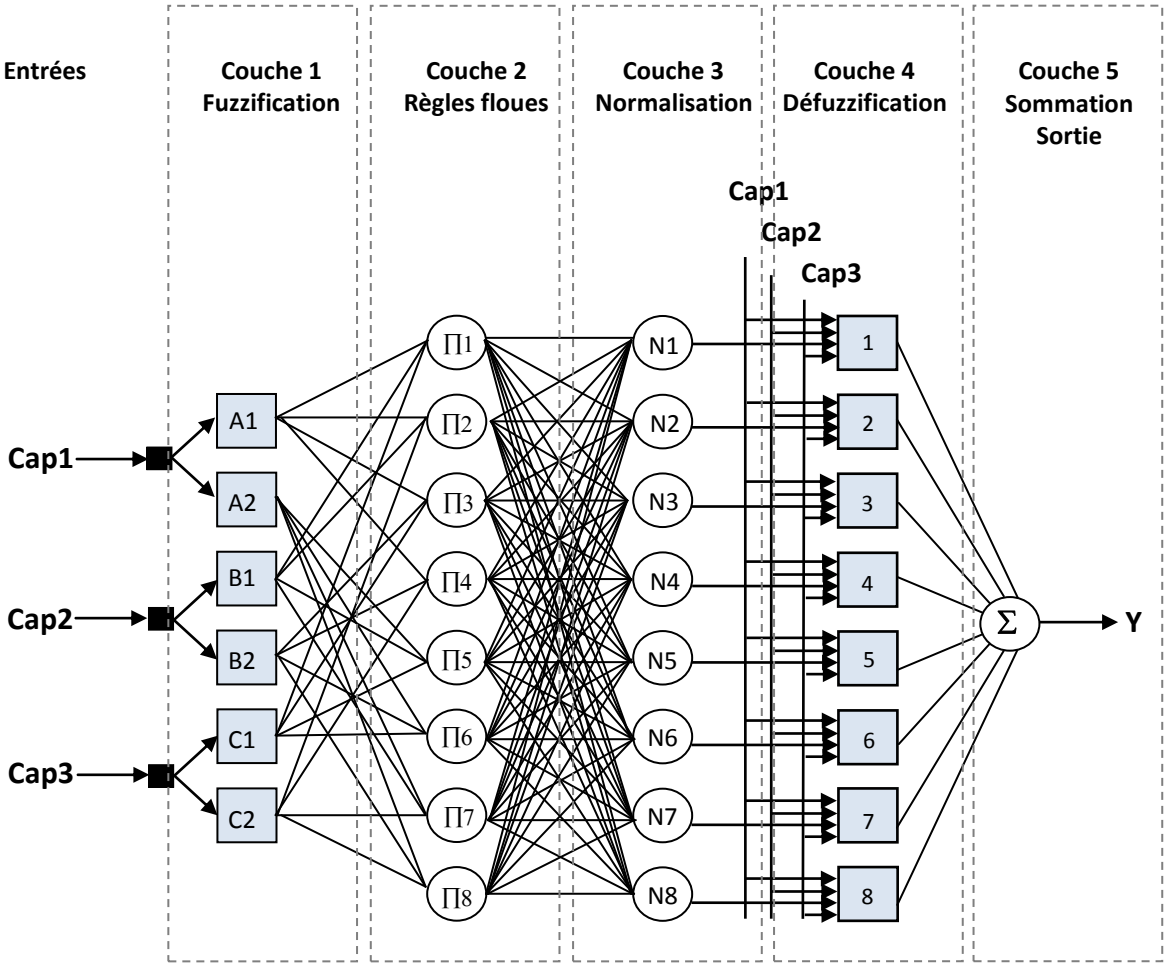


Figure 2.20 : Architecture du réseau ANFIS appliqué au navigateur flou du robot ARP.

Le réseau adaptatif ANFIS [R. Jang 1] [R. Jang 4] [M. Negnev] est un réseau multicouche dont les connexions ne sont pas pondérées, où ont toutes un poids de 1. Les nœuds sont de deux types différents selon leur fonctionnalité : les nœuds carrés (adaptatifs) contiennent des paramètres, et les nœuds circulaires (fixes) n'ont pas de paramètres. Toutefois chaque nœud (carré ou circulaire) applique une fonction sur ses signaux d'entrées.

Dans le réseau de la figure (2.20), les nœuds d'une même couche ont des fonctions issues d'une même famille que nous explicitons ci-dessous :

Couche 1 : Chaque nœud carré de cette couche est un neurone de fuzzification (figure 2.21), se comporte comme une fonction d'appartenance floue (gaussienne). Les sorties des nœuds de cette première couche représentent donc les degrés d'appartenance des entrées aux différentes ensembles flous.

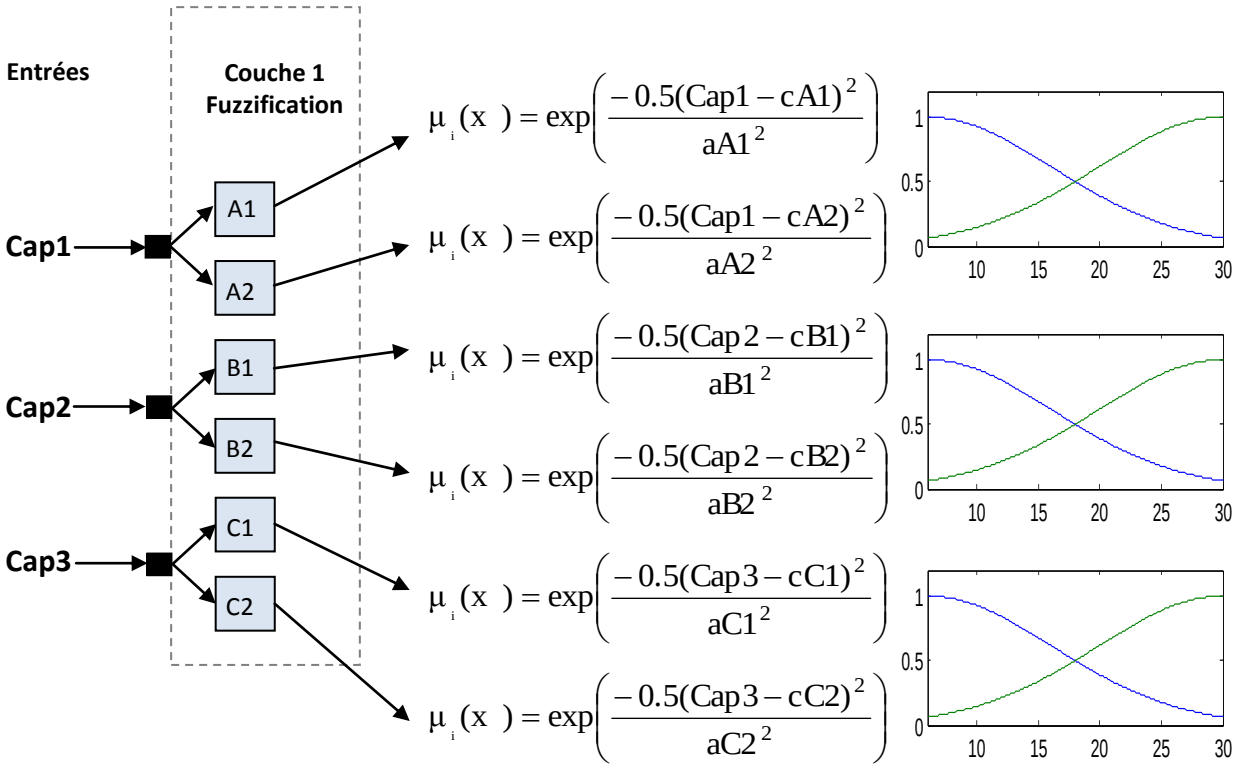


Figure 2.21 : La couche 1 du réseau ANFIS.

Couche 2 : Chaque nœud ***i*** de cette couche est un nœud circulaire appelé Π_i correspond à une règle floue de Sugeno (figure 2.22). Il reçoit les sorties des neurones de fuzzification et calcule son activation. La conjonction des antécédents est réalisée avec l'opérateur produit. Ainsi, la sortie du neurone *i* de la couche 2 est donnée par

$$\Pi_i^{(3)} = \prod_{j=1}^{k} \mu_{ji}^{(3)} \quad (6) \quad , \quad \textbf{exemple} : \prod_1 = \mu_{A1} * \mu_{B1} * \mu_{C1} = \mu_1$$

Où $\prod_1 = \mu_1$ représente le degré de vérité de la règle 1.

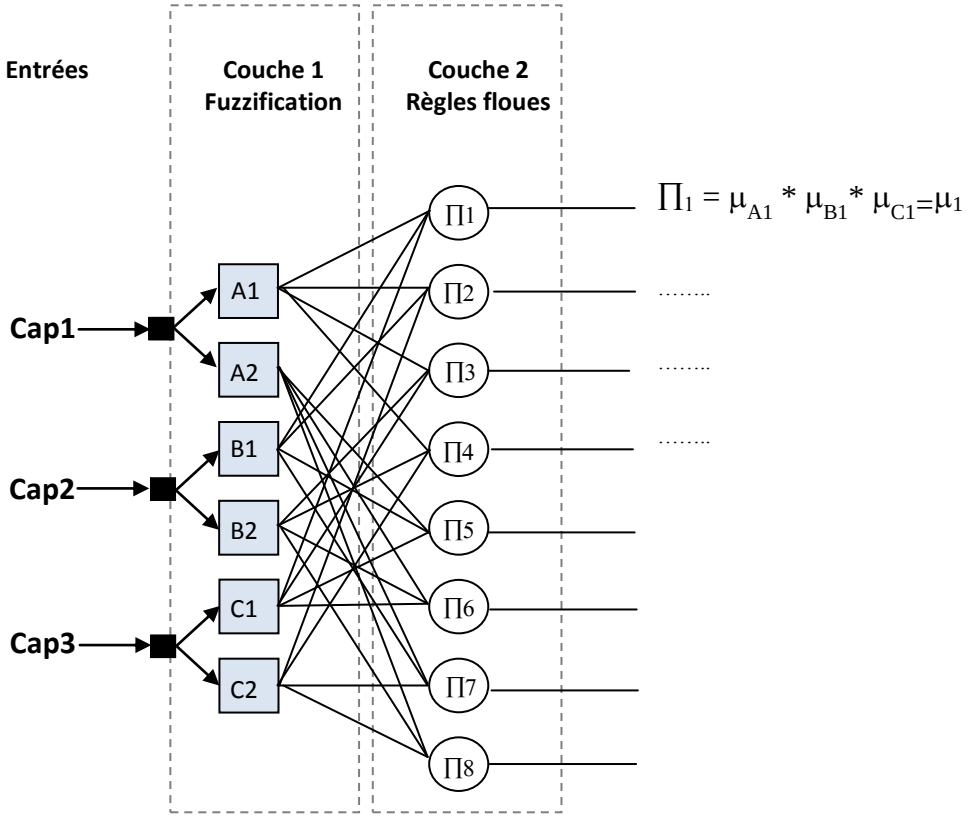


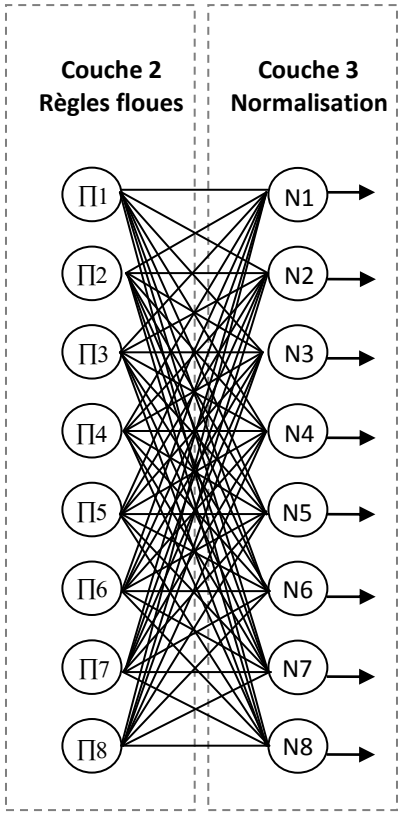
Figure 2.22 : La couche 2 du réseau ANFIS.

Couche 3 : Chaque neurone de cette couche est un nœud circulaire appelé N, qui calcule le degré de vérité normalisé d'une règle floue donnée (figure 2.23). La valeur obtenue représente la contribution de la règle floue au résultat final. Ainsi la sortie du neurone *i* de la couche 4 est :

$$y_i^{(4)} = \frac{x_{ii}^{(4)}}{\sum_{j=1}^{n} x_{ji}^{(4)}} = \frac{\mu_i}{\sum_{j=1}^{n} \mu_j} = \overline{\mu}_i \qquad (7)$$

Exemple :

$$y_{\mathrm{N1}}^{(4)} = \frac{\mu_1}{\mu_1 + \mu_2 + \mu_3 + \mu_4 + \mu_5 + \mu_6 + \mu_7 + \mu_8} = \overline{\mu}_1$$



$$y_{N1}^{(4)} = \frac{\mu_1}{\mu_1 + \mu_2 + \mu_3 + \mu_4 + \mu_5 + \mu_6 + \mu_7 + \mu_8} = \bar{\mu}_1$$

Figure 2.23 : La couche 3 du réseau ANFIS.

Couche 4 : Chaque nœud de cette couche est un nœud carré relié à un neurone de normalisation correspondant et aux entrées initiales du réseau (figure 2.24). Il calcule le conséquent pondéré de la règle sous jacente comme étant :

$$y_i^{(5)} = \bar{\mu}_i \left[k_{i0} + k_{i1} * cap1 + k_{i2} * cap2 + k_{i3} * cap3\right] \qquad (8)$$

Où les cap_i sont les entrées, et k_{i0}, $k_{i1,}$ k_{i2} et k_{i3} sont des paramètres du conséquent (paramètres de Sugeno à identifier) de la règle *i*.

Couche 5 : Comprend un seul neurone qui fournit la sortie de l'ANFIS [R. Jang 1] [R. Jang 4] [M. Negnev] en calculant la somme des sorties de tous les neurones de défuzzification (figure 2.24).

$$y = \sum_{i=1}^{n} x_i^{(6)} = \sum_{i=1}^{n} \bar{\mu}_i \left[k_{i0} + k_{i1} * cap1 + k_{i2} * cap2 + k_{i3} * cap3\right] \qquad (9)$$

Nous remarquons que la sortie globale du réseau est équivalente à la sotie du modèle de **Sugeno d'ordre 1** [M. Sugeno 2] [Tak Sug].

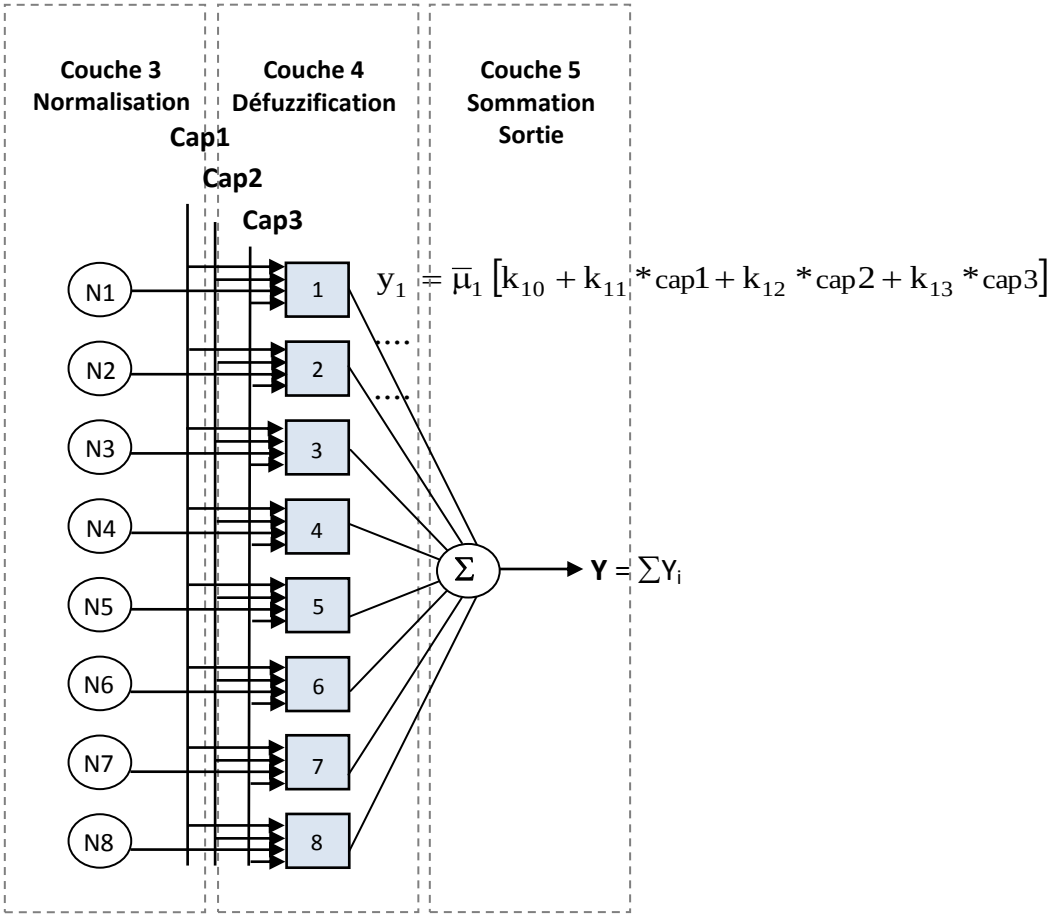


Figure 2.24 : les couches 4 et 5 du réseau ANFIS.

Le Nombre de nœuds de la couche 1 est toujours égal au nombre total de termes linguistiques définis. Le nombre de nœuds des couches 2, 3 et 4 est toujours égal au nombre de règles floues.

2.4.2.1 L'apprentissage du réseau ANFIS:

L'apprentissage [R. Jang 1] [R. Jang 4] [M. Negnev] à partir d'un ensemble de données (base de référence de la sortie vitesse désiré) concerne l'identification des 12 paramètres des prémisses (écarts type et centres des gaussiennes des 6 sous ensemble flou 2*6=12) et des 32 paramètres des conséquences (k_{i0} k_{i1} k_{i2} k_{i3} Pour i=1,8), la structure du réseau étant fixée. L'algorithme d'apprentissage commence par construire un réseau initial, ensuite nous appliquons une méthode d'apprentissage par rétropropagation de l'erreur. Jang a proposé d'utiliser une règle hybride d'apprentissage qui combine un algorithme de rétropropagation du gradient[1] avec une estimation par [J. L. Merri] moindres carrés[2].

[1] Rétropropagation du gradient : méthode de minimisation de l'erreur quadratique moyenne sur un ensemble de points. L'algorithme de rétropropagation du gradient est présenté a la fin de ce chapitre.
[2] Méthode des moindres carrés : un algorithme itératif de recherche de solution matricielle de type : $Y=A.k$, $k = [A^*.A]^{-1}.A^*.Y$ [J. L. Merri].

Chaque époque (itération) d'entraînement comprend une passe avant et une passe arrière. Les paramètres des conclusions des règles (paramètres des conséquences) qui sont linéaires par rapport à la sortie du système d'inférence floue sont estimés par la méthode des moindres carrées dans le sens forward (passe avant) du réseau, puis vient l'ajustement des paramètres des prémisses des règles par la méthode de rétropropagation du gradient dans le sens backward (passe arrière) du réseau, et ce, pour chaque itération de l'algorithme. Le tableau suivant nous résume les différentes étapes de l'algorithme :

Tableau 4 :

	Forward (une fois)	**Backward (une fois)**
Paramètres de la prémisse	Fixés	Rétropropagation du gradient
Paramètres de la conclusion	Moindres carrées	Fixés

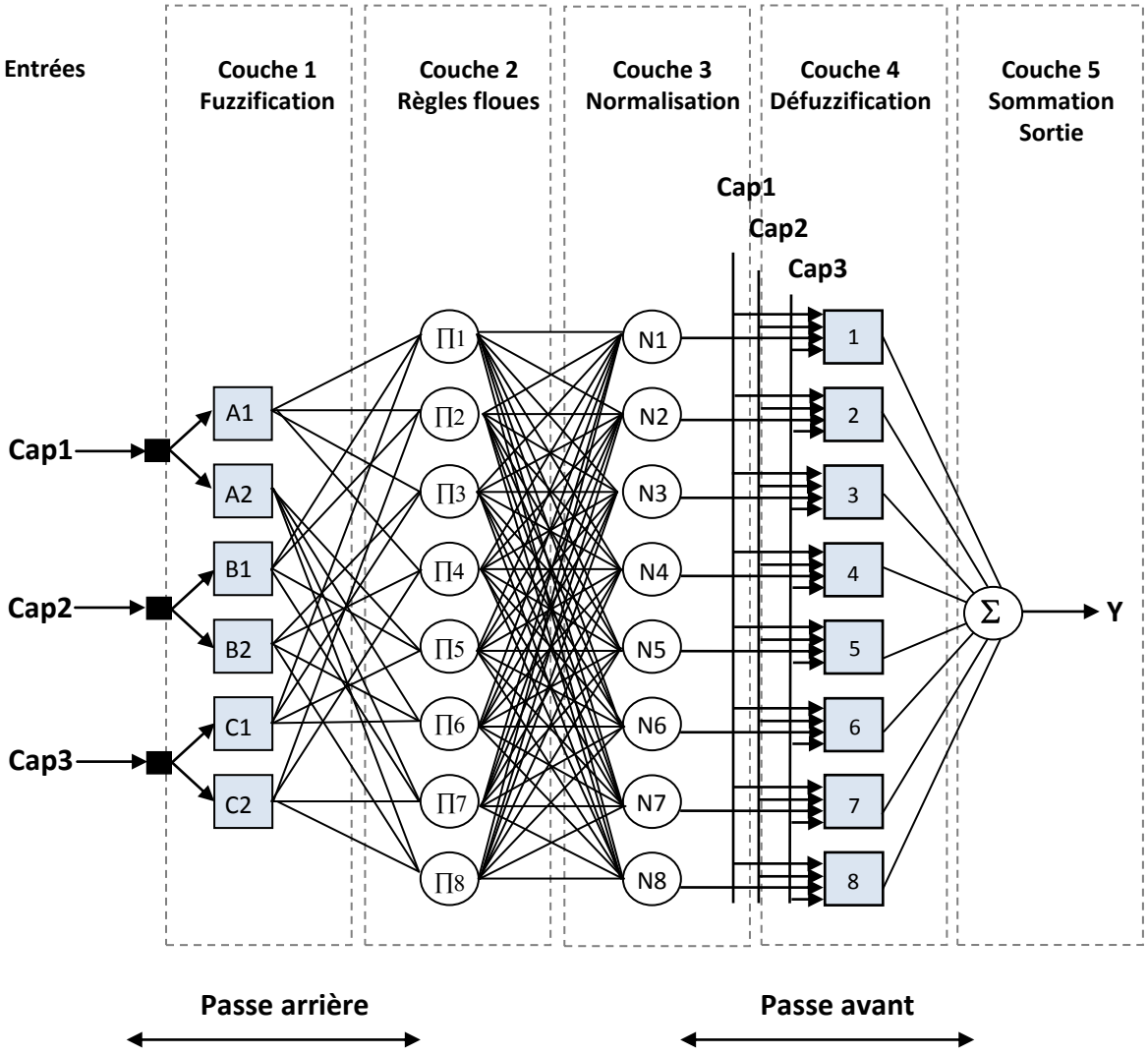


Figure 2.25 : Apprentissage du réseau ANFIS.

Passe avant : Lorsque les paramètres prémisses du réseau sont fixés (ceux des fonctions d'appartenance floues de la couche 1. Figure 2.21), la sortie du système ANFIS [M. Negnev] est linéaire par rapport aux paramètres conséquents (Figure 2.26). De fait, pour chaque couple (entrée / sortie désirée) de la base d'apprentissage, une fois les entrées propagées jusqu'à la couche 4, il est possible d'utiliser l'algorithme de l'estimation des moindres carrés afin de déterminer les paramètres conséquents du réseau.

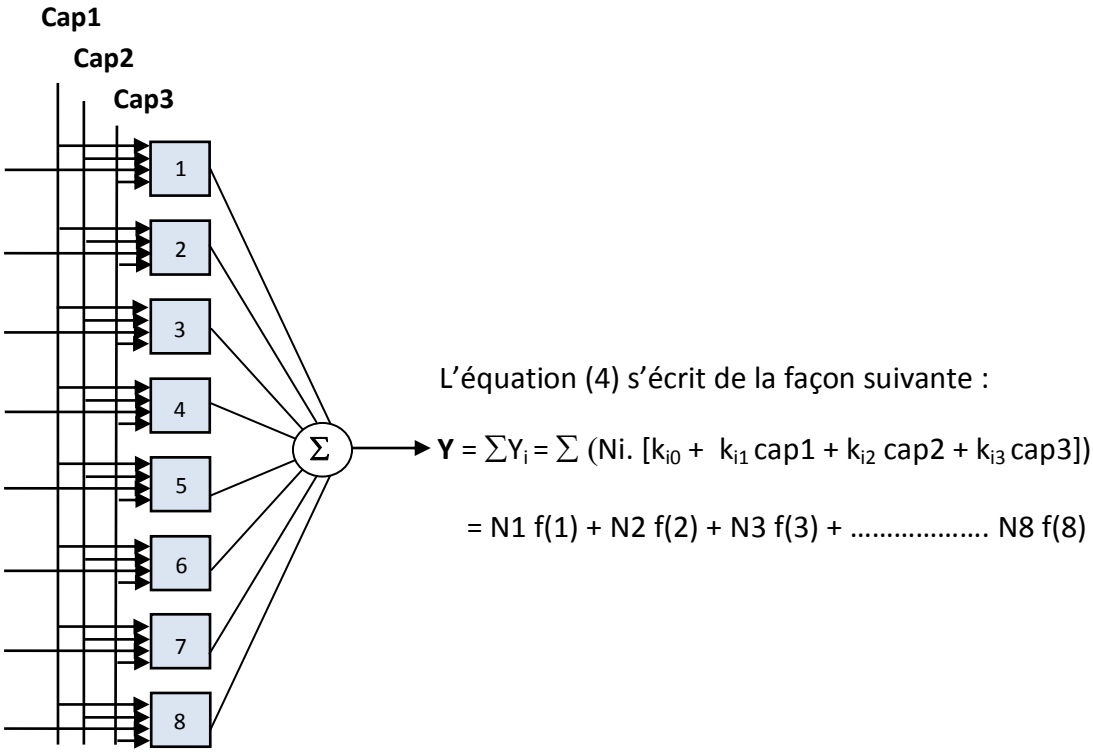
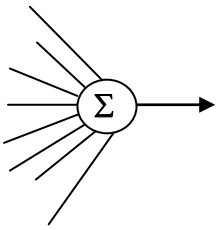


Figure 2.26 : Passe avant de l'apprentissage du réseau ANFIS.

Pour une base d'apprentissage d'une centaine de ligne, on remplace donc Y par le vecteur Yd qui est le vecteur de la sortie désiré (figure 2.27) :

Base d'apprentissage

	Cap1	Cap2	Cap3	Vitesse
1.	6	6	6	180
2.	7	7	7	194
	…			
	…			
100.	30 cm	30 cm	30 cm	255 correspond a 120 tr/mn



Yd(1) = N1 f(1) + N2 f(2) + N3 f(3) +..... N8 f(8)
Yd(2) = N1 f(1) + N2 f(2) + N3 f(3) +..... N8 f(8)
Yd(3) = N1 f(1) + N2 f(2) + N3 f(3) +..... N8 f(8)
....
...

Figure 2.27 : Phase d'estimation des paramètres des conséquents.

On obtient alors une fonction matricielle de type Yd=A.k :

$$\begin{bmatrix} yd(1) \\ yd(2) \\ .. \\ ... \\ \\ yd(P) \end{bmatrix} = \begin{bmatrix} \bar{\mu}1(1) & \bar{\mu}1(1)\ cap1..... & \bar{\mu}8(1)\ cap1...\bar{\mu}8(1)cap3 \\ \bar{\mu}1(2) & \bar{\mu}1(2)\ cap1..... & \bar{\mu}8(2)\ cap1...\bar{\mu}8(2)cap3 \\ ... & & \\ ... & & \\ \bar{\mu}1(P) & \bar{\mu}1(P)\ cap1..... & \bar{\mu}8(P)\ cap1...\bar{\mu}8(P)cap3 \end{bmatrix} \begin{bmatrix} k_{10} \\ k_{11} \\ .. \\ ... \\ \\ k_{83} \end{bmatrix}$$

La matrice A et le vecteur Ydésiré sont connus et le vecteur k (vecteur des paramètres des conséquent) est inconnu on peut alors estimer ($k = [A^*.A]^{-1}.A^*.Y$) en utilisant la méthode des moindre carré [J. L. Merri] avec A^* la pseudo-inverse de la matrice A.

Passe arrière : Une fois le vecteur k déterminé, le vecteur de sortie du réseau Y peut être calculé ainsi que le vecteur d'erreur associé, e :

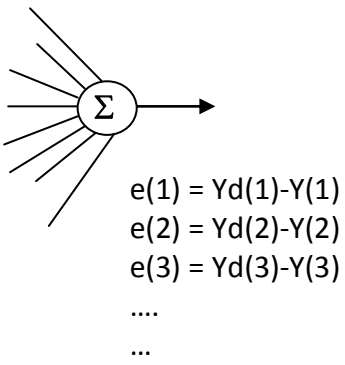


Figure 2.28 : Calcul du vecteur d'erreur.

Le vecteur d'erreur étant connu, l'erreur totale peut être calculée :

$$E_T = \frac{1}{2}\sum_{i=1}^{p} e^2(i) \qquad (10)$$

Le vecteur d'erreur et l'erreur totale sont calculés, les prémisses peuvent être obtenues par l'algorithme de rétropropagation du gradient [M. Negnev] [T. Bräunl]. À chaque itération d'apprentissage, ces paramètres sont corrigés de la forme suivante :

$$\mu_i(CAP_i) = exp\left(-0,5.\left[\frac{CAP_1 - (c_i + \Delta c_i)}{(a_i + \Delta a_i)} \right]^2 \right) \qquad (11)$$

Avec $\Delta x = -\eta . \delta E_T/\delta x$

Où η désigne le pas d'apprentissage [T. Bräunl], x le paramètre à modifier, et ($\delta E_T/\delta x$) la composante du gradient de l'erreur de prédiction inhérente au paramètre x, calculé par l'algorithme de rétropropagation du gradient.

2.4.2.2 Résultats :

Comme nous l'avons déjà cité dans l'objectif du travail présenté au début de ce mémoire, l'apprentissage on-line de notre navigateur flou est quasiment impossible vu les grands moyens matériels exigés par cette méthode. Nous avons donc choisi un apprentissage off-line. Bien que cette méthode ne requiert pas de moyens matériels mais exige des outils informatiques puissants. Vu l'architecture complexe de notre navigateur flou, l'apprentissage de ce dernier est réalisé sur **Matlab** [D. Bystrov] [J. L. Merri], un logiciel puissant pour le calcul numérique et matriciel et peut supporter de grandes bases d'apprentissage, et il dispose d'une syntaxe spécifique (langage de programmation de haut niveau).

Après 100 itérations d'apprentissage, l'erreur totale de notre réseau ANFIS présenté précédemment devient acceptable (figure 2.29):

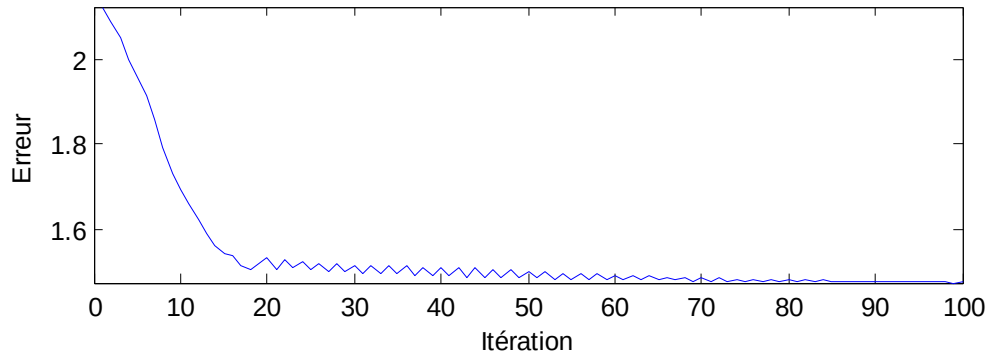


Figure 2.29 : L'erreur totale de l'apprentissage du navigateur flou du robot ARP.

Discussion 1 : Le processus d'apprentissage du réseau ANFIS (passe avant et passe arrière) se répète une centaine de fois, nous arrêtons ensuite l'apprentissage. La courbe d'erreur de la figure 2.29 montre que l'erreur totale de notre réseau tend vers le zéro au bout de 100 itérations.

On enregistre donc les nouvelles formes des fonctions d'appartenance des entrées capteurs. Les formes ont décalées et le chevauchement entre les sous ensemble flous a changé :

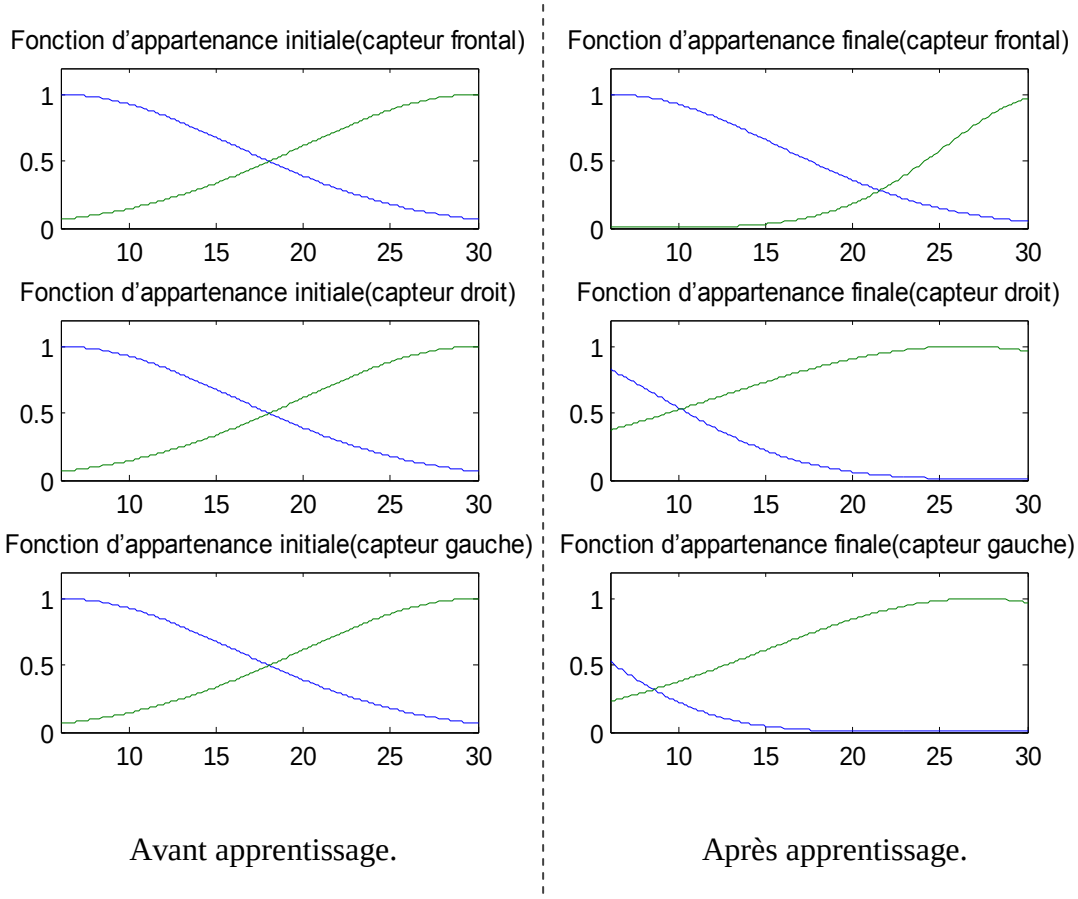


Figure 2.30 : Fonctions d'appartenances des trois entrées capteurs avant et après apprentissage.

Discussion 2 : Les trois fonctions d'appartenances à gauche de la figure 2.30 sont les fonctions d'appartenances initiales des trois entrées capteurs de notre navigateur flou présenté précédemment. Durant la passe arrière de l'apprentissage du réseau ANFIS [R. Jang 1] [R. Jang 4] [M. Negnev] l'algorithme de rétropropagation a ajusté les paramètres des prémisses (centres et écarts type des gaussiennes) des trois fonctions d'appartenances de notre navigateur flou. Après 100 itérations les fonctions d'appartenances ont changées de formes et sont affichées a droite de la figure 2.30. On peut alors enregistrer les nouveaux paramètres des prémisses en utilisant le workspace de **Matlab**:

Tableau 5 :

		Écart type	centre
Entrée Cap1	**Sous ensemble A1**	9,59	6,21
	Sous ensemble A2	6,23	31,48
Entrée Cap2	**Sous ensemble B1**	7,95	1,11
	Sous ensemble B2	14,38	26,26
Entrée Cap3	**Sous ensemble C1**	6,52	-1,36
	Sous ensemble C2	12,28	27,08

Discussion 3 : La base de données est écrite de la façon suivante : la première colonne représente l'entrée capteur frontale, la deuxième colonne représente l'entrée capteur droit, la troisième colonne représente l'entrée capteur gauche, et la quatrième colonne représente la vitesse désirée, la base de données est composé d'une centaine de ligne.

6	6	6	180
10	10	10	194
14	14	14	205
15	15	15	210
19	19	19	228
20	20	20	230
23	23	23	238
27	27	27	248
30 cm	30 cm	30 cm	255 correspond a 120 tr/mn
…			

Notre base de données à 4 dimensions (trois entrées et une sortie) ne peut pas être représentée graphiquement, on peut alors la représenter en 3 dimensions par paire d'entrées et une sortie (la sortie désirée en fonction de deux entrées) soit (Cap frontale - Cap droit- vitesse désirée) ou bien (Cap frontale - Cap gauche- vitesse désirée). La figure 2.31 représente un graphe en 3 dimensions de notre base d'apprentissage :

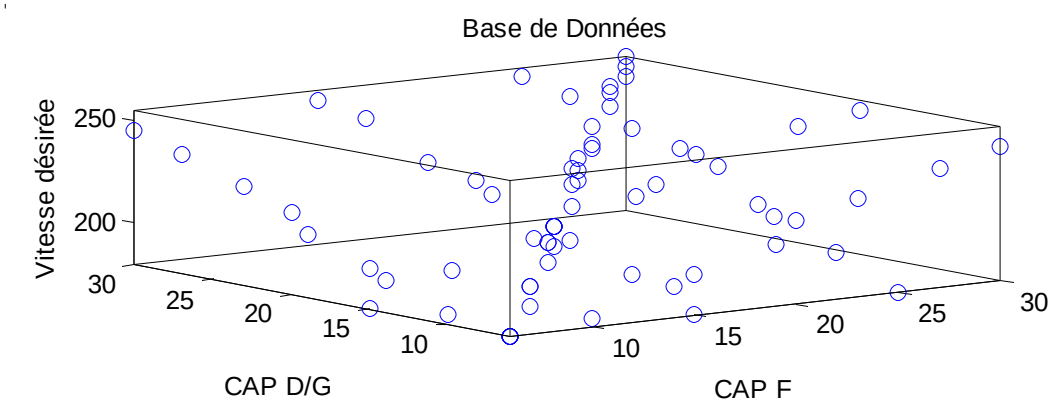


Figure 2.31 : Base de données de la sortie vitesse désirée (Cap frontale - Cap droit / Cap gauche- vitesse désirée).

Discussion 4 : Après l'apprentissage du réseau ANFIS [R. Jang 1] [R. Jang 4] [M. Negnev], la sortie réelle de ce dernier doit suivre en quelque sorte le même graphe de base de données présenté précédemment. En d'autre termes la vitesse réelle doit ressembler le mieux possible a la vitesse désirée. La figure 2.32 représente un graphe en 3 dimensions de la vitesse réelle en fonction des entrées. Ce graphe est appelé dans le jargon de la logique flou (surface de contrôle).

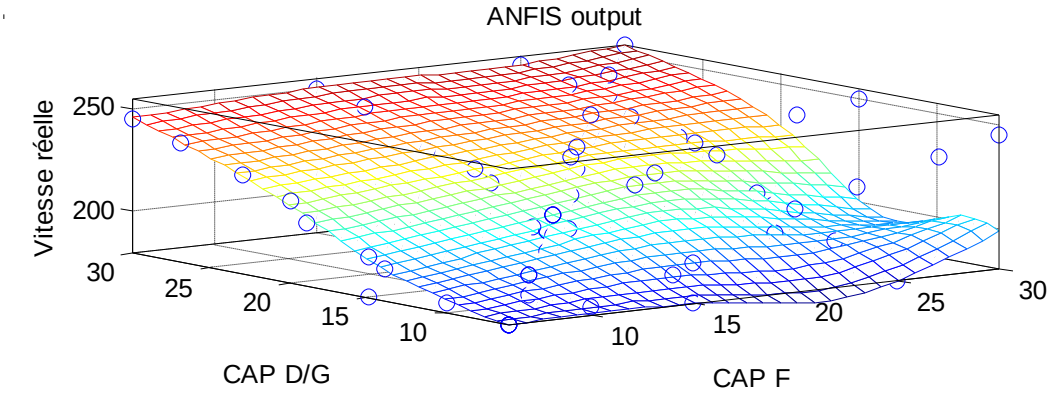


Figure 2.32 : Surface de contrôle de la sortie vitesse réelle du réseau ANFIS.

Discussion 5 : On remarque dans la figure 2.32 que la sortie réelle du réseau à bien suivit la sortie désirée sauf une légère différence (a droite de la figure) qui n'influe pas vraiment notre système. On peut améliorer la précision d'approximation de l'apprentissage en augmentant le nombre d'itération et le nombre de valeurs linguistiques par entrée (au lieu de 2 sous ensembles au niveau des entrées on met par exemple 3 sous ensembles flous). Pour notre cas les résultats obtenus sont largement suffisants.

Remarque : le réseau ANFIS est utilisé seulement en off-line pour l'identification et l'optimisation du navigateur flou une fois terminé on revient a notre navigateur flou, on remplace les anciens paramètres des prémisses par les nouveaux paramètres du **Tableau 5**. Et on remplace les 32 variables des paramètres des conséquentes (vecteur k_{ij}) cités précédemment par le **Tableau 6** enregistré à la fin de l'apprentissage (via le workspace de **Matlab**) :

Tableau 6 : vecteur k_{ij}.

-0,90	-17,81	1,62	108,67
3,99	-1,55	1,54	135,84
-1,14	-34,68	3,35	778,81
0,57	2,11	0,30	169,58
-5,05	343,40	-357,85	113,91
-22,02	4,32	24,71	142,82
50,84	-49,13	14,05	127,60
1,07	0,95	0,78	169,64

Exemple : On remplace les paramètres de sortie vitesse de la première règle $Vitesse = k_{i1} * Cap1 + k_{i2} * Cap2 + k_{i3} * Cap3 + k_{i0}$ par la première ligne du **Tableau 3** : $Vitesse = -0,90 * Cap1 - 17,81 * Cap2 + 1,62 * Cap3 + 108,67$. Et ainsi de suite pour les autres règles.

Remarque : Dans le cas d'un apprentissage on-line (ce qui n'est pas le cas de notre projet), le robot fait un test d'exploration de l'environnement et enregistre en temps réel la base de données. Ce type d'apprentissage est utilisé généralement pour le tracé de trajectoire (optimisation et planification de trajectoire) où on optimise aussi bien la position du robot que sa vitesse.

2.4.2.3 Résumé des algorithmes utilisés :

Algorithme 1 : Réseau ANFIS pour 3 entrées et une sortie.

---

1. **Initialisation et déclaration des variables et de la fonction gaussienne**
2. **1ère couche :** Fuzzification
   Pour i=1,2
   $O_i^1=\mu_{Ai}$ (cap1)
   $O_i^1=\mu_{Bi}$ (cap2)
   $O_i^1=\mu_{Ci}$ (cap3)
   Fin Pour
3. **Si** (passe arrière est bloqué) **Faire :**
4. **2ème couche :** Pondération des 8 règles floues
   1. $\Pi 1=\mu_{A1}$ (cap1).μ_{B1} (cap2).μ_{C1} (cap3)
   2. $\Pi 2=\mu_{A1}$ (cap1).μ_{B1} (cap2).μ_{C2} (cap3)
   3. $\Pi 3=\mu_{A1}$ (cap1).μ_{B2} (cap2).μ_{C1} (cap3)
   4. $\Pi 4=\mu_{A2}$ (cap1).μ_{B1} (cap2).μ_{C1} (cap3)
   5. $\Pi 5=\mu_{A1}$ (cap1).μ_{B2} (cap2).μ_{C2} (cap3)
   6. $\Pi 6=\mu_{A2}$ (cap1).μ_{B1} (cap2).μ_{C2} (cap3)
   7. $\Pi 7=\mu_{A2}$ (cap1).μ_{B2} (cap2).μ_{C1} (cap3)
   8. $\Pi 8=\mu_{A2}$ (cap1).μ_{B2} (cap2).μ_{C2} (cap3)
5. **3ème couche :** Normalisation
   Pour i=1,8
   $Ni=\Pi i / (\Pi 1+\Pi 2+\Pi 3+\Pi 4+\Pi 5+\Pi 6+\Pi 7+\Pi 8)$
   Fin Pour
6. **4ème couche :** Défuzzification
   Pour i=1,8
   $Y_i= Ni. [k_{i0} + k_{i1}\, cap1 + k_{i2}\, cap2 + k_{i3}\, cap3]$
   Fin Pour
7. **5ème couche :** Calcul de la sortie de Sugeno d'ordre 1
   Pour i=1,8
   $Y=\sum Y_i= \sum (Ni. [k_{i0} + k_{i1}\, cap1 + k_{i2}\, cap2 + k_{i3}\, cap3])$
   Fin Pour
   Calculer le vecteur k_{ij}
   $k = [A^*.A]^{-1}.A^*.Y$

Fin Faire

Algorithme 2 : rétropropagation du gradient

1. **Si** (passe avant est bloqué) **Faire :**
2. **1ère étape :** lecture de la base de données

   Pour i=1, P avec (P) ligne base d'apprentissage

   Yd(1)
   Yd(2)
   ….
   Yd(P)

   Fin Pour
3. **2ème étape :** Calculer les termes d'erreur.

   Vecteur d'erreur :

   Pour i=1, P

   e(1)=Yd(1)-Y
   e(2)=Yd(2)-Y
   ….
   e(P)=Yd(P)-Y

   Fin Pour

   Erreur Totale :

   Pour i=1, P

   $$E_T = \frac{1}{2}\sum_{i=1}^{p} e^2(i)$$

   Fin Pour
4. **3ème étape :** Mise à jour des centres et écarts type des fonctions gaussiennes des sous ensembles flou des entrées. Avec $\eta = 1$.

   $$\Delta a_i = -\eta.\frac{\delta E_T}{\delta a_i}, \quad \Delta c_i = -\eta.\frac{\delta E_T}{\delta c_i}$$

   Pour i=1,6

   $c_i' = c_i + \Delta c_i$

   $a_i' = a_i + \Delta a_i$

   Fin Pour

   Fin Faire
5. **4ème étape :** Répéter ce processus n itération jusqu'à ce que l'erreur E devienne acceptable

Algorithme 3 : Navigateur flou de Sugeno 3 entrées et 2 sorties.

Initialisation et déclaration des variables et de la fonction gaussienne

1. **Fuzzification :**

   Pour i=1,2

   $O_i^1=\mu_{Ai}$ (cap1)

   $O_i^1=\mu_{Bi}$ (cap2)

   $O_i^1=\mu_{Ci}$ (cap3)

   Fin Pour

2. **Règles floue :**

   1. $\Pi 1=\mu_{A1}$ (cap1).μ_{B1} (cap2).μ_{C1} (cap3)
   2. $\Pi 2=\mu_{A1}$ (cap1).μ_{B1} (cap2).μ_{C2} (cap3)
   3. $\Pi 3=\mu_{A1}$ (cap1).μ_{B2} (cap2).μ_{C1} (cap3)
   4. $\Pi 4=\mu_{A2}$ (cap1).μ_{B1} (cap2).μ_{C1} (cap3)
   5. $\Pi 5=\mu_{A1}$ (cap1).μ_{B2} (cap2).μ_{C2} (cap3)
   6. $\Pi 6=\mu_{A2}$ (cap1).μ_{B1} (cap2).μ_{C2} (cap3)
   7. $\Pi 7=\mu_{A2}$ (cap1).μ_{B2} (cap2).μ_{C1} (cap3)
   8. $\Pi 8=\mu_{A2}$ (cap1).μ_{B2} (cap2).μ_{C2} (cap3)

3. **Défuzzification :**

   Pour i=1,8

   Π= ($\Pi 1+\Pi 2+\Pi 3+\Pi 4+\Pi 5+\Pi 6+\Pi 7+\Pi 8$)

   $Ni=\Pi i / \Pi$

   Y_i= Ni. [k_{i0} + k_{i1} cap1 + k_{i2} cap2 + k_{i3} cap3] k_{ij} vecteur connu.

   Sortie Vitesse type Sugeno d'ordre 1 = $\sum Y_i$

   Fin Pour

   Agrégation (angle) de type Sugeno d'ordre 0 :

   N = valeur moyenne de la sortie négative de l'angle.

   P = valeur moyenne de la sortie positive de l'angle.

   Z = valeur moyenne de la sortie zéro de l'angle.

   Sortie Angle type Sugeno d'ordre 0 =

   [(N. $\Pi 1$) + (N. $\Pi 2$) + (P. $\Pi 3$) + (N. $\Pi 4$) + (Z. $\Pi 5$) + (N. $\Pi 6$) + (P. $\Pi 7$) + (Z. $\Pi 8$)] / Π.

Note : l'organigramme (1) (présenté à la page 41) représente de façon générale le processus de navigation réactive floue (navigation vers un but dans un environnement inconnu), ce dernier est basé sur **l'algorithme 3** qui représente le navigateur flou de Sugeno (contrôle en vitesse et angle de braquage du robot ARP).

2.5 Conclusion :

Nous nous sommes intéressés au cours de cette première partie de ce chapitre a présenter quelques théories de logique floue et de réseaux de neurones, nous avons aussi présenté le navigateur flou réactif retenu pour notre approche, ainsi le modèle neuro-flou adopté pour l'identification et l'optimisation de notre navigateur flou, nous avons démontré que le système ANFIS permet l'apprentissage automatique des paramètres du navigateur flou, et que cette technique d'apprentissage opère en fonction de l'information locale et produit uniquement des changements locaux dans le navigateur flou d'origine. Dans le prochain chapitre, nous allons tester et valider notre approche de navigateur flou réactif sur une plate forme de simulation en 3D que nous avons réalisée.

Chapitre 3

Sommaire partiel 3:

3

Simulation graphique

La simulation en robotique mobile est un outil précieux pour le développement d'algorithmes pour robots mobiles car elle permet en toute sécurité de valider les principes mis en œuvre. Nous présentons dans ce chapitre, la plateforme de simulation réalisée ainsi que les tests de navigation floue du robot ARP effectués sur cette plateforme.

3.1 Définition :

La simulation d'un système est une abstraction du comportement réel de ce système qui permet de mieux le comprendre avant de le construire. Dans le contexte de la robotique mobile, la simulation [T. Bräunl] est un outil incontournable pour l'étude de la coopération entre agents. En supprimant les difficultés matérielles liées aux expérimentations réelles, elle permet de focaliser l'attention sur les aspects comportementaux et organisationnels du système. Les principaux avantages à l'utilisation d'une simulation sont :

- Les expériences sont déterministes et reproductibles, ce qui s'avère très utile lors de la mise au point des algorithmes ou lorsqu'on souhaite effectuer des tests dans des conditions fixes.
- Il est possible d'effectuer des expériences longues sans se soucier des problèmes d'autonomie liés aux robots réels.
- La simulation permet aussi de réduire les risques entourant la programmation et la mise en production d'un système robotique.

3.2 Présentation de la plateforme de simulation réalisée :

Lors de la conception de notre plateforme de simulation, nos préoccupations ont été les suivantes :

- Assurer l'évolution simultanée et cohérente d'un ensemble d'agents indépendants évoluant dans un environnement continu.

- Offrir une grande flexibilité au niveau des possibilités d'interprétation des données et ainsi permettre l'étude du système selon plusieurs points de vue, en termes d'affichage (3D, 2D…) ou d'analyse de l'état interne des agents (affichage des résultats, calcul, etc.).

Pour réaliser notre plateforme de simulation spécialement dédiée à l'étude de la navigation réactive floue du robot ARP, nous avons utilisé le logiciel **Blender**[1] [L. Flavell], un logiciel *(Free Open Source Software)*[2] développé initialement pour les jeux vidéos et les animations 3D et effets spéciaux cinématographiques. De plus en plus de laboratoires utilisent **Blender** pour des projets de recherche et d'enseignement en robotique, il permet de programmer et de construire rapidement un monde virtuel en 3D. Et même modifier son programme source (dans le cas ou on veut améliorer le logiciel en rajoutant d'autres options). Il est aussi possible d'importer des objets 3D complexes pour construire l'environnement ou le robot depuis n'importe quel outil de modélisation 3D à travers le format **VRML** *(Virtual Reality Modeling Language)*. Ou bien le contraire, exporter vers d'autre outils par exemple (**Matlab**). **Blender** dispose en plus de la fenêtre graphique une fenêtre de programmation en langage **Python** qui permet d'accéder aux valeurs de chacun des blocs de l'interface graphique grâce à une autre fenêtre appelé (Éditeur logique).

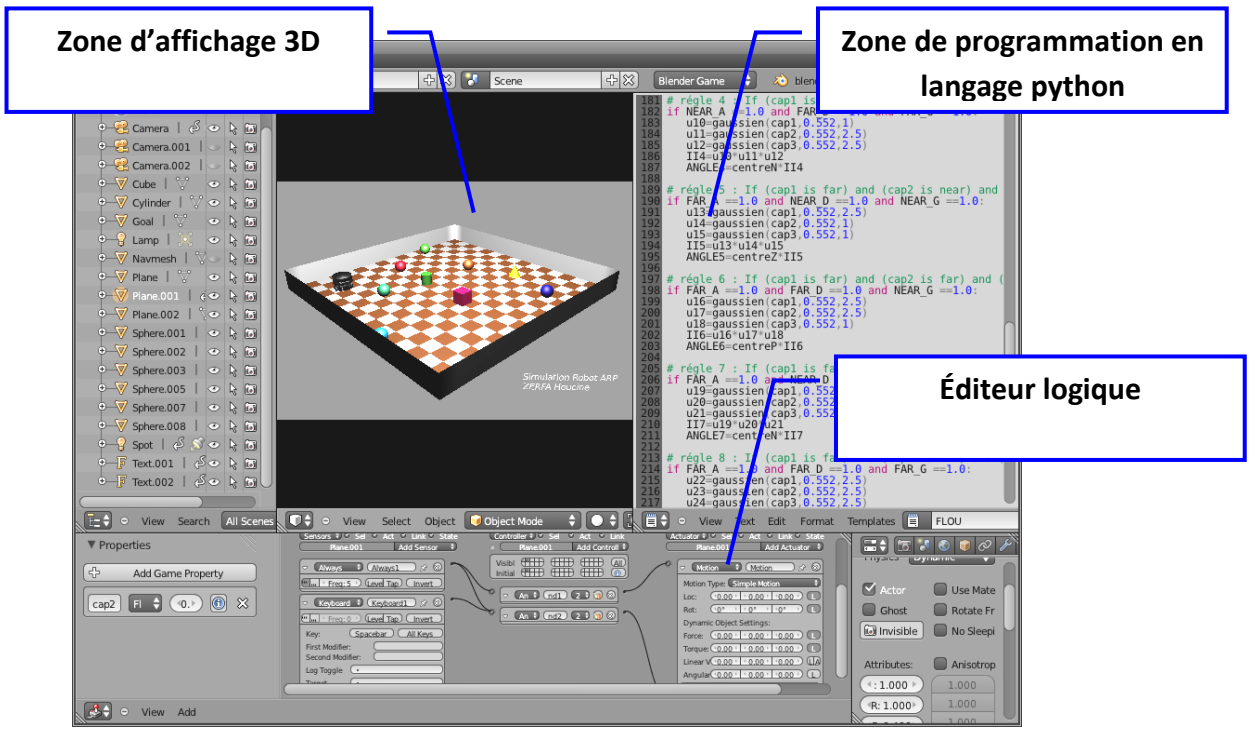


Figure 3.1 : Présentation du logiciel Blender.

[1] http://www.blender.org/
[2] Free Open Source Software : logiciel gratuit et modifiable.

Nous avons donc réalisé une plateforme de simulation 3D permettant de visualiser et de simuler le comportement du robot ARP en phase d'exécution des tâches. Nous avons cherché à reproduire des conditions proches de l'environnement réel (Architecture du robot ARP, obstacles, terrain…) on néglige (les glissements des roues, le poids du robot,..). Nous avons aussi créé plusieurs environnements de simulation afin de tester les capacités de notre système à faire évoluer des algorithmes adaptés à différentes conditions. D'une manière générale, chaque environnement est un terrain damier (pour le repérage du déplacement du robot), contenant plusieurs obstacles, ces derniers sont des blocs cubiques, sphériques ou cylindriques.

La plateforme de simulation 3D que nous avons développée englobe deux zones principales. Une zone d'affichage (voir figure 3.3) de résultats indiquant les analyses de données concernant la navigation du robot ARP (la vitesse du robot, sa position, son angle de braquage,…) et une zone de visualisation graphique (voir figure 3.2) avec perspective camera[3], on peut aussi afficher le tracé de trajectoire effectuée par le robot.

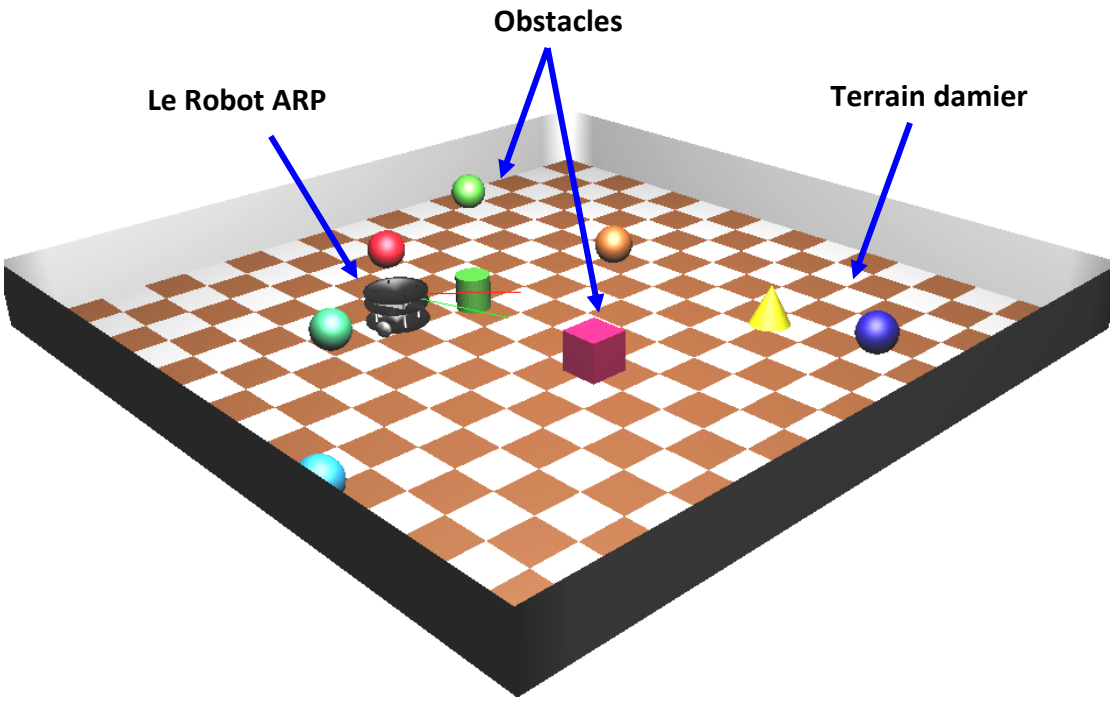


Figure 3.2 : Plateforme de simulation 3D du robot ARP.

[3] Perspective camera : angle de vue.

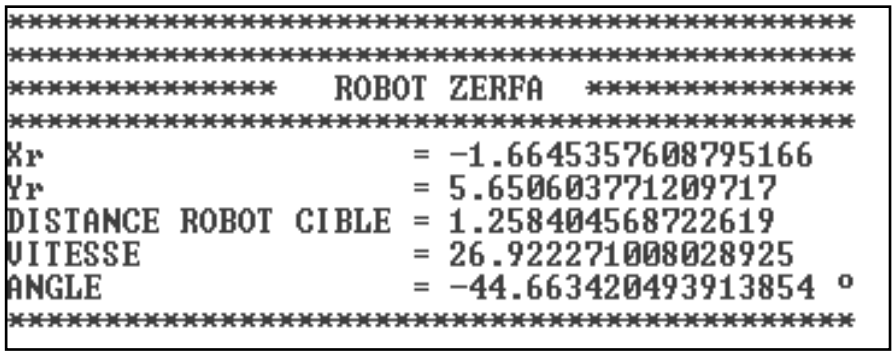


Figure 3.3 : Fenêtre d'affichage de données concernant la navigation du robot ARP (position du robot, angle de braquage, vitesse, distance entre le robot et le point d'arrivée).

3.2.1 Résultats de simulation :

Après avoir réalisé la plateforme de simulation, nous pouvons désormais, tester les différents comportements du robot ARP, nous allons examiner le navigateur flou du robot ARP présenté déjà au deuxième chapitre. Les différents tests de simulation sont présentés selon un ordre croissant du niveau de complexité (contrainte d'environnement). Mais d'abord, un rappel de l'objectif du travail.

Objectif : le robot doit se déplacer d'une position initiale A à une position finale B connue de manière autonome, tout en évitant les obstacles dans un environnement inconnu (le robot ignore complètement la structure de son environnement et ne fait aucune planification préliminaire). Les contraintes sont :

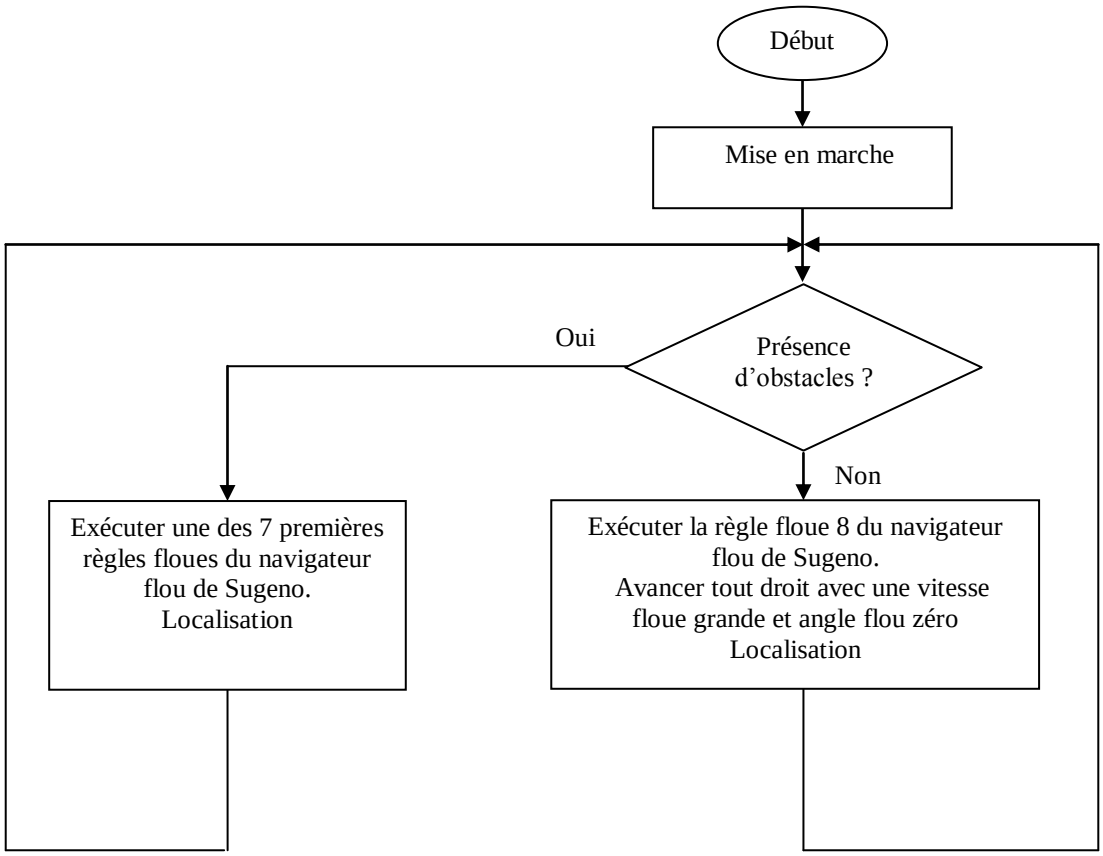
- **Les contraintes liées au robot :** le robot mobile ARP est de type uni-cycle non holonome, il ne peut donc pas se déplacer selon son (axe Y) mais il peut seulement se déplacer selon son (axe X) et faire une rotation autour de son (axe Z). Le robot possède seulement trois capteurs infrarouges linéaires (le champ de vision est limité) dans la zone frontale, gauche et droite.

- **Les contraintes liées à l'environnement :** Les obstacles sont des contraintes définies dans un espace de deux dimensions (X, Y). La présence d'obstacles dans l'environnement rend la tache de navigation difficile, le robot doit éviter ces obstacles et atteindre son but final.

3.2.1.1 Expérimentation 1 :

Dans cette section, nous nous intéressons au comportement d'évitement d'obstacles qui est un comportement de base dans la navigation du robot ARP. Nous allons tester ce comportement en utilisant les huit règles floues de notre navigateur flou de Sugeno [Tak Sug] [M. Sugeno 2] présenté au deuxième chapitre (page 38) sans changer la sortie angle (zéro) de la huitième règle. Le robot fait donc une exploration du terrain sans se diriger vers un point précis. **L'organigramme (1)** présenté au deuxième chapitre (page 41) devient alors :

Organigramme (2) de la navigation réactive floue (exploration du terrain) du robot ARP:



La figure 3.4 présente une simulation de ce comportement dans un terrain encombré par des obstacles.

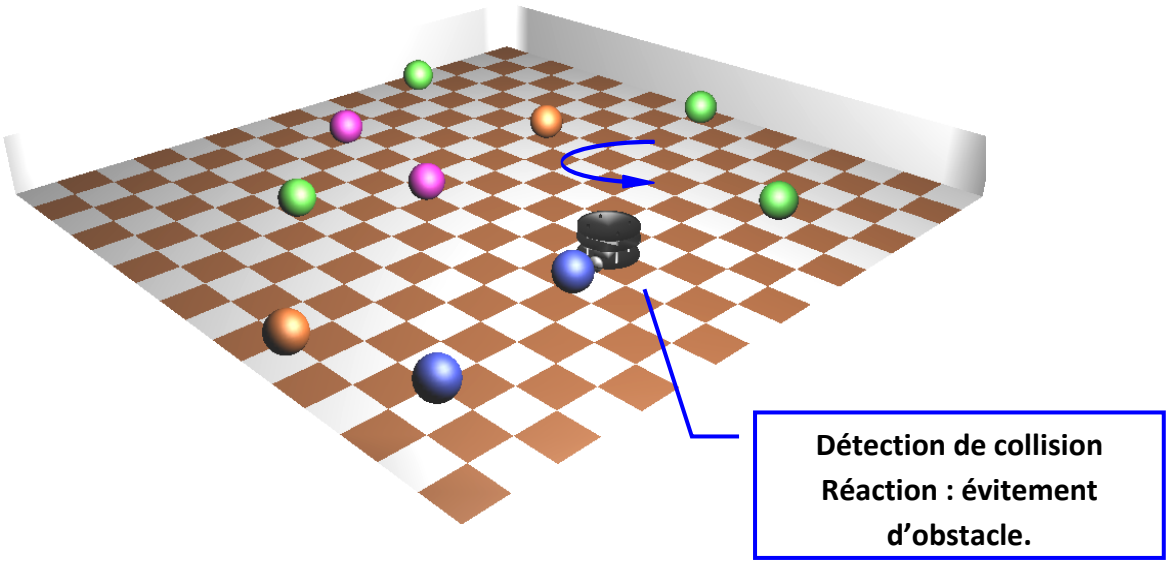


Figure 3.4 : (Environnement 1) Exploration du terrain par le robot ARP.

Discussion 1: Dans le scénario de la figure 3.4, le navigateur flou de Sugeno [Tak Sug] [M. Sugeno 2] implémenté est déjà optimisé (ses paramètres de vitesse ont été déjà identifiés voir chapitre 2 page 54). Le robot fait une exploration du terrain et se dirige aléatoirement, tout d'abord on remarque que le robot n'entre jamais en collision avec les obstacles, et qu'il parvient bien à les éviter en changeant constamment sa vitesse et son angle de braquage. Cette simulation valide l'efficacité du comportement flou d'évitement d'obstacles.

3.2.1.2 Expérimentation 2 :

Dans cette section, nous réunirons le comportement d'évitement d'obstacle et le comportement de convergence vers un but. Le robot doit donc aller d'un point A à un point B de coordonnés connus tout en évitant les obstacles. Le navigateur flou de Sugeno implémenté est déjà optimisé et la politique de navigation est celle de **l'organigramme (1)** présenté au deuxième chapitre (page 41) on remplace donc l'angle de la huitième règle floue par l'angle (θ) (voir chapitre 2 page 39).

Un autre environnement de simulation est envisagé cette fois ci, nous avons rajouté une cible sous forme conique (pour le repérage du point d'arrivée), et deux lignes indicatrices, une ligne rouge pour l'angle (θ) utilisé dans la huitième règle floue, et une ligne verte pour l'angle utilisé dans les septes premières règles floue. La présence des obstacles dans l'environnement de navigation augmente la complexité du système. Les positions des obstacles ne changent pas durant la simulation (environnement statique), sauf la cible est dynamique on peut la déplacer a l'aide d'un simple clique de la souris. Pendant sa navigation vers une cible, le robot utilise, en plus du comportement de convergence vers un point but, le comportement d'évitement d'obstacles présenté précédemment.

Test 1 :

Dans la simulation qui suit, l'environnement est peu encombré et le robot doit se rendre par exemple en (10,7) en carreau, en partant de (0,0) carreau.

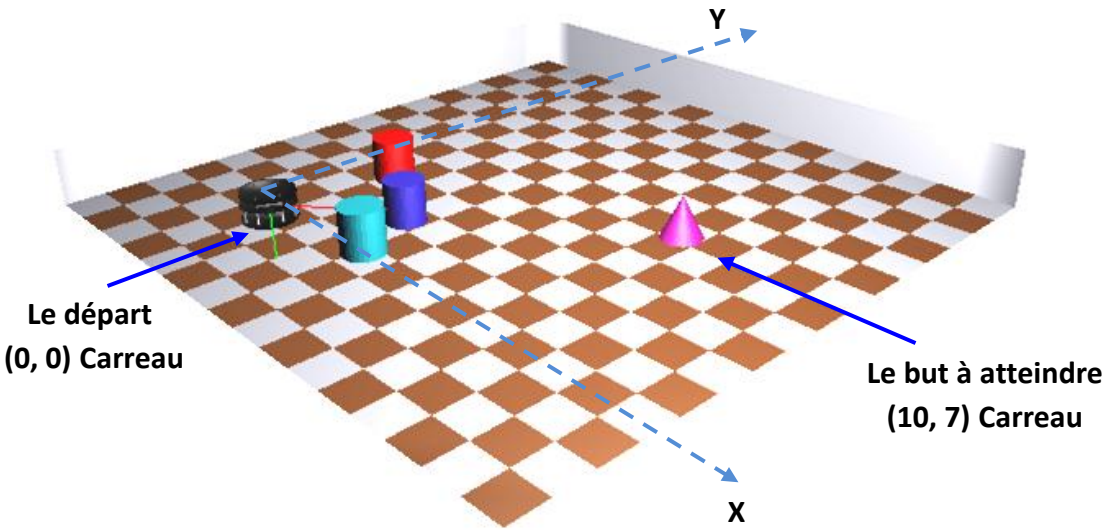


Figure 3.5 : (Environnement 2) Vue panoramique de la simulation (Convergence vers un point but dans un environnement peu encombré).

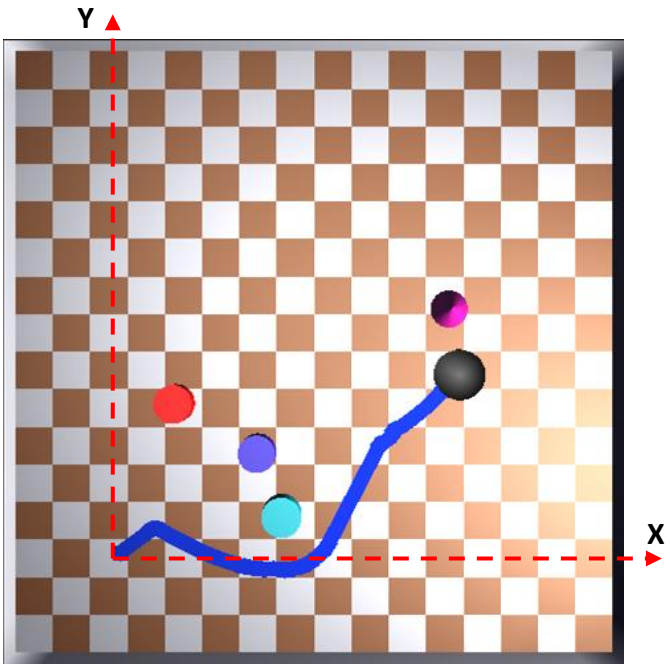


Figure 3.6 : (Environnement 2) Trajectoire du robot ARP enregistré dans un environnement peu encombré.

Discussion 2: Dans le scénario de la figure 3.6, la trace des déplacements effectués par le robot est représentée en bleu. Le robot progresse autour des différents obstacles situés dans son espace d'évolution en modifiant constamment sa direction par rapport au but visé qu'il finit par atteindre. Sur ce même environnement nous avons changé le point d'arrivée a plusieurs reprise, Encore une fois, cette simulation met en évidence l'efficacité du navigateur flou le robot parvient toujours à éviter les obstacles et atteindre son but dans un environnement peu encombré.

Test 2 :

Dans ce qui suit nous augmentons la complexité de l'environnement d'évolution du robot ARP en augmentant le nombre d'obstacles. Le robot doit se rendre par exemple en (8,11) en partant de (0,0).

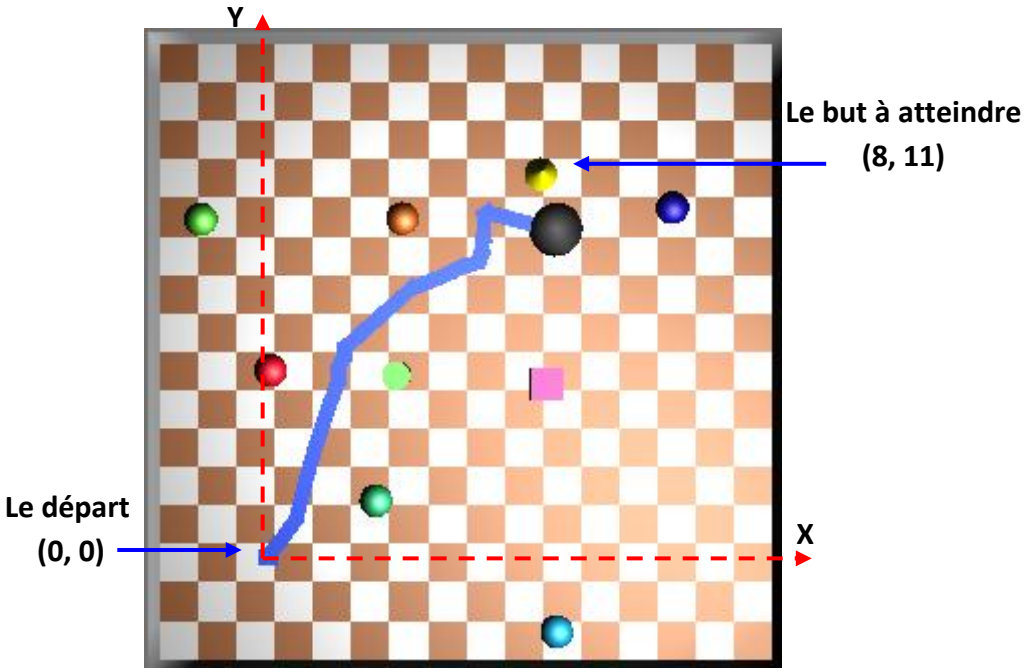


Figure 3.7 : (Environnement 3) Trajectoire du robot ARP enregistré dans un environnement encombré.

Discussion 3: Dans le scénario de la figure 3.6, l'environnement est encombré par une dizaine d'obstacles. Nous avons changé le point d'arrivée à plusieurs reprises et le robot parvient toujours à atteindre son but en évitant les obstacles. Les résultats enregistrés de la navigation réactive floue du robot ARP sont satisfaisant.

Test 3 :

Dans ce test le robot part de (0,0) et doit par exemple se rendre en (8,11). L'environnement est encombré par un obstacle (un mur homogène) en forme de U.

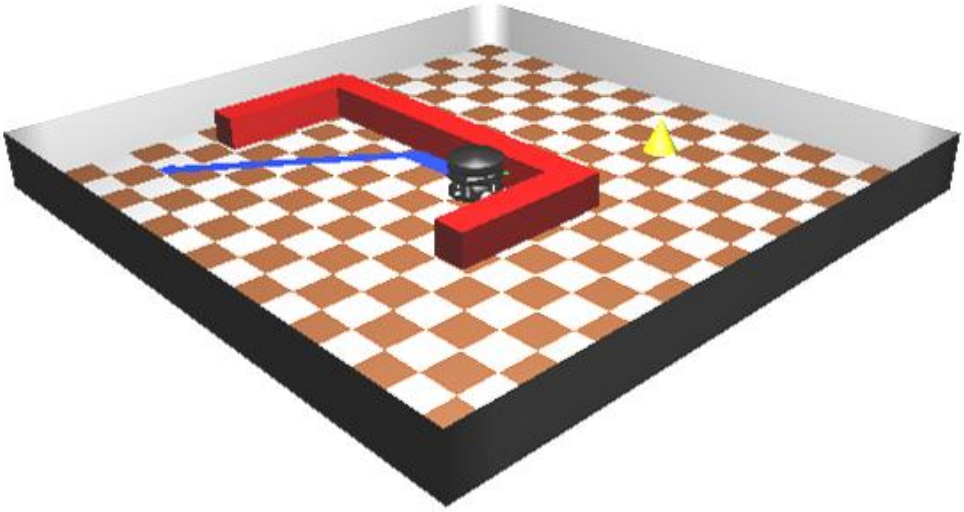


Figure 3.8 : (Environnement 4) navigation du robot ARP dans un environnement encombré par un obstacle en forme de U.

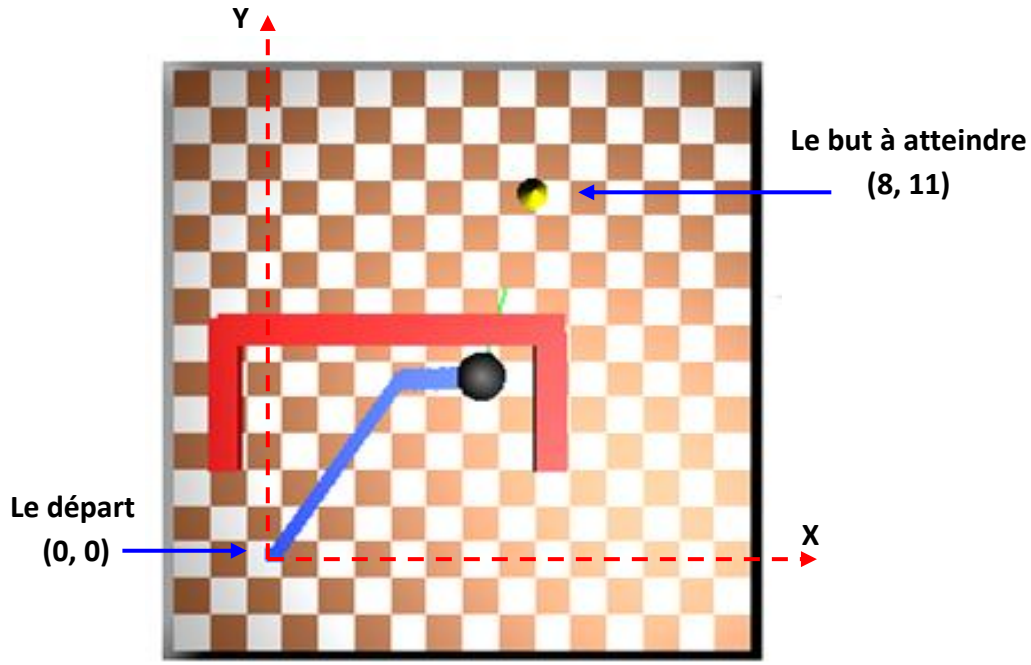


Figure 3.9 : (Environnement 4) Trajectoire du robot ARP enregistré dans un environnement encombré par un obstacle en forme de U.

Discussion 4: la figure 3.9 représente les résultats d'un scénario de navigation dans lequel le robot se retrouve piégé dans une situation de blocage. Le robot passe son temps à commuter entre les comportements (évitement d'obstacles et convergence vers un but) et le robot ne parvient pas à atteindre le but désigné. Ce genre de problème est très fréquent dans la navigation réactive des robots mobiles, il existe une solution pour sortir de ce piège, W.L. Xu [W.L. Xu] a proposé une technique qui utilise des cibles virtuelles pour sortir le robot du piège dans lequel il est tombé, à partir du moment où il reconnaît être tombé dans un piège.

Remarque : Dans toutes les simulations présentées dans ce chapitre, le navigateur flou a généré des commandes adéquates de vitesse et d'angle au robot ARP. Nous rappelons que ce navigateur est déjà optimisé (les paramètres de vitesse ont été déjà identifiés voir chapitre 2 page 54). Et que les paramètres utilisés dans la simulation sont celles de ce navigateur flou [3 entrées capteurs et 2 sorties (vitesse et angle)] a une seule différence d'échelle par exemple au niveau de la vitesse : la valeur de 255 tr/mn de la vitesse présenté au chapitre 2 page 37, correspond a une vitesse de 40 cm/s en simulation.

Nous avons aussi effectué un test de simulation où le navigateur flou du robot ARP n'est pas optimisé c'est-à-dire sa vitesse n'a pas été identifiée, la sortie vitesse du navigateur flou est donc de type Sugeno d'ordre 0 (voir chapitre 2 page 31). Ce test de simulation à marqué dans certaines situations quelques légères oscillations au niveau de la vitesse du robot ARP qui n'influent pas vraiment la trajectoire du robot. Et cela revient au problème de l'adaptabilité des méthodes réactives par logique floue. En comparant avec le navigateur flou optimisé, ce dernier répond vraiment au choix du modèle de référence choisie par le concepteur.

3.3 Conclusion

La plateforme de simulation en 3D présentée dans ce chapitre, nous a permis de tester et valider notre approche de navigateur flou réactif. On a montré que notre approche de navigation a permet d'apporter une solution au robot ARP pour l'évitement d'obstacles et la convergence vers un but dans des environnements plus ou moins encombrés. Dans le prochain chapitre, nous présentons la conception et la réalisation (mécanique, électronique, et informatique) du robot ARP ainsi que les expériences pratiques réalisées avec ce robot dans un environnement réel afin de valider notre approche.

Chapitre 4

Sommaire partiel 4:

4

Réalisation du robot mobile ARP et résultats expérimentaux

Nous allons au cours de ce chapitre, présenter d'abord la réalisation du robot mobile ARP. Ensuite nous présentons quelques résultats expérimentaux réalisés avec ce robot.

4.1 Présentation du robot mobile ARP :

Comme nous l'avons déjà indiqué dans l'objectif du travail présenté au début de ce mémoire, le robot mobile réalisé doit être autonome et capable de se localiser en permanence dans son environnement. C'est-à-dire que le robot doit être équipé d'un système de localisation par odométrie et doit aussi avoir un système sensoriel pour la détection d'obstacle. Le robot doit aussi effectuer des missions de navigation, il s'agit de se déplacer d'une position initiale A à une position finale B tout en évitant les obstacles. Nous avons donc choisi d'équiper le robot mobile ARP de trois télémètres infrarouges pour la détection d'obstacles et de deux codeurs optiques pour l'odométrie, nous avons aussi choisi de l'équiper de deux actionneurs (moteurs) et d'un système de communication radio, pour que le robot puisse recevoir les coordonnes du point destinataire afin de bien accomplir sa mission de navigation. Nous avons trois parties principales à travailler sur le robot : la partie mécanique auquel les différents capteurs et actionneurs sont fixés, la partie informatique (codes et algorithmes) c'est la partie intelligente du robot, et la partie électronique qui consiste donc à faire le pont entre l'informatique et la mécanique. Les différents aspects liés à la réalisation du robot mobile ARP sont présentés dans la prochaine section.

4.1.1 L'Aspect mécanique :

Le robot réalisé est de type uni-cycle (type 6 voir chapitre 1 page 8) et la mécanique de sa plateforme mobile est relativement simple, la forme globale du robot est cylindrique d'un diamètre de 25 centimètres et sa locomotion est assurée par deux roues motrices diamétralement opposées. Les roues sont fines avec un espacement connu, ce qui permet d'avoir un contact quasi-ponctuel avec le sol, leur diamètre est de 6 centimètres et sont directement fixées sur les moteurs. La stabilité est assurée par deux roues folles placées à l'avant et à l'arrière du robot. Les moteurs utilisés sont des moteurs réducteurs à courant continu MFA 942 15V de 21.2W et leur facteur de réduction est de 1:100 avec une vitesse nominale de 140 (tr/m).

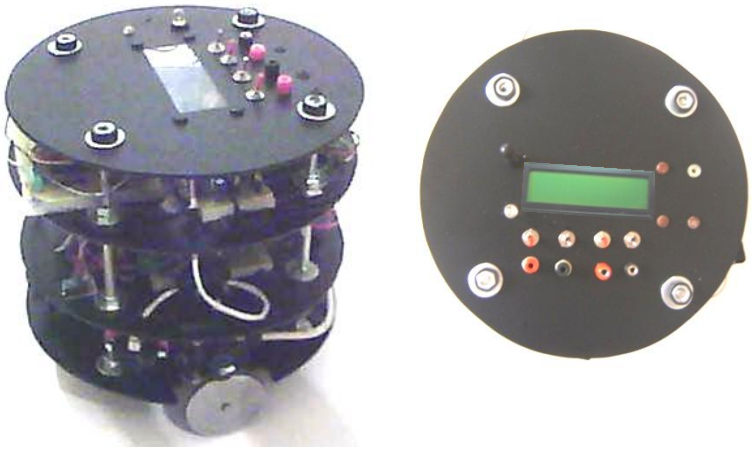


Figure 4.1 : Le robot ARP vue de profil et de haut.

Figure 4.2 : Les moteurs du robot ARP.

4.1.2 L'Aspect électronique :

4.1.2.1 Les batteries :

Le robot étant parfaitement autonome, il fonctionne donc sur batterie. Les différents circuits et actionneurs nécessitent des tensions et des courants différents. Donc, nous avons choisi de prendre deux batteries de 12V/1.3 Ah, la première, alimente la carte de puissance et les moteurs, et la deuxième batterie alimente la partie logique du robot (capteurs, microcontrôleurs,…).

Figure 4.3 : Les batteries du robot ARP.

Pour recharger les batteries [Yann], il convient d'utiliser un chargeur adapté, nous proposons aussi de modifier une alimentation sans interruption d'un PC de bureau qui permet aussi la recharge des batteries 12V. Cependant, un inconvénient apparaît : quand les batteries sont déchargées, il faut les sortir du robot, les charger et les remettre en place. À force, cela devient contraignant. Pour simplifier, nous avons utilisé les interrupteurs **(marche/arrêt)** du robot ARP pour pallier à ces inconvénients. Voir la figure 4.4. La recharge se fait en mode off et sans sortir les batteries du robot.

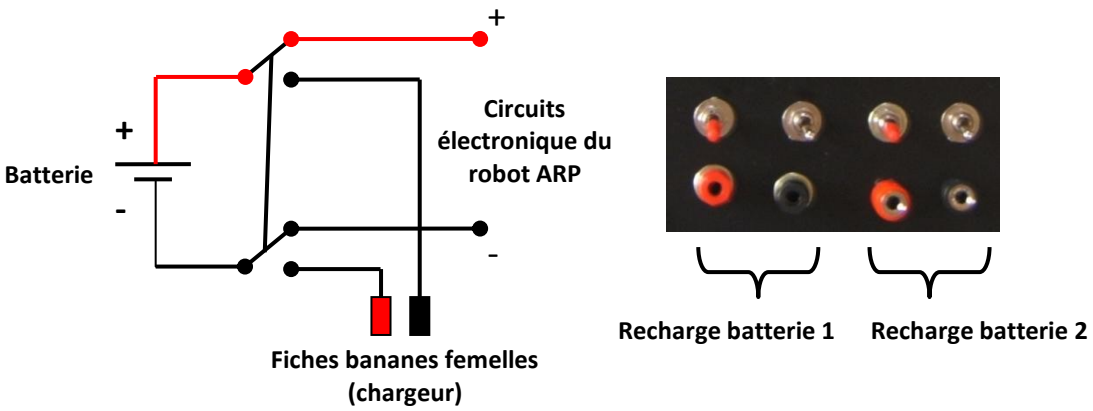


Figure 4.4 : Système de recharge batteries.

4.1.2.2 Les codeurs optiques :

Deux codeurs optiques incrémentaux de résolution (1 impulsion par tour) sont reliés aux roues de chacun des deux moteurs du robot ARP, ce qui nous permet de récupérer une information sur l'incrémentation angulaire des moteurs. On peut alors calculer la vitesse du robot et sa position. Les codeurs permettent de renvoyer des signaux rectangulaires (voir chapitre 1 page 10) dont la fréquence est proportionnelle à la vitesse de rotation du moteur. Grâce à ces codeurs, il est possible de faire exactement le nombre de tours de roue que l'on souhaite.

Les codeurs utilisés ont des signaux sur trois files, un fil (N) pour le comptage d'impulsion par tour, et deux files (A et B) pour connaître le sens de rotation des moteurs. Les files de type (N) des deux codeurs sont reliés au PIC via T0CKI et T1CKI correspondant aux broches 6 et 15 du microcontrôleur 18F452.

Remarque : Le périmètre de chaque roue du robot ARP est de 20 cm, les codeurs fournissent donc une impulsion chaque 20 cm, ce qui est vraiment imprécis pour la localisation du robot et son asservissement. Pour améliorer la précision du système, nous avons choisi d'installer ces codeurs au-dessus des roues (au lieu de les installer sur l'axe des roues), fixés sur un flasque à base de ressort (pour absorber les vibrations). Quand les roues font un tour complet, les codeurs fournissent 20 impulsions c'est-à-dire (1 impulsion par centimètre). Les codeurs optiques du robot ARP sont représentés à la figure 4.5 et leur installation mécanique est représentée à la figure 4.6.

Figure 4.5 : les codeurs optiques du robot ARP.

Figure 4.6 : Installation mécanique des codeurs optiques du robot ARP.

4.1.2.3 Les télémètres infrarouges :

Trois télémètres infrarouges permettent de détecter les obstacles. Ils sont répartis autour du robot ARP sur trois zones (frontale, gauche, et droite). Leur portée est nettement suffisante par rapport à l'utilisation que nous allons en faire. Ils sont très directifs, ce qui peut nous apporter une bonne fiabilité de la mesure. Leur réponse est analogique, donc plus facile à analyser et exploiter par notre microcontrôleur. Les capteurs infrarouges renvoient une tension proportionnelle par rapport à la distance de l'obstacle. Nous utilisons les capteurs Sharp de la série GP2D120 qui sont représentés à la figure 4.7. L'allure de la tension fournie par les capteurs de cette série en fonction de la distance de l'obstacle est représentée à la figure 4.8.

Figure 4.7 : Télémètres infrarouges du robot ARP.

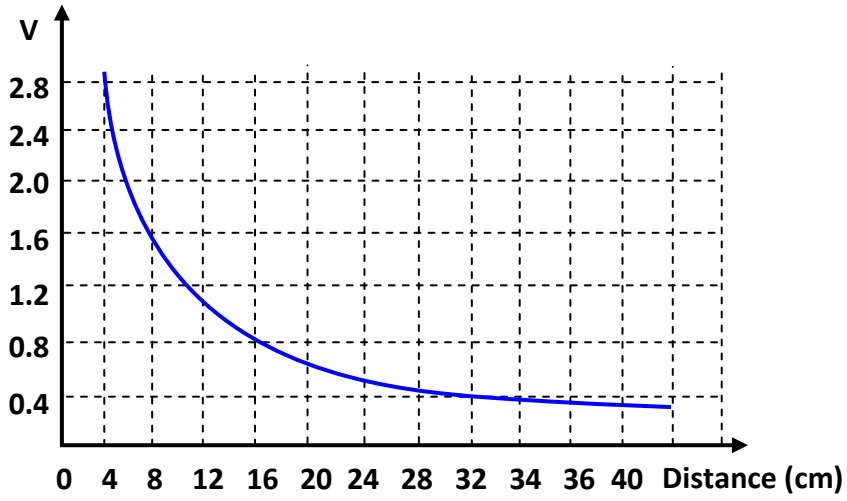


Figure 4.8 : Courbe de tension/distance des télémètres série GP2D120.

4.1.2.4 Le module de transmission sans fil :

Les ondes radio restent un véhicule privilégié pour toutes communications à courte, moyenne ou longue distance, le module choisi est une paire radio de chez AUREL (émetteur TX433, et le récepteur RF 433). Ce module nous permet de mettre en place une liaison série sans fil. Il permet des communications dans la bande de 433.92 Mégahertz dans une zone géographique de 400 mètres. Le débit est souvent entre 300 et 115200 bps.

Remarque : Pour filtrer le bruit, un circuit de codage est raccordé à un montage émetteur qui va pouvoir filtrer la trame de donnée envoyée par l'émetteur. Côté réception, des circuits appropriés pourront facilement reconstituer le signale transmis.

L'émetteur et le récepteur doivent être équipés d'une antenne. Pour cela, on peut utiliser un court morceau de fil de cuivre rigide d'une longueur de 16 centimètres, ou bien une petite antenne.

Figure 4.9 : Module radio Émetteur TX433/ Récepteur RF 433.

4.1.2.5 Cartes électroniques :

Plusieurs approches peuvent être utilisées pour la conception électronique d'un système, robot ou autre. L'approche utilisée pour le robot ARP est de type modulaire : il y a donc plusieurs cartes dans le robot. Ce choix est justifié par le fait qu'il permet un développement plus simple. En effet, chaque carte est indépendante des autres, tout en contribuant à un tout. Il est donc possible de concevoir électroniquement une carte, de la programmer et de la tester, permettant ainsi de finaliser une section complète du robot. Ensuite, cette carte se relie aux autres pour constituer le système. En cas de problème, la localisation et la correction du défaut sont très rapides et les autres parties du robot sont toujours fonctionnelles. Quatre cartes électroniques assurent le bon fonctionnement de notre robot ARP :

a. **La carte principale s'occupant de l'acquisition et du traitement des signaux :** La carte mère est le cerveau principal du robot. Elle contient un PIC18F452 qui agit comme gestionnaire des comportements du robot. Tous les algorithmes de haut niveau du fonctionnement du robot y sont codés. Sur la carte est aussi présent un autre microcontrôleur PIC16F877A dédié à la gestion de l'affichage sur LCD

[Yann]. La communication entre ces deux interfaces se fait par le bus I2C. La carte mère supporte directement les télémètres infrarouges, codeurs optiques, Récepteur RF 433, boutons, et afficheur LCD de 2 lignes de 16 caractères. La carte mère et son afficheur sont représentés respectivement a la figure 4.10 et figure 4.11. La figure 4.12 montre le schéma électrique de cette carte.

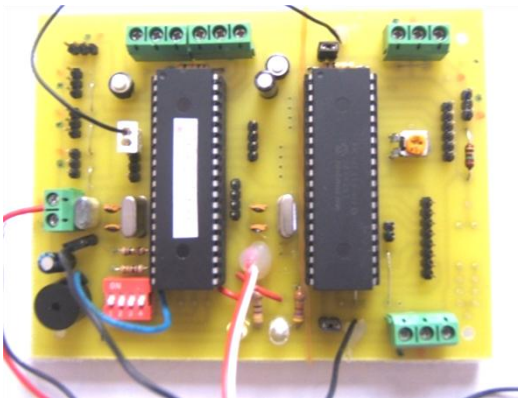


Figure 4.10 : Carte mère du robot ARP.

Figure 4.11 : Afficheur du robot ARP.

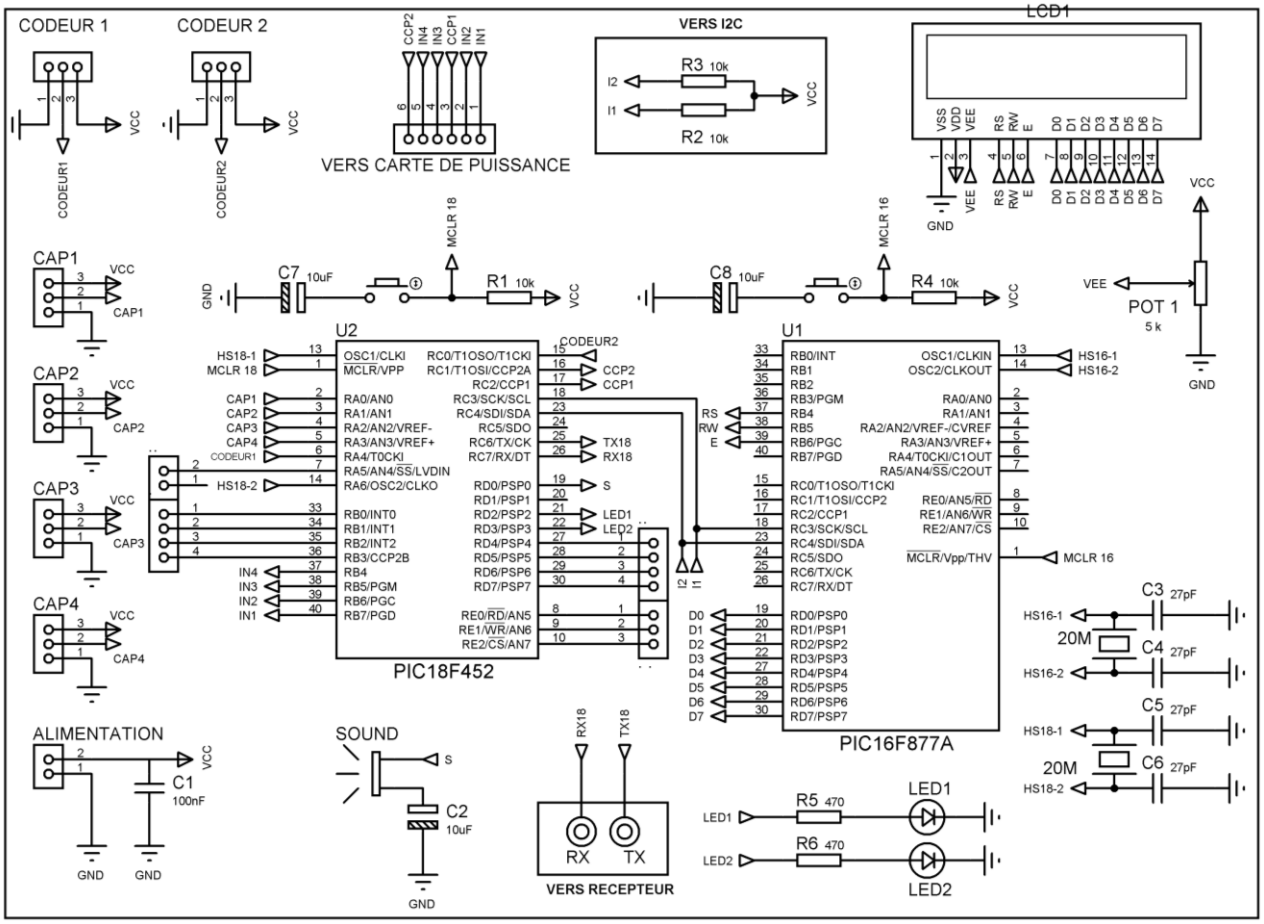


Figure 4.12 : Schéma électrique de la carte mère du robot ARP.

b. La carte de puissance : permet de contrôler la tension à fournir aux moteurs (par une modulation PWM[1]) ainsi que le sens de déplacement selon les ordres de la carte mère. Vue la consommation importante du courant par les deux moteurs, nous avons opté pour une architecture a double circuits de puissance de type L298, chaque circuit fournit au moteur un courant max de 3A. Un ventilateur est associé à la carte de puissance pour le refroidissement des circuits L298. La carte de puissance et son ventilateur sont représentés à la figure 4.13. La figure 4.14 montre le schéma électrique de la carte de puissance.

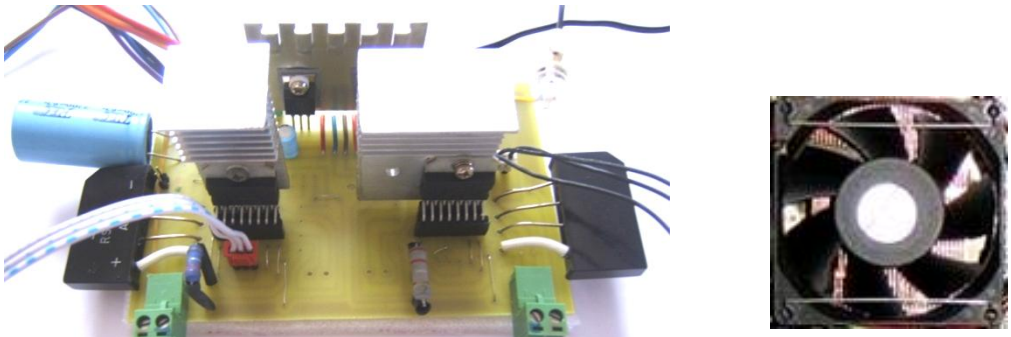


Figure 4.13 : Carte de puissance du robot ARP et son ventilateur.

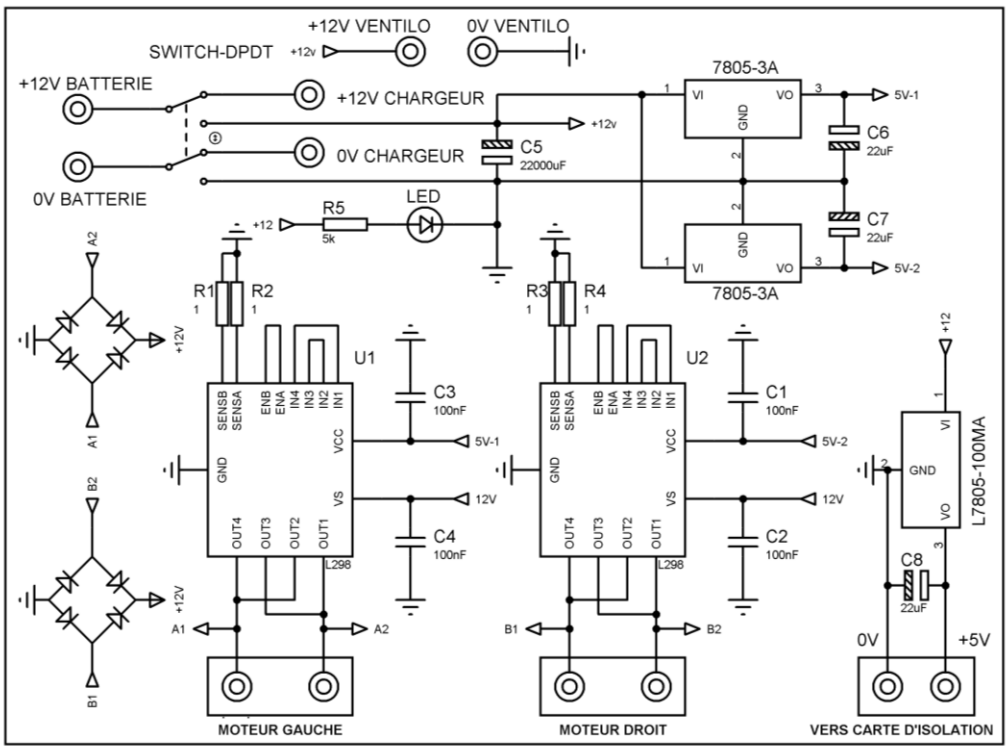


Figure 4.14 : Schéma électrique de la carte de puissance.

[1] PWM : Pulse Width Modulation (modulation de largeur d'impulsion).

c. **La carte d'isolation galvanique :** Afin d'éviter les problèmes de parasitage des circuits de commande par les moteurs de puissance, nous avons choisi de séparer deux grands ensembles, la partie logique et la partie puissance. Toute communication effectuée entre les deux parties se fait par le biais d'optocoupleurs [Yann], évitant ainsi tout lien électrique. La carte d'isolation galvanique est représentée à la figure 4.15. La figure 4.16 montre le schéma électrique de cette carte.

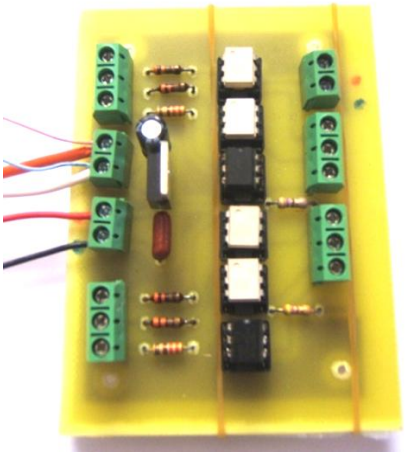


Figure 4.15 : Carte d'isolation galvanique du robot ARP.

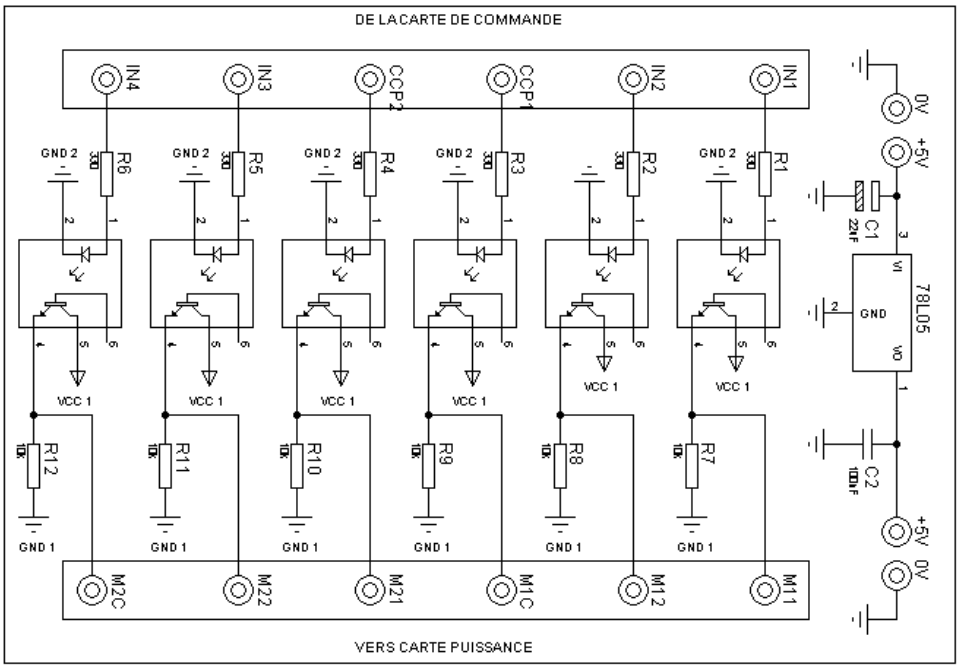


Figure 4.16 : Schéma électrique de la carte d'isolation galvanique.

d. Cartes émetteur récepteur 433 Mhz : Afin de pouvoir aisément envoyer des informations a notre robot depuis un PC en utilisant la liaison RS 232 sans fil, nous avons réalisé une carte module émetteur à base de MAX 232 et un circuit codeur, et une carte module récepteur avec un circuit décodeur. La carte module émetteur et son schéma électrique sont représentés respectivement a la figure 4.17 et figure 4.18. La carte module récepteur et son schéma électrique sont représentés respectivement a la figure 4.19 et figure 4.20.

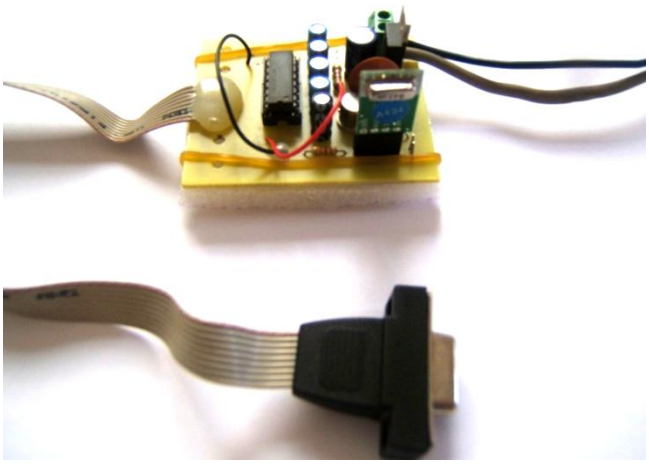


Figure 4.17 : Carte module émetteur TX433.

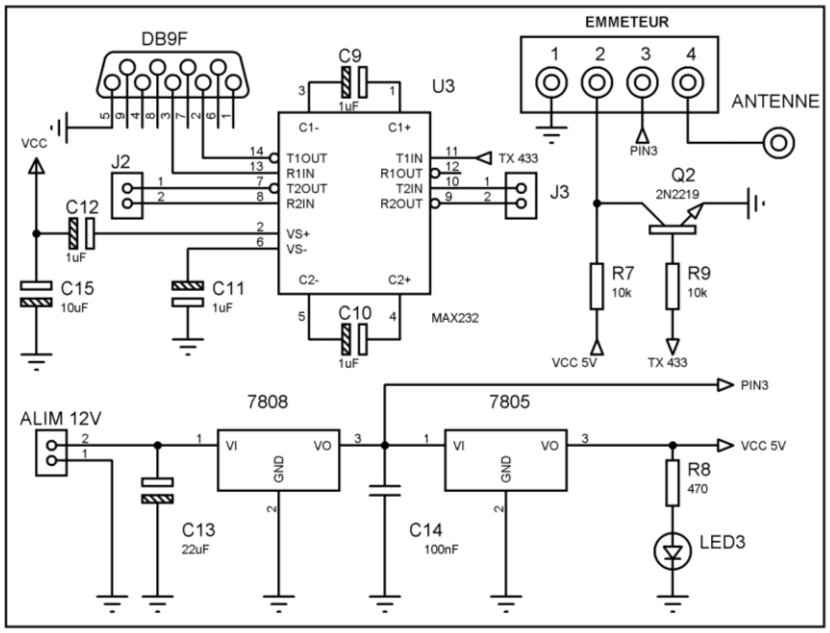


Figure 4.18 : Schéma électrique de la carte module émetteur TX433.

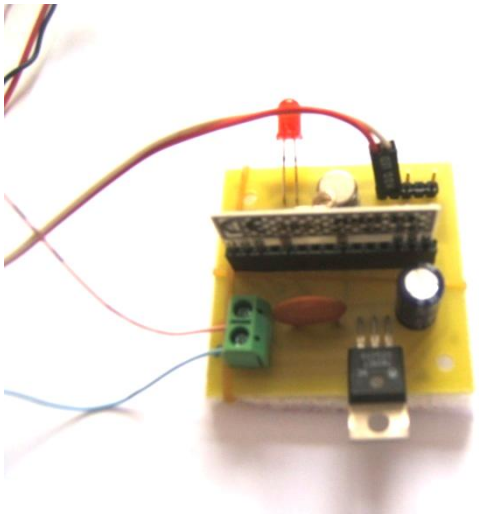


Figure 4.19 : Carte module récepteur RF 433.

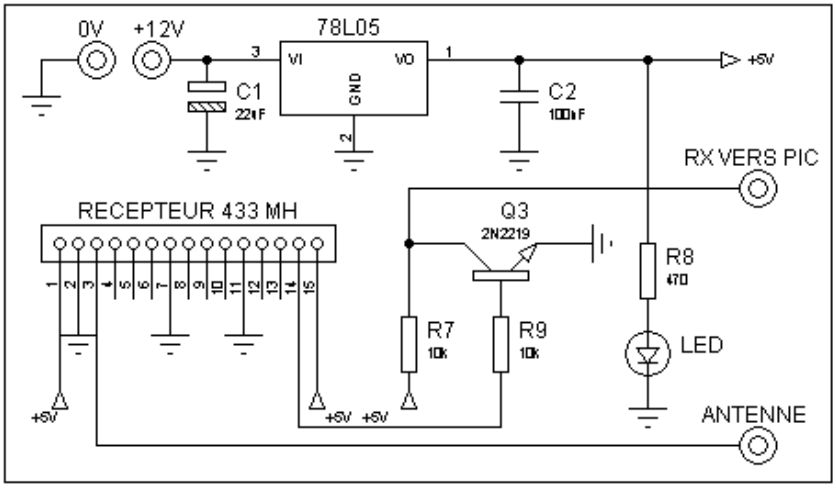


Figure 4.20 : Schéma électrique de la carte module récepteur RF 433.

La figure 4.21 représente la connexion des différentes cartes électroniques aux divers capteurs et actionneurs du robot ARP.

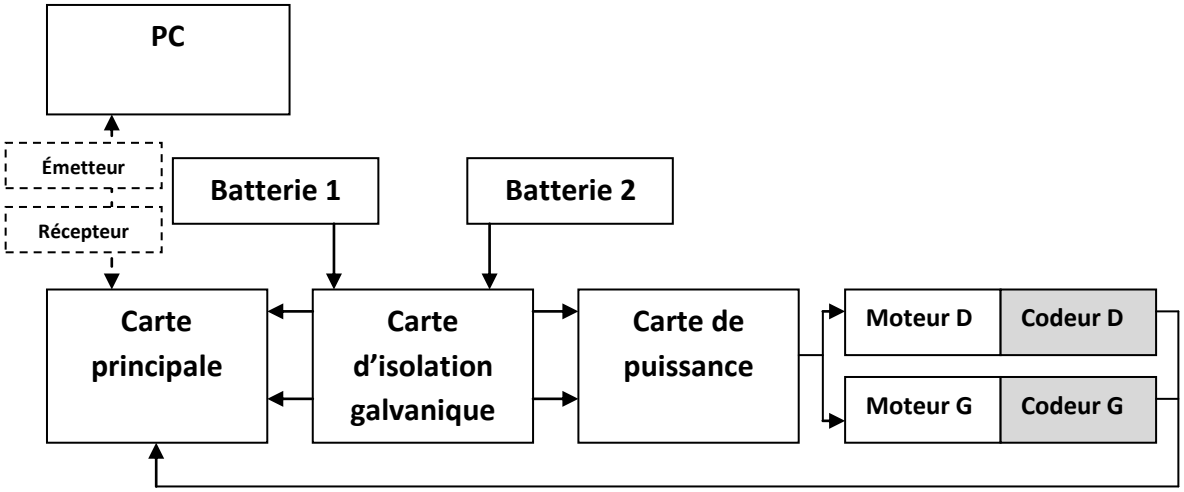


Figure 4.21 : Schéma bloc du fonctionnement générale du robot ARP.

Discussion 1 : Le robot reçoit des informations sur le point destinataire, ces informations sont envoyées depuis un PC via le module radio sous forme numérique exemple : (X, Y) = (120, 140). **L'envoi de données se fait une fois et au début de la navigation**. Le robot enregistre donc ces informations et calcule son chemin, cette partie est gérée par la carte principale et son alimentation est assurée par la batterie 1. Le robot ensuite prend une décision et envoie des commandes (via la carte d'isolation galvanique) à la carte de puissance qui à son tour commande les deux moteurs du robot ARP. L'alimentation de la carte de puissance et les moteurs est assurée par la batterie 2. Le retour des informations sur la vitesse et la position du robot ARP est assuré par les deux codeurs (D pour droit et G pour gauche) et se fait au niveau de la carte principale.

4.1.3 Partie informatique distante :

Pour assurer l'envoi des informations au robot ARP concernant le point destinataire, il est donc nécessaire de réaliser une interface informatique adapté à cette tâche. Pour ce faire, nous avons choisi de développer notre application en utilisant le logiciel **Borland Builder C++** qui est un IDE (Environnement de développement intégré) et qui regroupe tout un ensemble d'outils permettant d'effectuer un maximum de tâches de développement au sein du même environnement de travail. Il a aussi la particularité de générer des applications portable c'est-à-dire des codes standards pouvant s'exécuter depuis n'importe quelle machine et sans avoir déjà installer **Builder C++**, contrairement a **Matlab** qui permet aussi la conception de ce genre d'application mais cette application doit toujours s'exécuter d'une machine où le **Matlab** est déjà installé.

Nous avons donc réalisé une application IHM (Interface Homme Machine), un logiciel faisant une interface entre l'utilisateur et le robot. Cette IHM permet d'envoyer les coordonnés de la position à atteindre par le robot ARP par liaison série [H. Zerfa] sous forme de trames numérique (la trame X, la trame Y), elle permet aussi le retour visuel de la caméra installée dans l'environnement [H. Zerfa]. L'IHM dispose, en outre, de deux boutons réservé a la simulation 3D présenté au troisième chapitre (la simulation est faite en langage python, nous avons donc réalisé un programme en **Builder C++** qui permet de faire le lien entre ces deux langages différent). La figure 4.22 représente l'IHM réalisée.

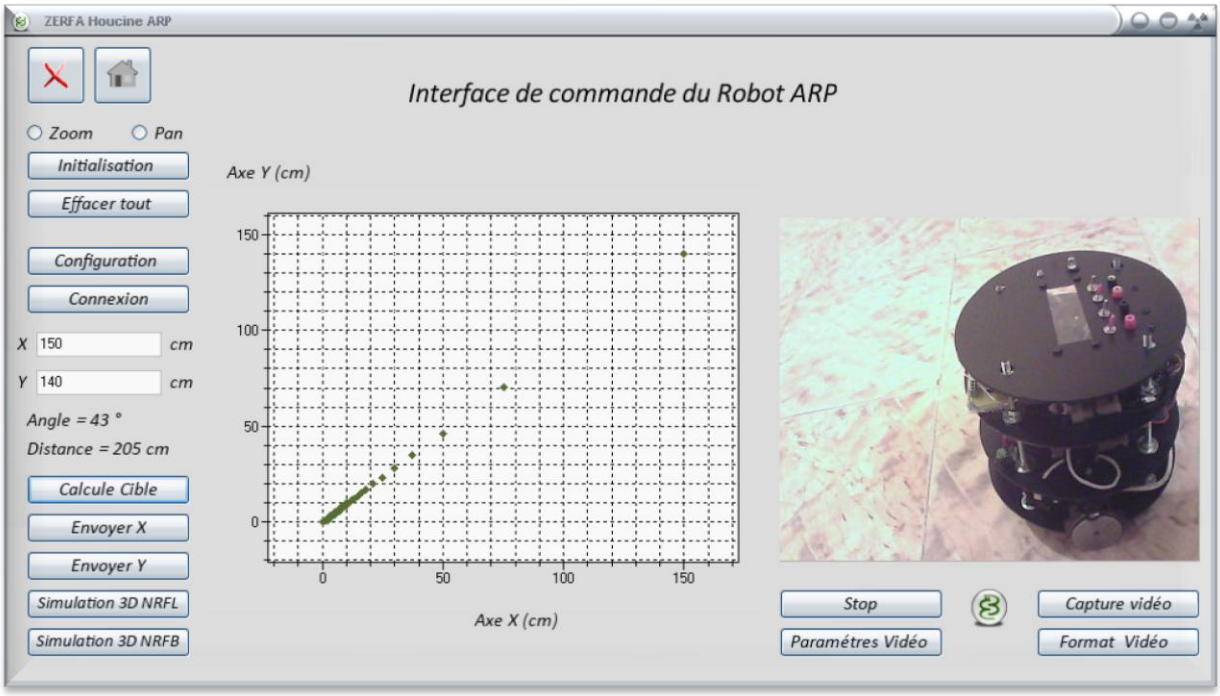


Figure 4.22 : IHM du robot ARP.

4.1.4 Partie informatique embarquée :

Afin d'interfacer le matériel et de faire fonctionner l'ensemble des cartes comme un tout fonctionnel, il est donc nécessaire de programmer la carte principale du robot ARP. Nous avons donc choisi de programmer en langage C en utilisant le logiciel **MikroC** [Mikro 1] [Mikro 2] de chez *MikroElektronika*, qui est un compilateur puissant adapté à tous les microcontrôleurs de type *12F*, *16F*, *18F*.

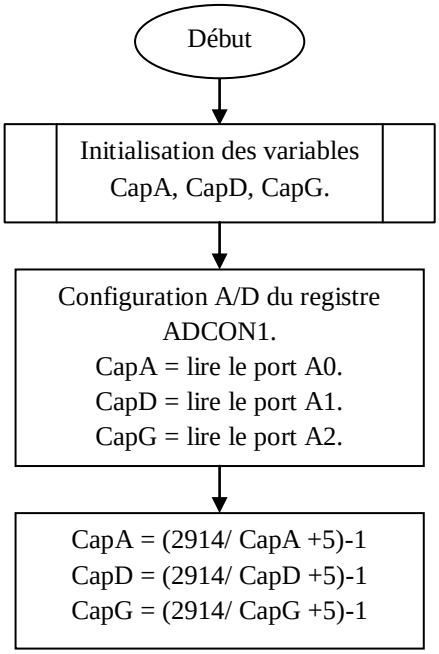
Remarque : La programmation des algorithmes du robot ARP exige une version commerciale et complète du logiciel **MikroC**, il existe une version gratuite mais limitée et ne répond pas à ce niveau d'algorithmes.

Nous avons donc réalisé deux type de programme : un **programme de bas niveau** qui gère l'acquisition et la conversion analogique numérique des capteurs infrarouges [Mikro 2], l'acquisition des données codeurs optiques, la gestion de l'afficheur LCD, la gestion de la liaison série et du Bus I2C et d'autres configurations des microcontrôleurs. Des sous programme sont associés a ce programme de bas niveau, il s'agit de l'odométrie et de la régulation en position et en vitesse du robot ARP. Et **un programme de haut niveau** de navigation réactive floue, il s'agit du navigateur flou de Sugeno [Tak Sug] [M. Sugeno 2] déjà optimisé (ses paramètres de vitesse ont été déjà identifiés) présenté au deuxième chapitre pages 41-57. Dans la suite de cette section nous présentons quelques algorithmes nécessaires à la configuration de la carte principale.

a. Acquisition et calibrage des télémètres infrarouges :

L'acquisition des trois télémètres infrarouges se fait au niveau des ports analogiques (A0, A1, A2) du microcontrôleur 18F452. Le registre ADCON1 [Mikro 2] est configuré pour une conversion analogique - numérique. Pour une bonne mesure de distance, un étalonnage des télémètres est nécessaire. Nous utilisons une formule spécifique à la série GP2D120 [Mikro 2] obtenue par expérience pratique par rapport a la courbe de la figure 4.8 page 75 :
Distance = (2914/AD+5)-1. Où **AD** est le résultat de la conversion analogique – numérique. Nous pouvant alors obtenir une mesure approximative de la distance robot – obstacle.

Organigramme (3) de l'acquisition des trois télémètres infrarouges du robot ARP sous MikroC :

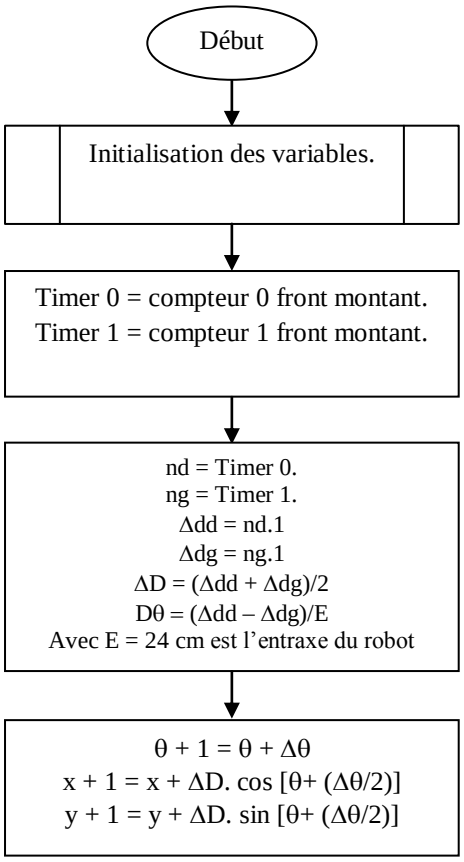


Remarque : La sortie de **l'organigramme (3)** constitue l'entrée de **l'algorithme (3)** présenté au deuxième chapitre page 57.

b. L'odométrie du robot ARP :

Les Timers 0 et 1 du microcontrôleur 18F452 sont configuré pour lire les deux codeurs optiques du robot ARP. L'acquisition est numérique (pas de conversion). Le modèle mathématique de l'odométrie du robot ARP est le même que celui présenté déjà au premier chapitre page 11.

Organigramme (4) de la configuration odométrique du robot ARP sous MikroC :



c. L'asservissement en position et en vitesse du robot ARP :

Pour pouvoir contrôler les moteurs du robot ARP, il faut analyser leur comportements statique et dynamique, c'est-à-dire connaitre leur fonction de transfert. A l'aide des codeurs incrémentaux nous avons identifié les deux moteurs du robot ARP, leurs fonctions de transfert est comme celle d'un système du premier ordre caractérisé par une constante de temps τ : **$H(p)=X(p)/Y(p)= G_S / (\tau p+1)$.** Avec G_S : gain de la tachymétrie.

La courbe de la figure 4.23 représente la vitesse de rotation des deux moteurs du robot ARP en fonction de leur tension d'alimentation.

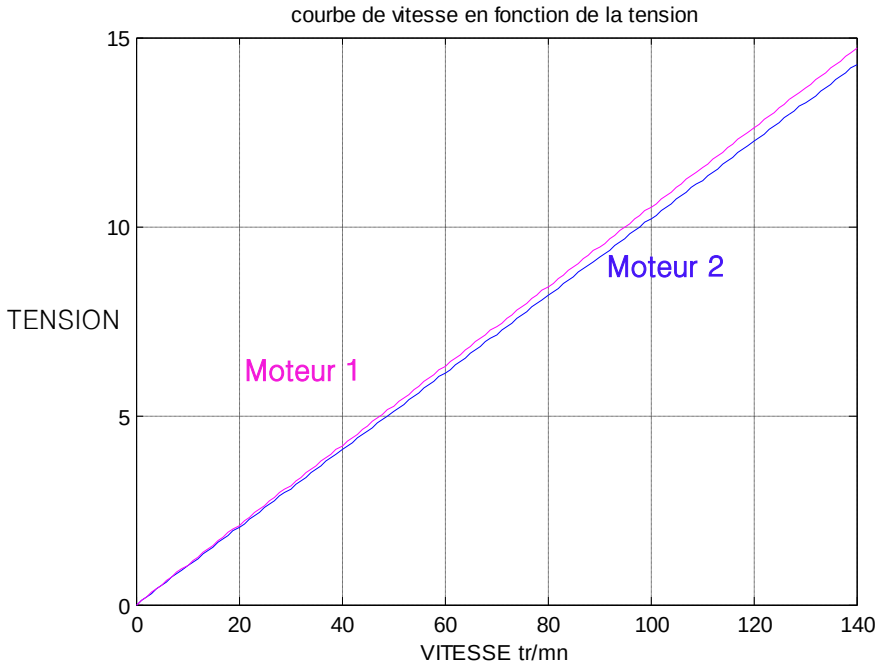


Figure 4.23 : Caractéristique de V=f(ω) des moteurs 1 et 2.

La courbe de la figure 4.24 représente la réponse indicielle (réponse a un échelon) des deux moteurs du robot ARP.

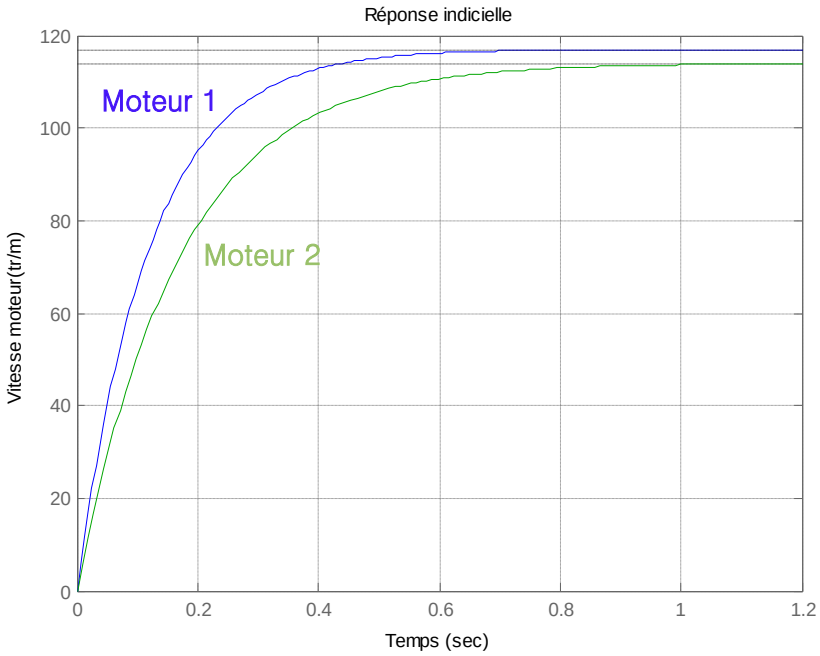


Figure 4.24 : Réponses des deux moteurs du robot ARP.

D’après la figure 4.24 on enregistre la Réponse du moteur 1 :

$$H1(p) = \frac{X1(p)}{Y1(p)} = \frac{0.10}{0.12p+1}$$

Et la Réponse du moteur 2 :

$$H2(p) = \frac{X2(p)}{Y2(p)} = \frac{0.10}{0.17p+1}$$

D’après les figures 4.23 et 4.24 On remarque bien qu’il existe une différence de vitesses entre ces deux moteurs. Ce qui aura pour effet de faire dévier le robot au lieu d’avancer droit (voir figure 4.25), ce qui entraînera de graves erreurs de navigation.

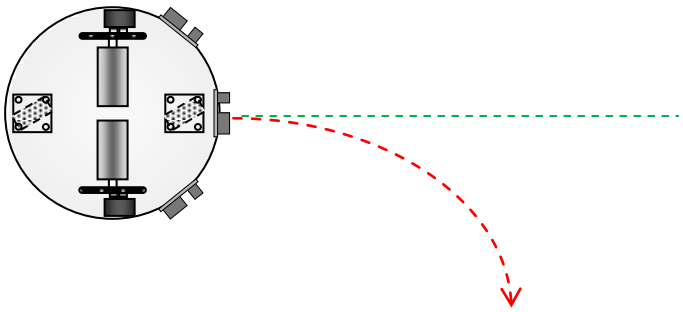


Figure 4.25 : Déviation de la trajectoire du robot.

Le besoin d’asservissement est donc double : d’une part, on souhaite pouvoir asservir la position du robot, c’est-à-dire savoir à tout instant où se situe le robot et pouvoir se déplacer vers des coordonnées précises. D’autre part, de nombreux éléments (légères différences de moteurs, parasites, mauvais équilibrage du robot, ...) font que deux moteurs supposés identiques alimentés de la même manière ne réagiront pas strictement de la même manière sans asservissement.

Pour pouvoir modéliser l’asservissement du robot ARP, il nécessaire de connaître sa position et sa vitesse (selon une composante de translation et une composante de rotation) en fonction de l’historique d’évolution des vitesses respectives de ses moteurs. On peut alors définir les composantes instantanées de vitesse de translation (**V**) et de rotation (**ω**) en fonction des vitesses de translation respectives ($\mathbf{V}_{\text{droite}}$) et ($\mathbf{V}_{\text{gauche}}$) des roues droite et gauche (voir figure 4.26). Avec **E** la distance entre les deux roues. On a alors :

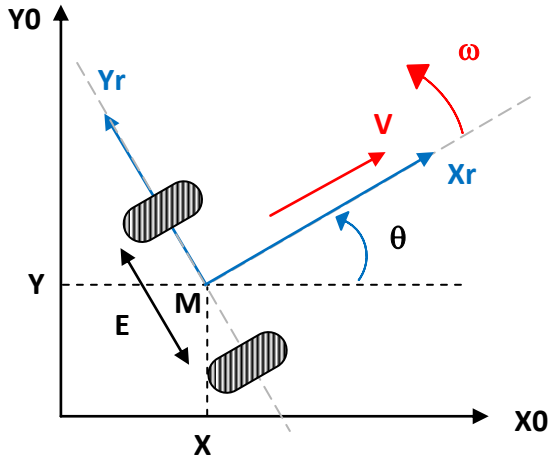


Figure 4.26 : Cinématique du robot ARP.

$$V = (V_{droite} + V_{gauche})/2. \qquad (1)$$
$$\omega = (V_{droite} - V_{gauche})/E. \qquad (2)$$

Cette modélisation suppose une symétrie parfaite des roues et un contact ponctuel et parfait entre les roues et le sol (pas de glissement, environnement plat, roues suffisamment fines pour connaître avec exactitude la position du centre de rotation,...). Par combinaison des deux équations **(1)** et **(2)** nous obtenons:

$$V_{droite} = V + \omega .E/2 \qquad (3)$$
$$V_{gauche} = V - \omega .E/2 \qquad (4)$$

V_{droite} et V_{gauche} sont les grandeurs asservies que les moteurs doivent prendre, la figure suivante représente l'asservissement bimoteur du robot ARP :

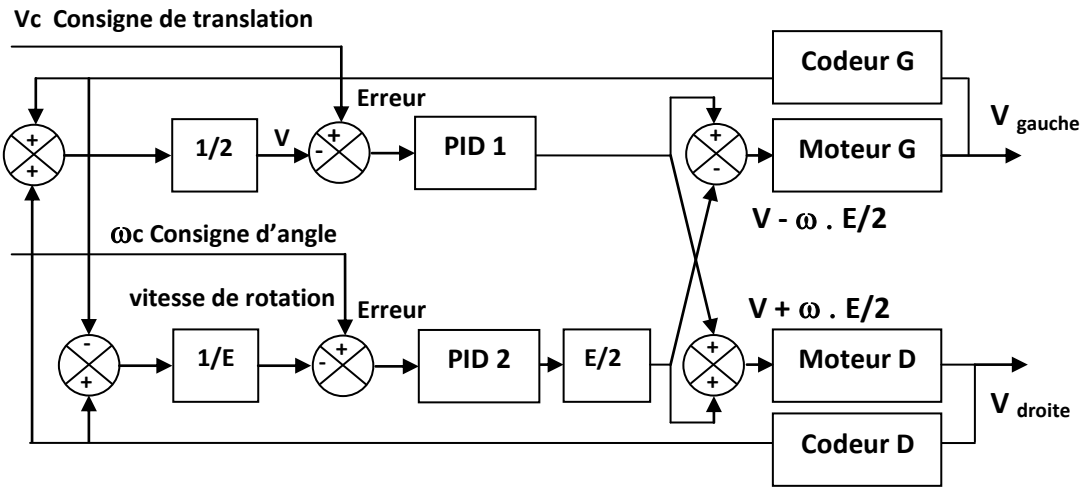


Figure 4.27 : Schéma fonctionnel de l'asservissement bimoteur du robot ARP.

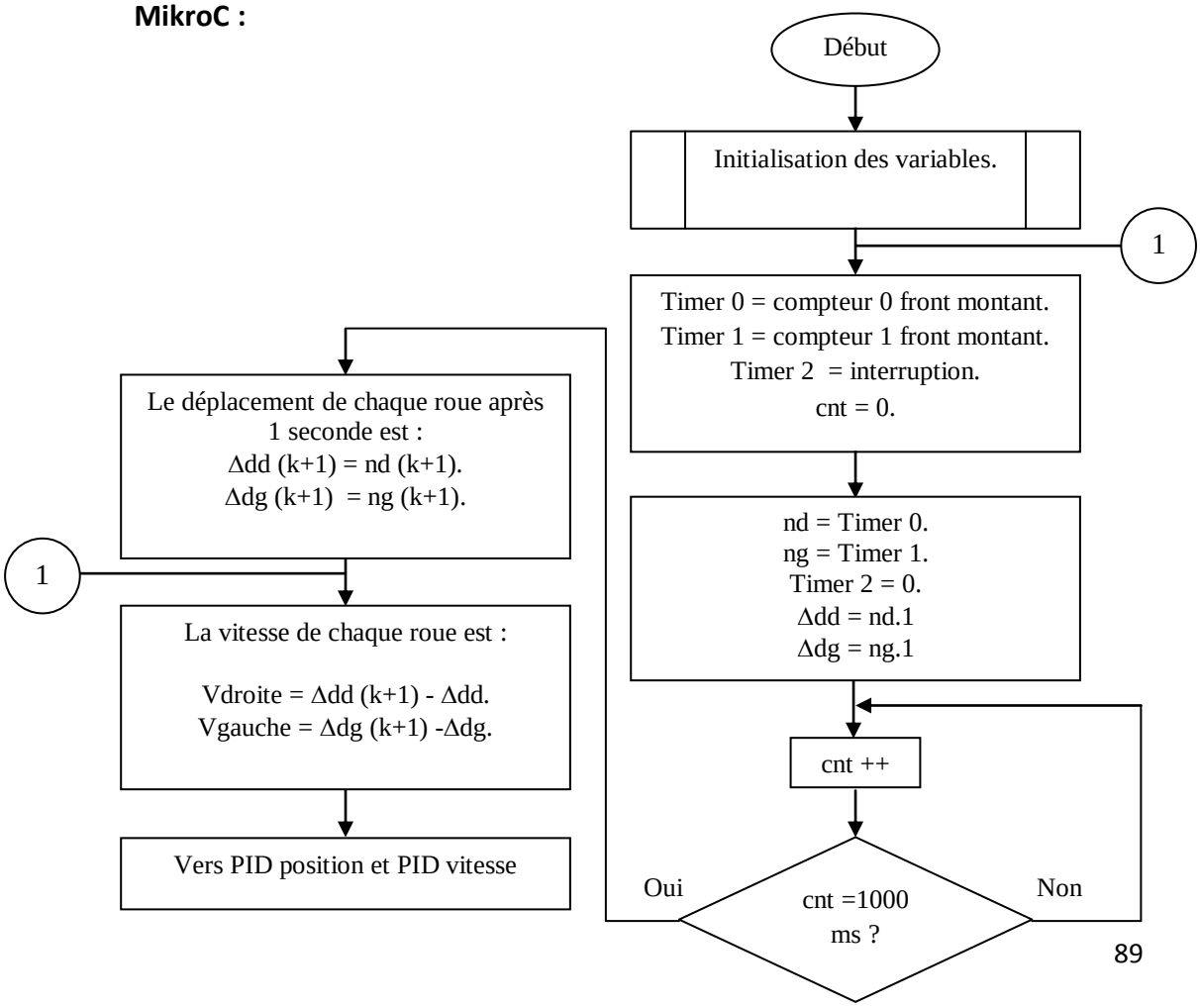
Le mouvement de rotation de l'ensemble est obtenu en appliquant une vitesse de rotation différente à chacune des deux roues motrices, cette architecture d'asservissement bimoteur assure une très bonne coordination des mouvements des deux roues en rotation ou en mouvement rectiligne et obtenir ainsi les mouvements désirées.

Après avoir présenté l'architecture de l'asservissement bimoteur du robot ARP, on passe maintenant à la partie programmation :

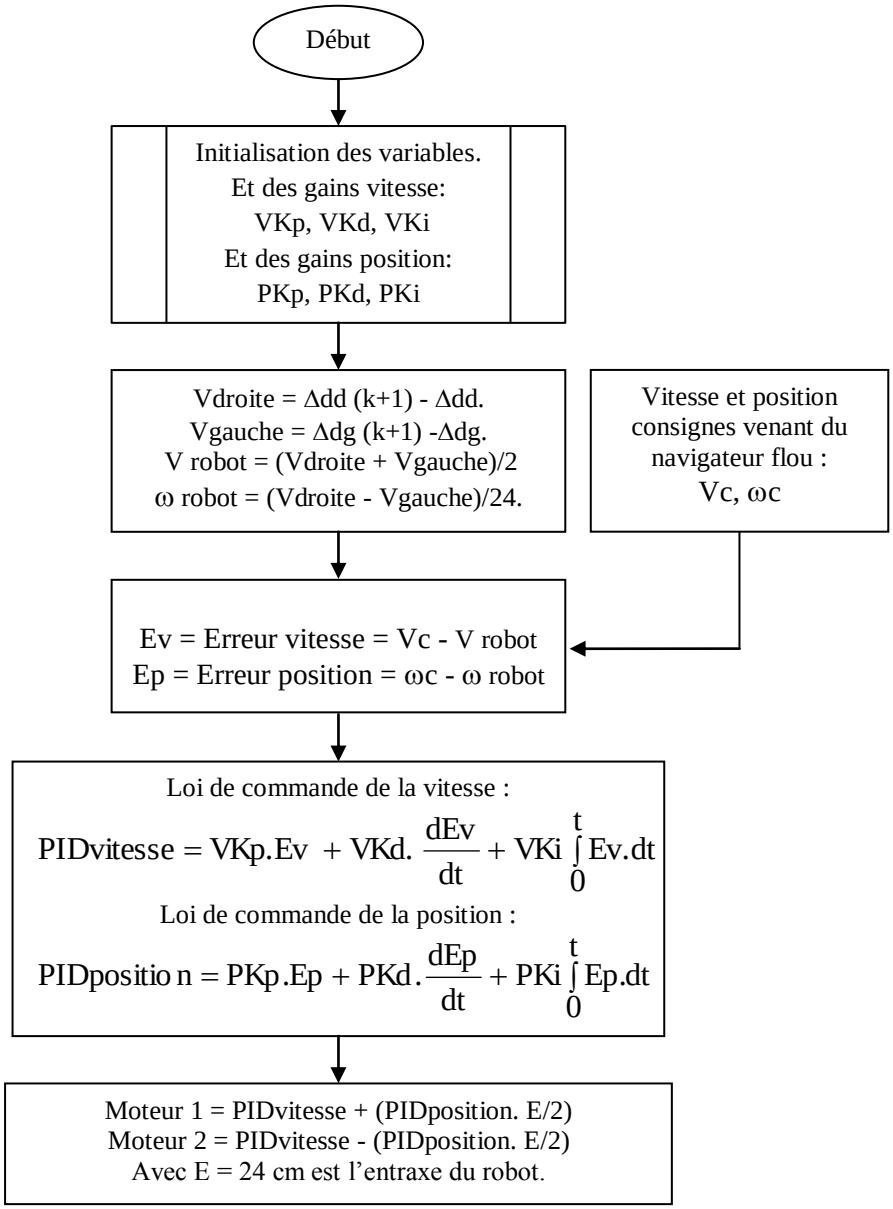
La loi physique générale de la vitesse est **V = Déplacement / temps**. Du modèle odométrique présenté précédemment (organigramme 4), on peut alors récupérer le déplacement de chaque roue du robot ARP grâce aux Timers 0 et 1 configurés pour l'acquisition des deux codeurs. Un troisième Timer est utilisé cette fois ci pour le calcul de temps. Il s'agit du Timer 2 configuré pour déclencher une interruption interne du microcontrôleur 18F452 chaque 1 seconde. On peut alors calculer la vitesse de chaque roue par la mesure de la différence de déplacement effectué chaque 1 seconde. Il s'agit tout simplement de la dérivée du déplacement. La relation de la vitesse devient alors :

V = [Déplacement à l'instant (t +1) - Déplacement à l'instant (t)].

Organigramme (5) : mesure de la vitesse de chaque roue du robot ARP sous MikroC :



Organigramme (6) de l'asservissement en vitesse et en position du robot ARP sous MikroC :



Remarque : Les consignes vitesse et position sont générées par le navigateur flou de Sugeno [Tak Sug] [M. Sugeno 2] présenté au deuxième chapitre pages 41-57. Ces consignes seront comparées aux mesures réelles, pour pouvoir ensuite calculer les lois de commandes PIDvitesse et PIDposition.

4.1.4.1 Mise en œuvre des algorithmes à bord du robot ARP :

La mise en œuvre des algorithmes à bord du robot n'est pas immédiate. Nous avons développé une architecture de contrôle comportementale [Brooks 1] [Brooks 2] [Yann] composée des différents modules présentés précédemment. Ces comportements sont arbitrés par un module d'arbitration (priorité). Il s'agit de lister toutes les tâches que le robot ARP aura à accomplir, puis les classer de la moins prioritaire à la plus importante. La figure 4.28 représente l'architecture de contrôle du robot ARP.

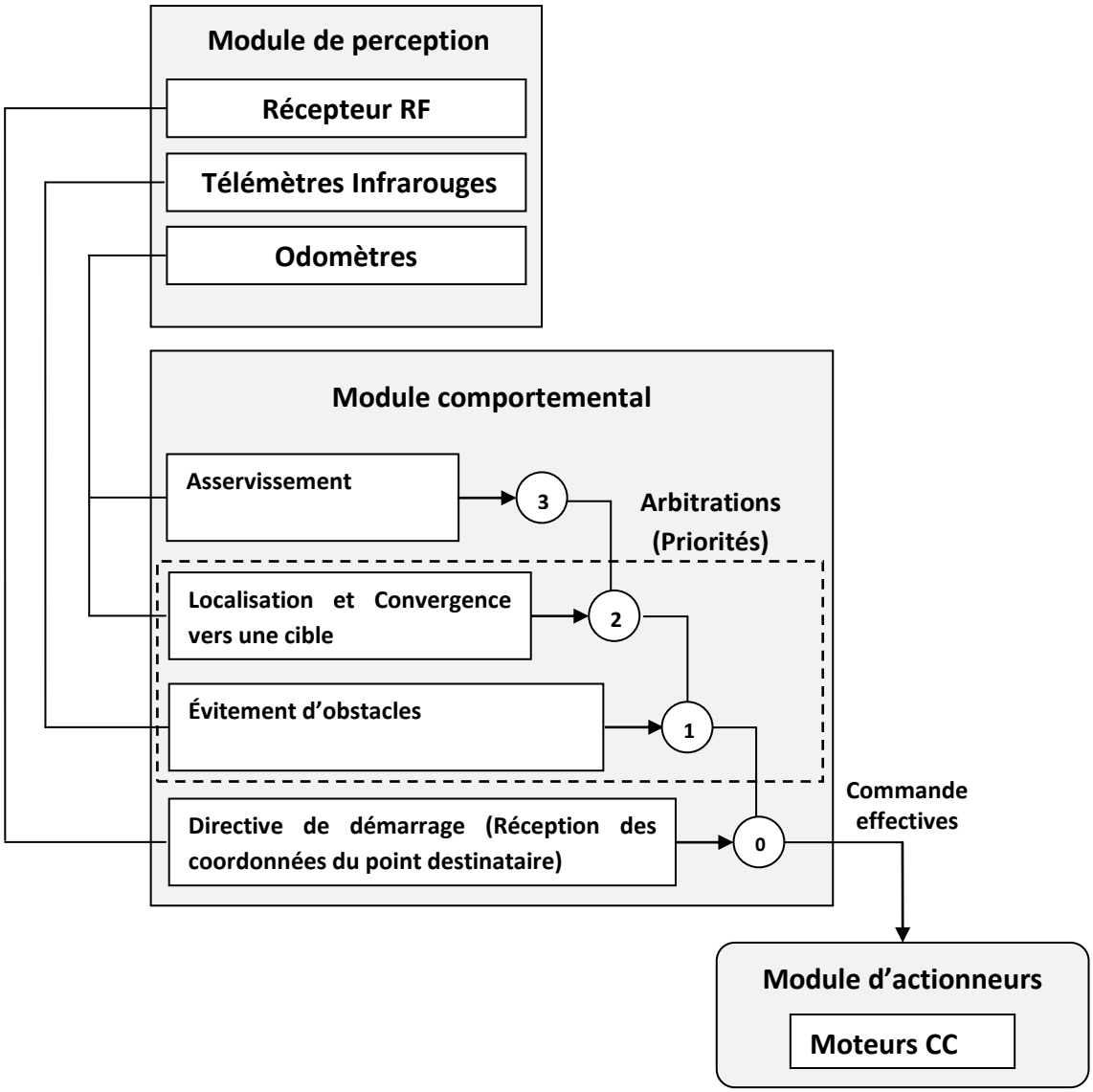


Figure 4.28 : Architecture de contrôle du robot ARP.

Discussion 2 : L'architecture retenue est comportementale (réactive), constituée de trois modules principaux : Le module de perception qui permet d'acquérir des informations de l'environnement. Un module d'action, qui exécute toute sorte de commandes. Et un module comportemental composé du navigateur flou [(convergence vers une cible) et (évitement d'obstacles)] et des comportements de localisation et

asservissement. Au démarrage, le robot doit d'abord recevoir les coordonnées concernant le point destinataire ensuite vérifier s'il n'y a pas d'obstacles, calculer un chemin et enfin vérifier si le point d'arrivée est atteint.

Discussion 3 : Après avoir présenté les différents algorithmes ainsi que l'architecture de contrôle du robot ARP. Nous pouvons désormais, programmer la carte principale du robot. Nous rappelons que la programmation est en langage C sous MikroC. Après la compilation des programmes, un fichier d'extension (.hex) est généré. Ce dernier est implémenté sur la carte principale via un programmateur électronique. Nous proposons un schéma électronique d'un programmateur que nous avons réalisé dédié à la programmation de toutes les catégories de PIC voir figure 4.29.

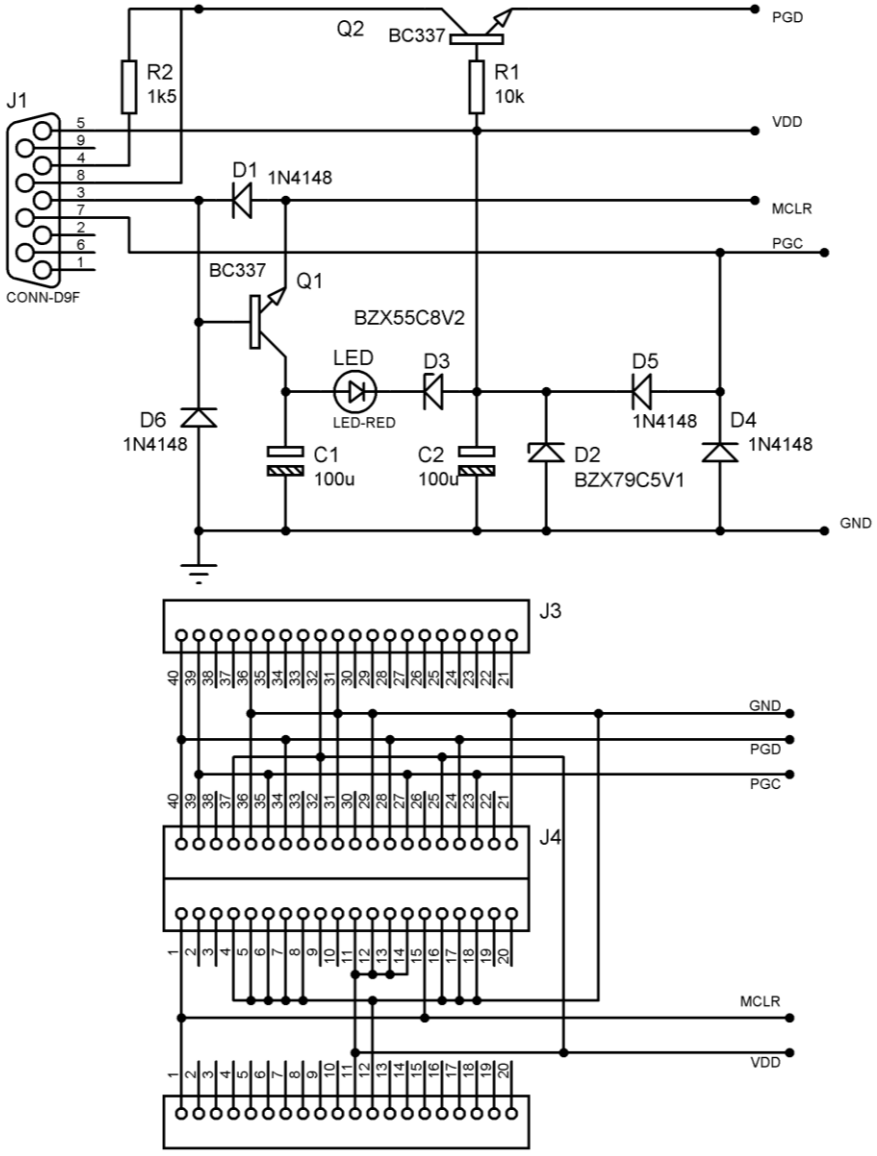


Figure 4.29 : Programmateur de PIC.

4.2 Résultats expérimentaux :

Dans cette section, nous présentons quelques tests pratiques de navigation floue du robot ARP. Nous rappelons que le robot est complètement autonome et il est doté d'un navigateur flou qui permet de gérer sa vitesse et son angle de braquage (ce navigateur flou est déjà optimisé et les paramètres de vitesse ont été déjà identifiés. Voir chapitre 2 pages 54-57). Le robot doit exécuter des missions de navigation qui consiste d'aller d'une position initiale A à une position finale B. Et il est soumit a une contrainte d'environnement concernant la non-collision aux obstacles encombrant son environnement. Et une contrainte de non holonomie. Nous rappelons que le robot ignore complètement la structure de son environnement et ne fait aucune planification.

4.2.1 Expérimentation 1 :

L'objectif visé dans cette première expérimentation est de juger l'efficacité du comportement d'évitement d'obstacles de notre robot. Pour ce faire, nous allons tester **l'organigramme (2)** présenté au troisième chapitre (page 64). Nous rappelons que cet organigramme est basé sur les huit règles floues de notre navigateur de Sugeno [Tak Sug] [M. Sugeno 2] présenté au deuxième chapitre (page 38) sans changer la sortie angle (zéro) de la huitième règle. Le robot fait donc une exploration du terrain sans se diriger vers un point précis.

Au début de cette expérience, le robot est placé aléatoirement dans un sol plat peu encombré et la communication radio (entre le PC et le robot) est désactivée (pas d'envoi de données dans cette expérience). Les résultats de cette expérimentation sont illustrés dans les figures suivantes :

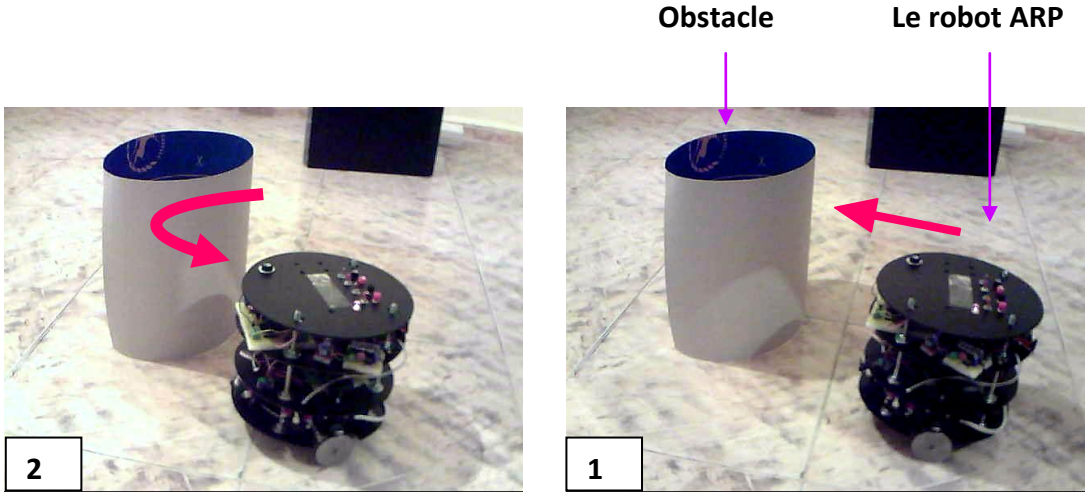


Figure 4.30 : Étapes 1 et 2 du comportement d'évitement d'obstacles.

Discussion 4: Dans l'étape 1 de la figure 4.30, l'obstacle est suffisamment loin du robot. Ce dernier se dirige avec un angle zéro et une vitesse maximale, le navigateur flou du robot est donc en train d'exécuter la huitième règle floue. Dans l'étape 2, le

robot détecte l'obstacle qui se trouve dans son chemin. Le navigateur flou du robot exécute une des sept premières règles floues et diminue la vitesse de translation et change l'angle de braquage du robot ARP d'une manière progressive. Le robot est donc en train d'éviter l'obstacle.

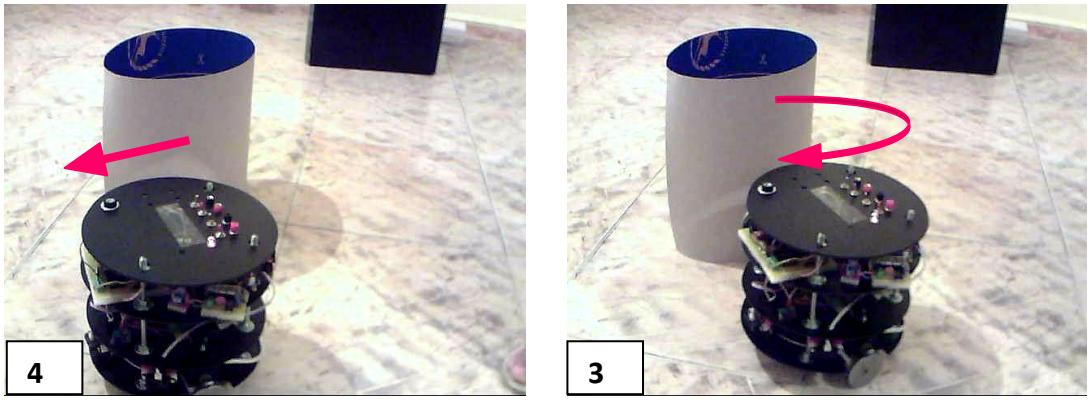


Figure 4.31 : Étapes 3 et 4 du comportement d'évitement d'obstacles.

Discussion 5: Dans l'étape 3 de la figure 4.31, le robot s'éloigne petit à petit de l'obstacle. Le navigateur flou du robot augmente sa vitesse de translation progressivement et l'angle de braquage tend vers le zéro. Dans l'étape 4, la voie du robot est libre (pas d'obstacles). Le navigateur flou génère une vitesse maximale et un angle de braque zéro et le robot ne fait aucune correction de trajectoire (pas de convergence vers un point cible).

Nous avons testé le robot dans des situations différentes (obstacle a droite, a gauche, a droite et a gauche, en face, en face et a droite, en face et a gauche, en face et a gauche et adroite). Les résultats obtenus sont satisfaisants. Le robot parvient toujours à éviter les obstacles. Cette expérimentation valide le comportement flou d'évitement d'obstacles du robot ARP dans un environnement peu encombré.

4.2.2 Expérimentation 2 :

Cette fois-ci, nous testons en plus du comportement d'évitement d'obstacles le comportement de convergence vers un but. Le robot doit naviguer d'un point A à un point B tout en évitant les obstacles. Le navigateur flou de Sugeno implémenté est déjà optimisé et la politique de navigation est celle de **l'organigramme (1)** présenté au deuxième chapitre (page 41). L'angle zéro de la huitième règle floue devient alors θ (voir chapitre 2 page 39).

Test 1 :

Dans ce premier test, l'environnement est peu encombré et la surface du sol est d'une dizaine de m^2. Le robot initialement est placé dans le sens des **X** croissant (il fait un angle **0** avec l'axe **OX**) voir étape 1 de la figure 4.32, et sa position actuelle est **(0,0).** La communication radio est activée. Au démarrage, le robot ne fait aucun mouvement et attend à ce qu'on lui envoie les coordonnées du point destinataire. En utilisant un PC de bureau, nous branchons la carte module émetteur radio présenté précédemment (page 80). On lance ensuite notre application IHM du robot ARP (présenté à la page 83). On envoie alors les coordonnées du point destinataire. Exemple : point destinataire **(X=120, Y=130) cm**. Le robot ARP doit donc se rendre en **(120,130)** en partant de **(0,0)**. L'envoi de données se fait une fois et au début de la navigation. Dés que le robot reçoit les coordonnées du point destinataire, il démarre immédiatement et ne fait aucune planification de l'environnement.

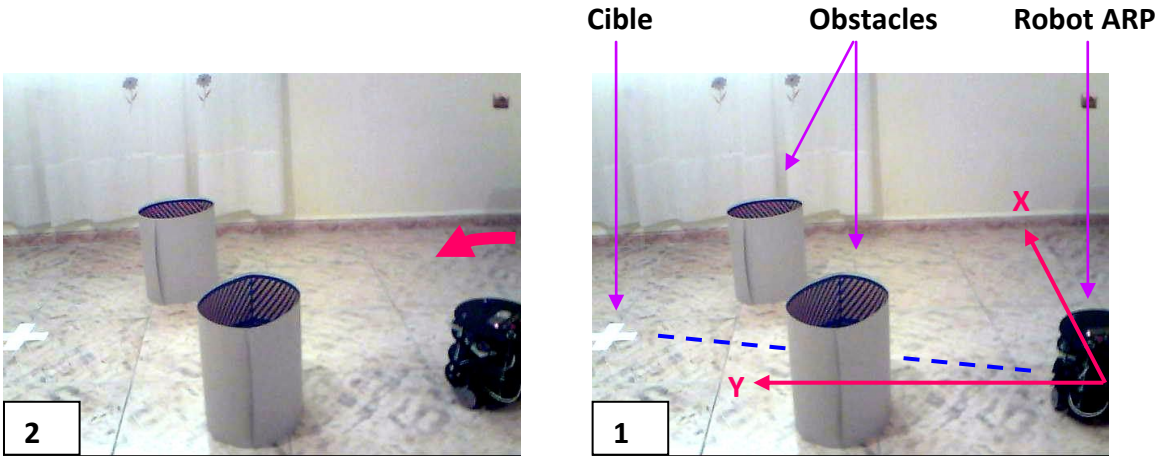


Figure 4.32 : Étapes 1 et 2 de la navigation réactive floue du robot ARP dans un environnement peu encombré.

Discussion 6: Dans les étapes 1 et 2 de la figure 4.32, les obstacles sont suffisamment loin du robot ARP. Ce dernier effectue un pivotement vers la gauche et se dirige vers le point destinataire qui se trouve à la position **(120,130)** marqué par une croix blanche dans le terrain (voir figure 4.32). Le navigateur flou exécute la huitième règle et génère une vitesse maximale et un angle **θ = arctg [(130-0) / (120-0)] = 47°.** La distance à parcourir dans le cas d'une ligne droite est **D = √ [(0-120) * (0-120) + (0-130) *(0-130)]=177 cm.**

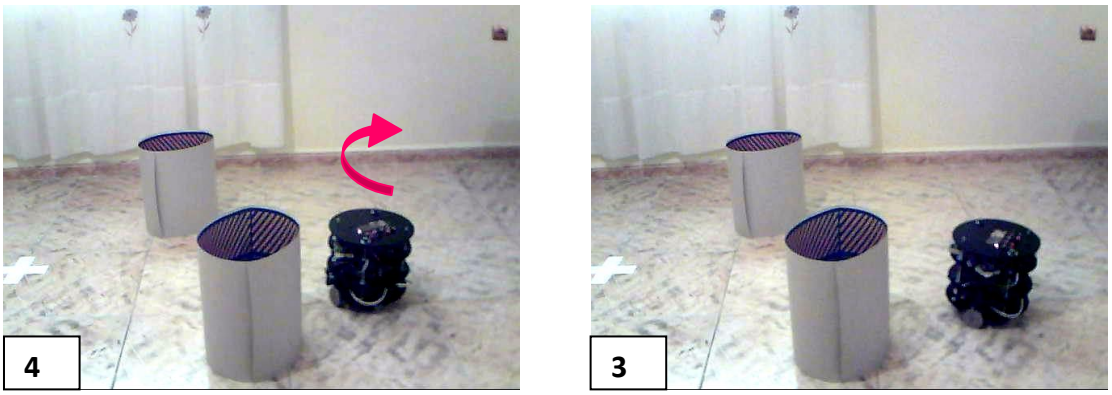


Figure 4.33 : Étapes 3 et 4 de la navigation réactive floue du robot ARP dans un environnement peu encombré.

Discussion 7: Dans les étapes 3 et 4 de la figure 4.33, le robot ARP détecte un obstacle à sa gauche. Le navigateur flou du robot exécute une des sept premières règles floues et diminue progressivement la vitesse de translation du robot et change son angle de braquage dans le sens opposé de l'obstacle. Le robot est donc en train d'éviter l'obstacle.

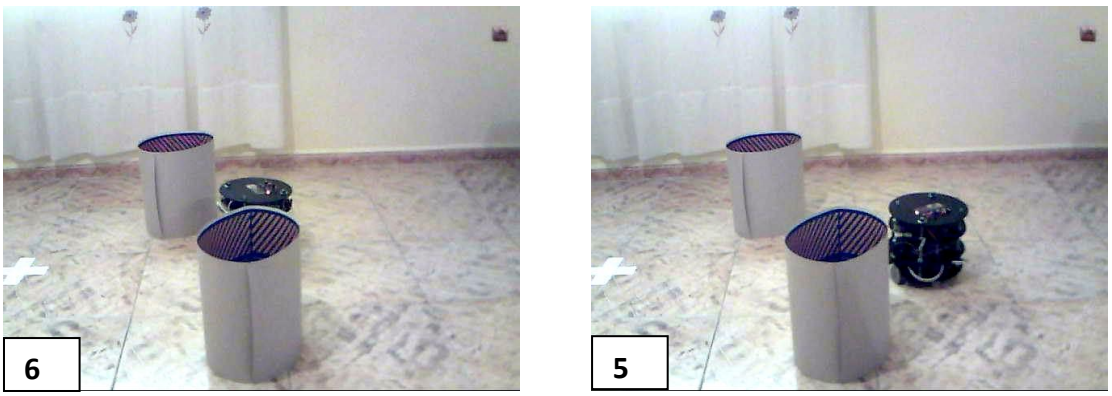


Figure 4.34 : Étapes 5 et 6 de la navigation réactive floue du robot ARP dans un environnement peu encombré.

Discussion 8: Dans l'étape 5 de la figure 4.34, le robot ARP sort de sa situation de détection d'obstacles et recalcule à nouveau sa direction à partir de sa position actuelle **(60, 30)**. Le navigateur flou exécute la huitième règle et génère une vitesse maximale et un angle **θ = arctg [(130-30) / (120-60)] = 50°.** La distance qui reste à parcourir dans le cas d'une ligne droite est **D = √ [(60-120) * (60-120) + (30-130) *(30-130)]=116 cm.** On remarque que la distance qui sépare le robot a la cible s'est diminuée, le robot se rapproche donc de son point destinataire. Dans l'étape 6 et pendant que le robot se dirige vers son point destinataire, le robot ARP détecte a nouveau un obstacle mais

cette fois ci a sa droite. Le navigateur flou génère la vitesse et l'angle de braquage adéquat pour sortir de cette situation.

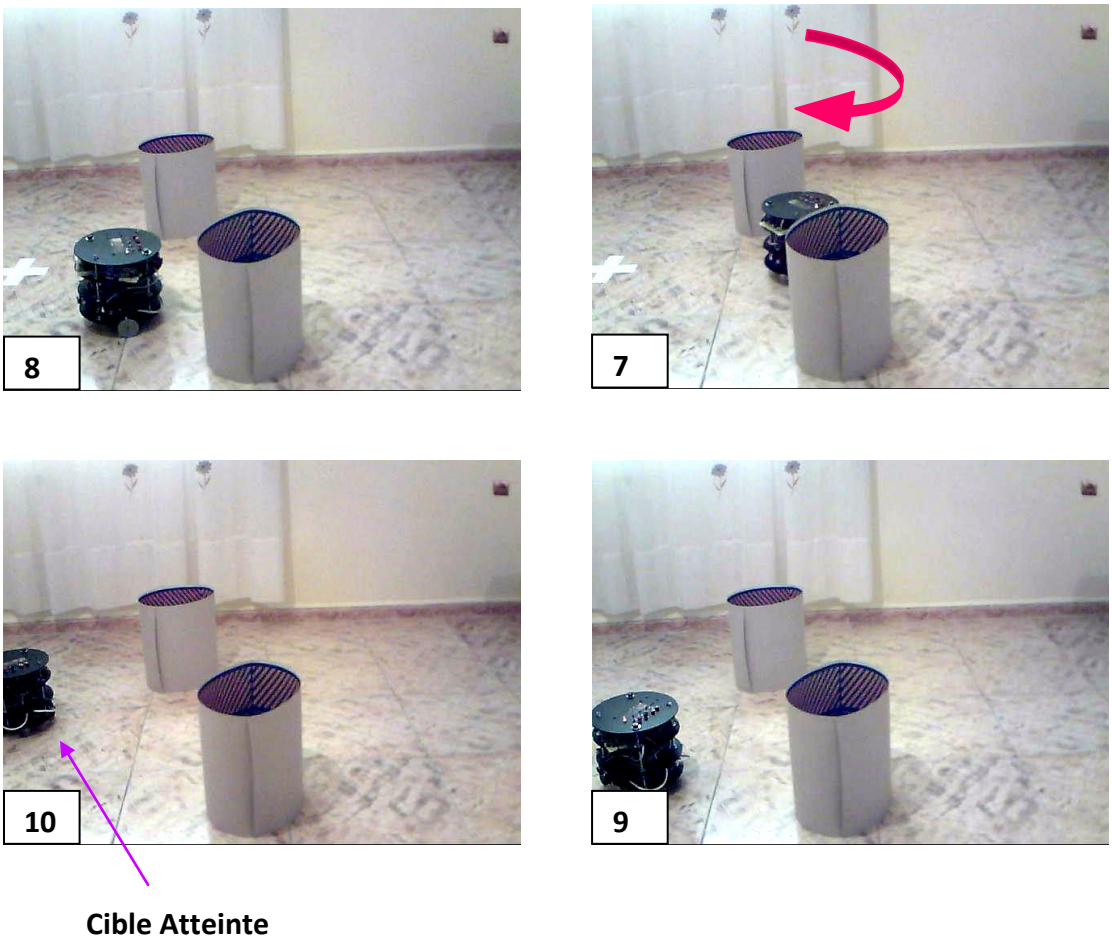


Figure 4.35 : Étapes 7, 8, 9 et 10 de la navigation réactive floue du robot ARP dans un environnement peu encombré.

Discussion 9: Dans l'étape 7 de la figure 4.35, le robot ARP se dirige vers la gauche en évitant l'obstacle qui se trouve à sa droite. Une fois sortit de cette situation, dans l'étape 8 le robot corrige à nouveau son angle **θ** et continue dans l'étape 9 à avancer vers son point destinataire et vérifie à chaque instant si la distance **D <= 20 cm.** Dans l'étape 10, la condition de distance se vérifie, le robot s'arrête et sa mission est terminé (cible atteinte). Le robot ARP signale sa réussite en déclenchant une sonorité pendant 100 ms.

Le bilan est plutôt bon, nous avons obtenu de bon résultats de la navigation réactive floue du robot ARP dans un environnement peu encombré, le robot ARP parvient à éviter les différents obstacles et atteindre son but.

Test 2 :

Cette fois ci, nous allons refaire le même test précédent mais dans un environnement plus encombré. Nous avons alors augmenté le nombre d'obstacles qui se trouvent dans l'environnement. Et la distance à parcourir par le robot ARP est aussi augmentée. Dans ce scenario le robot doit se rendre en **(40,300) cm** en partant de **(0,0) cm**.

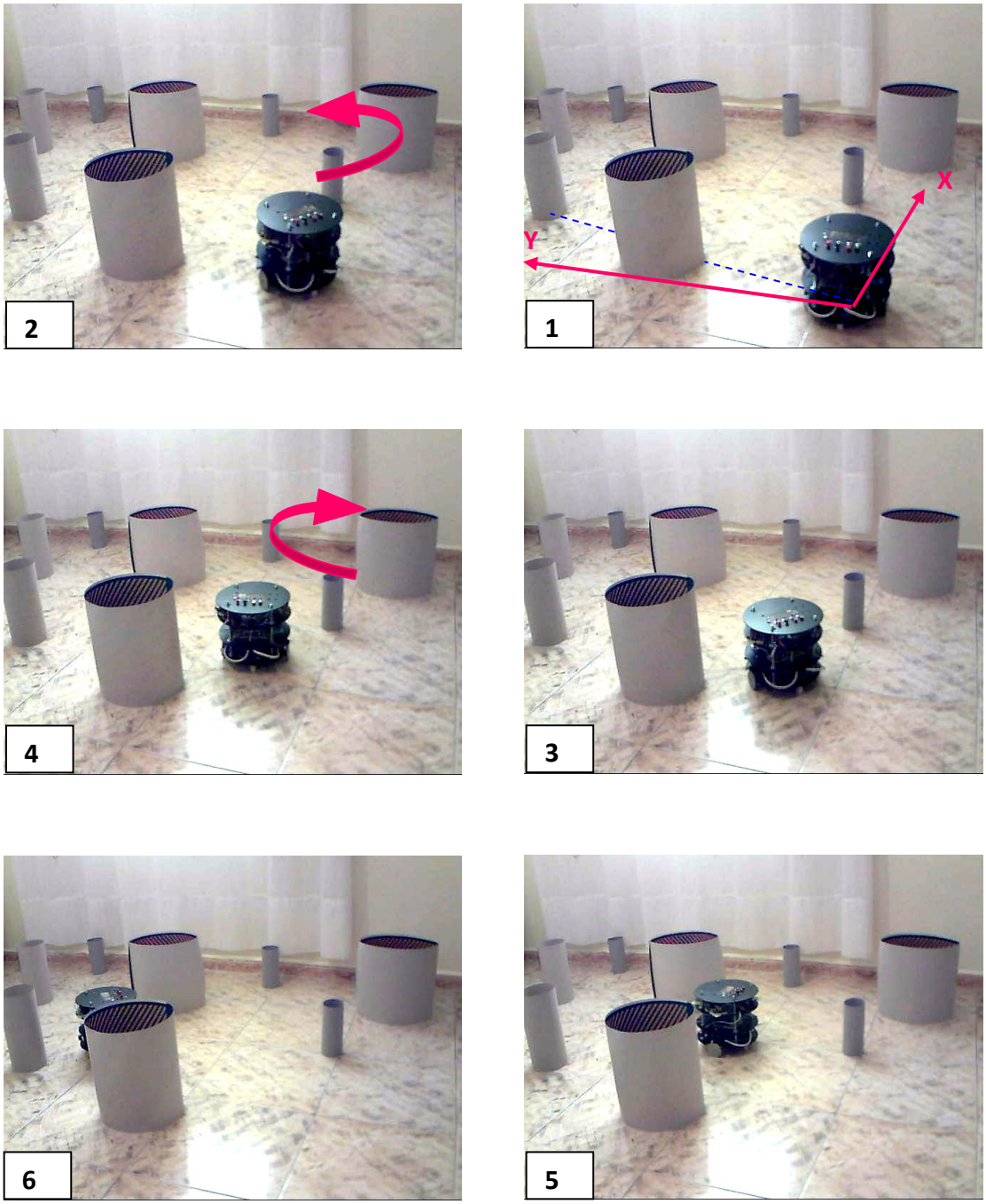


Figure 4.36 : Étapes de 1 à 10 de la navigation réactive floue du robot ARP dans un environnement très encombré.

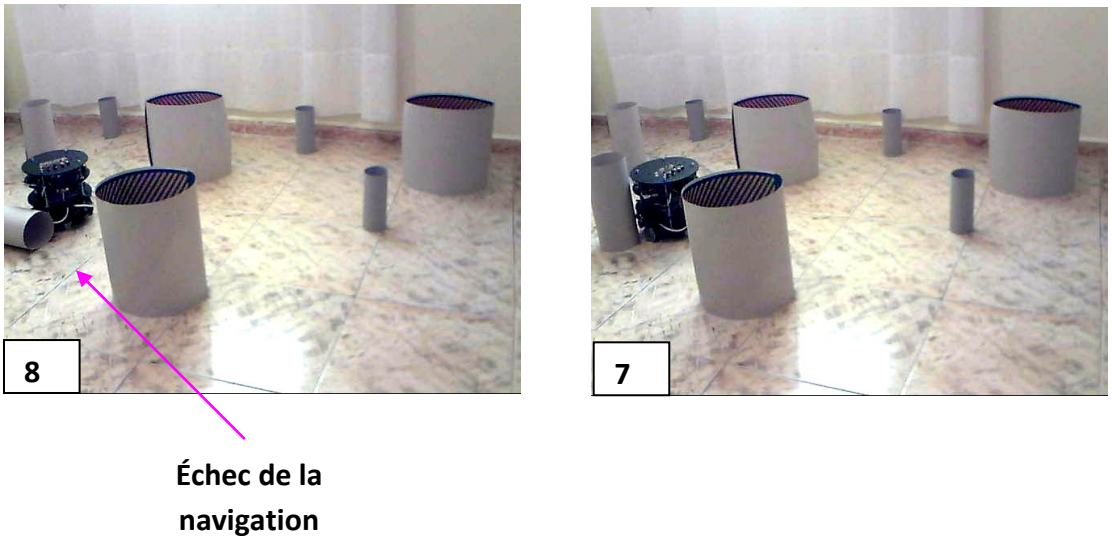


Figure 4.37 : Étapes 7 et 8 de la navigation réactive floue du robot ARP dans un environnement très encombré.

Discussion 10 : De l'étape 1 à 8 de la figure 4.36 et la figure 4.37, le robot ARP reçoit les coordonnées du point destinataire **(40,300) cm**, et démarre immédiatement et commence à se diriger vers son point but tout en évitant les obstacles. Mais au bout d'un moment, la navigation du robot ARP dans cet environnement encombré a échoué, et cela revient au nombre réduit des télémètres infrarouges dédiés à la détection d'obstacles, nous rappelons que le robot ARP est doté seulement de trois télémètres infrarouge linéaires, ce qui réduit son champs de vision. L'échec de navigation revient aussi à la technique de localisation utilisée (l'odométrie) qui est une technique avec une précision qui reste correcte sur de faibles distances (plus la distance est grande plus les erreurs sont accumulées). Et revient aussi aux odomètres du robot ARP qui ne sont pas assez précis, nous rappelons que ces codeurs ont une résolution de (1 impulsion par tour).

Nous avons aussi relevé certains problèmes lors des expériences réelles. On a en effet remarqué que le robot dérapé parfois sur un sol de carrelage mal fait, il est donc nécessaire que le sol soit le plus plat possible.

4.3 Conclusion :

Dans ce chapitre nous avons au début présenté la réalisation mécanique, électronique et informatique du robot ARP, nous avons aussi présenté son architecture de contrôle et quelques algorithmes nécessaires à sa configuration. Par la suite nous avons implémenté sur le robot notre système de navigation réactive floue (présenté au deuxième chapitre). Des expériences pratiques de navigation floue ont été faites avec le robot ARP. Ces expériences ont permis de tester et valider les algorithmes développés dans ce mémoire. Le résultat obtenu de la navigation floue du robot ARP dans un environnement peu encombré est satisfaisant. Cependant, la navigation du robot ARP a échouée dans un environnement très encombré. Nous avons conclu alors, que la maturité de la méthode de navigation réactive floue et sa capacité à résoudre les problèmes réels dépend surtout des capacités de localisation et de perception du robot ARP, il est donc nécessaire de doter le robot ARP d'un bon système de perception et de localisation.

Conclusion générale

Le travail que nous avons réalisé dans ce mémoire propose une stratégie de navigation réactive floue d'un robot mobile dans un environnement inconnu. Le robot doit donc défier son environnement en évitant les obstacles et en se dirigeant vers un point précis dans son espace de navigation. Le principe de cette approche de navigation floue a fait l'objet du deuxième chapitre. Il consiste à développer un navigateur flou qui permet de gérer la vitesse et l'angle de braquage du robot. Pendent la conception de ce navigateur flou, nous nous sommes attachés plus précisément à résoudre le problème de la détermination de ses paramètres à l'aide d'un système d'apprentissage off-line (neuro-flou) résumé dans le deuxième chapitre.

Une deuxième phase a été mise en œuvre a pour but de trouver le moyen permettant le test en toute sécurité du navigateur flou développé précédemment. Dans le troisième chapitre, nous avons présenté la plateforme de simulation que nous avons réalisée pour valider notre approche de navigation. Les résultats obtenus sont très satisfaisants.

L'objectif a été aussi de réaliser un robot mobile réel, et de tester les capacités de cette approche de navigation réactive floue sur un plan pratique. Nous avons donc réalisé le robot mobile ARP présenté au quatrième chapitre, ce robot est équipé de télémètres pour la détection d'obstacles et de codeurs optiques pour la localisation odométrique, et un module radio pour la communication avec le PC. Nous avons aussi développé dans ce travail une architecture de contrôle comportementale (réactive) présenté au quatrième chapitre, permettant à chaque étape de navigation de sélectionner le comportement adéquat à exécuter par le robot afin d'atteindre la configuration désirée.

Par la suite, ce navigateur réactif flou a été testé sur le plan pratique, nous l'avons donc implémenté sur le robot ARP, pendant la phase de navigation nous avons rencontré plusieurs problèmes techniques, surtout avec le matériel de mesure (capteurs) et les problèmes liés à l'environnement (sol déformé). Le robot ARP a pu mener sa mission seulement dans des environnements peu encombré, pour des raisons de matériels de perception et de localisation et même de capacité de calcul.

Les travaux effectués donnent lieu à plusieurs voies de recherche qu'il nous apparaît utile de creuser, les perspectives sont multiples. Deux pourraient par exemple être dégagées à court terme :

- La première concerne le matériel de perception et de localisation, il s'agit de rajouter au robot ARP, un ou plusieurs télémètres pour améliorer sa capacité de perception, et de l'équiper d'un bon système d'odométrie plus précis.

- La deuxième concerne les algorithmes d'apprentissages off-line, il serait intéressant de les remplacer par d'autres algorithmes d'apprentissages on-line avec la mise en place d'un PC embarqué. Il s'agit d'agrandir les capacités mémoire et traitements du robot mobile.

Cette liste non exhaustive fournit cependant une idée assez précise du travail restant à accomplir.

Bibliographie

- [Ahmad]. Ahmad M. Ibrahim, Ph.D. Senior Member, IEEE. Fuzzy Logic for Embedded Systems Applications. McMaster University, Hamilton, Ontario, Canada. 2004.
- [Arkin 1]. Arkin (R.C.). Dynamic Replanning for a Mobile Robot Based on Internal Sensing. In: Proc. of the IEEE International Conference on Robotics and Automation, pp. 1416.1421. Scottsdale, Arizona, USA, 1989.
- [Arkin 2]. Arkin (R.C.). Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. International Journal on Robotics and Autonomous Systems, vol. 6, n_ 1-2, 1990, pp. 105.122. ISSN 0921-8830.
- [Arkin 3]. Arkin, R. C. Motor Schema-Based Mobile Robot Navigation. The International Journal of Robotics Research, August 1989, pp. 92-112.
- [Arleo]. Arleo A, J. Del R. Millán, and D. Floreano. Efficient learning of variable resolution cognitive maps for autonomous indoor navigation. In IEEE Transactions on Robotics and Automation, volume 15, pages 990–1000, 1999. 3 citations pages 99, 156, 158.
- [Belker]. Belker, T., Schulz, D. Local Action Planning for Mobile Robot Collision Avoidance, Intelligent Robots and System. IEEE/RSJ International Conference on Volume 1, 30 Sept.-5 Oct. 2002 Page(s):601 - 606 vol.1.
- [Bilgic]. Bilgic. T and I. Turksen. Model based localization for an autonomous mobile robot. Proceedings of the 1995 IEEE Conference on Systems, Man and Cybernetics, pp. 6.
- [Borenstein]. Borenstein. J, B. Everett & L. Feng. Navigating mobile robots (Systems and techniques). A. K. Peters, Ltd. Wellesley, 1996.
- [Brooks 1]. Brooks (R.A.). A Hardware Retargetable Distributed Layered Architecture for Mobile Robot Control. In: Proc. of the IEEE International Conference on Robotics and Automation, pp. 106.110. Raleigh, USA, April 1987.
- [Brooks 2]. Brooks (R.A.). Intelligence without representation. Artificial Intelligence, 1(47):139–159. 1991.
- [Brooks 3]. Brooks, R. A. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, 1986, pp. 14-23.

- [BUHLER]. BUHLER. H. Réglage par logique floue. Presse polytechniques et universitaires Romandes. 1994.

- [Buhmann]. Buhmann. J, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot rhino. AI Magazine, 16(1), 1995. 2 citations pages 156,160.

- [Cartwright]. Cartwright. B. A and T. S. Collett. Landmark maps for honeybees. Biol. Cybern. 57:85–93.1987.

- [C. Biihler]. Christian Biihler, Ralf Hoelper, Helmut Hoyer, Wolfram Humann. Autonomous robot technology for advanced wheelchair and robotic aids for people with disabilities. Robotics and Autonomous Systems 14 (1995) 213- 222.

- [Cirstea]. Cirstea. M. N, A. Dinu, J.G. Khor, M. McCormick. Neural and Fuzzy Logic Control of Drives and Power Systems. 2002.

- [Collin]. Collin (I.), Meizel (D.), Lefort (N.), Govaert (G.). Local Map Design and Task Function Planning for Mobile Robots. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 273.280. Munchen, Germany, 1994.

- [D. Hong]. Daehie Hong, Steven A. Velinsky and Kazuo Yamazaki. Tethered mobile robot for automating highway Maintenance operations. Robotics & Computer- Integrated Manufacturing, Vol. 13, No. 4, pp. 297-307, 1997.

- [D. FILLIAT]. David FILLIAT. Robotique Mobile. École Nationale Supérieure de Techniques Avancées Paris Tech. 5 Octobre 2011.

- [D. Bystrov]. Dmitry Bystrov, Jerker Westin. Practice. Neuro-Fuzzy Logic Systems. Matlab Toolbox GUI. 2010.

- [Er M.J]. Er, M.J, Tien Peng Tan, Sin Yee Loh. Control of a mobile robot using generalized dynamic fuzzy neural networks. Microprocessors and Microsystems, Volume 28, Issue 9, 2 November 2004, Pages 491-498.

- [Foudil]. Foudil Abdessemed, Khier Benmahammed, Eric Monacelli. A Fuzzy - based reactive controller for a non-holonomic mobile robot. Robotics and Autonomous Systems 47 (2004) 31–46.

- [G. Frappier]. G. Frappier. Système inertiels de navigation pour robots mobiles. Séminaire. Les robots mobiles. EC2, Paris, 1990.

- [Gaussier]. Gaussier. P, C. Joulain, J.P. Banquet, S. Lepretre, and A. Revel. The visual homing problem: an example of robotics/biology cross-fertilisation. Robotics and autonomous systems, 30(1-2):155–180, 2000.

- [Gourichon]. Gourichon. S and J.-A. Meyer. Using colored snapshots for short-range guidance in mobile robots. International Journal of Robotics and Automation, submitted for publication, Special Issue on Biologically Inspired Robots, 2001.

- [H. Zerfa]. Houcine Zerfa. Conception et réalisation d'un Robot Mobile devant être télé-opéré par internet. Mémoire d'ingénieur de l'université des sciences et de la technologie d'Oran. Algérie.2009.

- [J. L. Merri]. Jean-Louis Merrien. Analyse Numérique Avec Matlab. Exercices et problèmes. Série DUNOD. 2007.

- [Kadri]. Kadri. M, O. Azouaoui, R. Kezerli et A-H Hariz. Comportement d'évitement d'obstacles basé sur les réseaux de neurones et la logique floue pour un robot mobile autonome. 4th International Conference on Computer Integrated Manufacturing CIP'2007.

- [Khatib]. Khatib. O. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. 1985 IEEE International Conference on Robotics and Automation St. Louis Missouri. March 25-28, 1990, pp. 500-505.

- [Lambrinos]. Lambrinos. D, R. Möller, T. Labhart, R. Pfeifer, and R. Wehner. A mobile robot employing insect strategies for navigation. Robotics and Autonomous Systems, special issue: Biomimetic Robots, 30:39–64, 2000.

- [L. Flavell]. Lance Flavell. Beginning Blender: Open Source 3D Modeling, Animation, and Game Design. 2010.

- [Latombe]. Latombe. J.-C. Robot Motion Planning. Boston: Kluwer Academic Publishers, Boston, 1991. 3 citations pages 97, 156, 157.

- [Laumond]. Laumond. J.-P. Robot Motion Planning and Control. Lectures Notes in Control and Information Sciences 229. Springer, 1998. 3 citations pages 155, 157, 163.

- [Lebedev]. Lebedev, D.V., Steil, J.J. and Ritter, H.J. The dynamic wave expansion neural network model for robot motion planning in time-varying environments, Neural Networks, Volume 18, Issue 3, April 2005, Pages 267- 285.

- [LI]. LI Yunwang, GE Shirong, ZHU Hua, FANG Haifang, GAO Jinke. Mobile platform of rocker-type coal mine rescue robot. Mining Science and Technology 20 (2010) 0466–0471.

- [Lin]. Lin. C.T., C.S. Lee, Neural network based fuzzy logic control and decision system, IEEE Trans. Comput. 40 (1991) 1320–1336.

- [M. Sugeno 1]. M. Sugeno, G.T. Kang. Structure identification of fuzzy models, Fuzzy Sets Syst. 28 (1988) 15–33.

- [Mamdani]. Mamdani. E. H. Application of fuzzy algorithms for control of simple dynamic plant. (Proc. IEEE, vol.121. No. 12, 1976).

- [M. ALDON]. Marie-José ALDON. Capteurs et méthodes pour la localisation des robots mobiles. Laboratoire d'informatique, de robotique et de microélectronique de Montpellier (LIRMM), UMR CNRS, université de Montpellier II, France. Techniques de l'ingénieur 1999.

- [Mataric]. Mataric, M. Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior. Journal of Experimental & Theoretical Artificial Intelligence, 9, 323-336. (1997).

- [M. Negnev]. Michael Negnevitsky. Artificial Intelligence A Guide to Intelligent Systems Second Edition. 2005.

- [M. Sugeno 2]. Michio Sugeno and Takahiro Yasukawa. A Fuzzy-Logic-Based Approach to Qualitative Modeling. School of the Tokyo Institute of Technology, Yokohama, Japan. Tokyo Institute of Technology, Japan. (IEEE TRANSACTIONS ON FUZZY SYSTEMS, VOL. l, NO. 1. FEUR AR.Y 1993).

- [Mikro 1]. MikroElektronika. Development tools - Books – Compilers. MikroC User's manual. 2002.2008.

- [Mikro 2]. MikroElektronika. Robotics experiment with PIC microcontroller based on RoboPICA robot kit. 3rd Edition. 2002.2008.

- [Motlagh]. Motlagh. O, S.H.Tang, N.Ismail, A.R.Ramli. An expert fuzzy cognitive map for reactive navigation of mobile robots. Fuzzy Sets and Systems 201 (2012)105–121.

- [N. MORET]. Nicolas MORETTE. Contribution à la Navigation de robots mobiles : approche par modèle direct et commande prédictive. Thèse de doctorat de l'université d'Orléans, France. 2009.

- [Ouadah]. Ouadah, N., Azouaoui, O., Hamerlain, M. Implémentation d'un contrôleur flou pour la navigation d'un robot mobile de type voiture. Troisième Congrès francophone, Majecstic 2005, 16-18 Novembre 2005, Rennes (France).

- [P. Reignier]. Patrick Reignier. Pilotage réactif d'un robot mobile étude du lien entre la perception et l'action. Thèse de doctorat de l'institut National Polytechnique De Grenoble. Décembre 1994.

- [Petru]. Petru Rusu, Emil M. Petriu, Thom E. Whalen, Aurel Cornell, and Hans J. W. Spoelder. Behavior-Based Neuro-Fuzzy Controller for Mobile Robot Navigation. IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 52, NO. 4, AUGUST 2003.

- [P. PREUX]. Philippe PREUX. Fouille de données. Notes de cours. Université de Lille 3. France. 2006.

- [Pruski]. Pruski A. Robotique mobile, la planification de trajectoire. ED. Hermes, Paris 1996.

- [Rodney]. Rodney A. Brooks. How to Build Complete Creatures Rather than Isolated Cognitive Simulators. In Architectures for Intelligence, pages 225– 239, 1991.

- [R. Jang 1]. Roger Jang JS. ANFIS: Adaptive-network-based fuzzy inference system. IEEE Trans Syst, Man Cybernetics 1993; 23(3):665–85.

- [R. Jang 2]. Roger Jang JS. Sun CT. Neuro-fuzzy modeling and control. Proc IEEE 1995; 83(3):378–406.

- [R. Jang 3]. Roger Jang JS. Neuro-fuzzy modeling: architecture, analyses and applications, Dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, 1992.

- [R. Jang 4]. Roger Jang JS. Sun C. Functional equivalence between radial basis function networks and fuzzy inference systems. IEEE Transn Neural Networks 1993; 4:156–9.

- [R. Jang 5]. Roger Jang JS. Fuzzy controllers based on temporal back propagation, IEEE Trans. Neural Netw. 3 (1992) 714–723.

- [R. Jang 6]. Roger Jang JS. Input selection for ANFIS learning. In: Proceeding of the fifth. IEEE international conference on fuzzy systems. vol. 2. 1996. p. 1493_9.

- [R. SIEGW]. Roland SIEGWART, Illah R. NOURBAKHSH. Introduction to Autonomous Mobile Robots. 2004. Massachusetts Institute of technology.

- [Shen]. Shen JC. Fuzzy neural networks for tuning PID controller for plants with under damped responses. IEEE Trans Fuzzy Syst 2001; 9(2): 333–42.

- [Song]. Song, K, Chang, C. Reactive Navigation in Dynamic Environment Using a Multisensor Predictor. IEEE Transactions on Systems, Man, and Cybernetics, 29, 870-880. (1999).

- [Steven M 1]. Steven M. Lavalle. Planning Algorithms. Cambridge University Press, May 2006. 2 citations pages 155, 163.

- [Steven M 2]. Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998. Page 157.

- [T. Bräunl]. Thomas Bräunl. Embedded Robotics Third Edition, Mobile Robot Design and Applications with Embedded Systems. School of Electrical, Electronic and Computer Engineering The University of Western Australia 35 Stirling Highway, M018 Crawley, Perth, WA 6009 Australia. 2008.

- [Thrun]. Thrun. S. Learning metric-topological maps for indoor mobile robot navigation. Artificial Intelligence, 99(1):21–71, 1999. 10 citations pages 89, 99, 100, 108, 111, 112, 141, 143, 156, 160.

- [Tak Sug]. Tomohiro Takagi and Michio Sugeno. Fuzzy Identification of Systems and Its Applications to Modeling and Control. School of the Tokyo Institute of Technology, Yokohama, Japan. (IEEE transactions on systems, MAN, and cybernetics, vol. SMC-15. No 1.1985).

- [Valentino]. Valentino Braitenberg. Vehicles: Experiments in Synthetic Psychology. The MIT Press. 1986.

· [W.L. Xu]. W.L. Xu, S.K. Tso, Y.H. Fung. Fuzzy reactive control of a mobile robot incorporating a real/virtual target switching strategy. Robotics and Autonomous Systems 23 (1998) 171-186.

· [Wang]. Wang, X., Hou, Z., Zou, A., Tan, M., Cheng, L. A behavior controller based on spiking neural networks for mobile robots, Neuro-computing Volume 71, Issue 4-6 (January 2008) Pages 655-666.

· [Yang]. Yang-Ge Wu, Jing-Yu Yang & Ke Liu. Obstacle detection and environment modeling based on multisensor fusion for robot navigation. Artificial Intelligence in Engineering 10 (1996) 323-333.

· [Yann]. Yann-LEIDWANGER. Robots mobiles intelligents. Du capteur au comportement. ETSF. DUNOD. 2006.

· [ZINSER]. ZINSER. K, R. SCHREIBER. La logique floue: une nouveauté prometteuse de la technique d'automatisation. La Technique moderne. N° 1-2- 1994.

M. Houcine ZERFA, né le 16 septembre 1984 à Oran, a obtenu son baccalauréat technique série Génie Mécanique en juin 2002. Il est titulaire d'un diplôme d'Ingénieur d'État en Électronique option « Contrôle » de l'Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf USTO MB, obtenu en juillet 2009. Inscrit en Magister en Électronique option « Automatique Robotique Productique » à l'Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf, depuis janvier 2010. Son domaine de recherche est orienté vers la réalisation et contrôle des robots mobiles par des systèmes intelligents.

Conception, Réalisation et Commande Floue d'un Robot Mobile

Résumé :

Nous avons présenté dans ce document la conception, réalisation et commande floue d'un robot mobile. Nous avons réalisé un système de navigation réactive basé sur la logique floue, ce système, nous l'avons ajusté en off-line par un algorithme neuro-flou. Nous avons aussi réalisé une plateforme de simulation en 3D pour la validation de cette approche de navigation floue. Nous avons ensuite réalisé un robot mobile de type uni-cycle à roues différentielles, doté de deux odomètres et trois télémètres infrarouges et un module radio pour la communication avec le PC. Le robot est asservi en vitesse et en position par un régulateur classique PID. Enfin nous avons réalisé un logiciel de supervision qui permet d'envisager des points but dans l'espace de navigation que le robot doit atteindre tout en assurant la non-collision avec les obstacles.

Mots-clés : Robot Mobile (Différentiel), Localisation et Perception des Robots Mobiles, Odométrie, Navigation réactive floue, Simulation des Robots Mobiles, Logique Floue, Réseaux de Neurone, Système Neuro-Flou, ANFIS.

Conception, Realization and Fuzzy Control of Mobile Robot

Abstract:

The work presented in this document concerns the conception, realization and fuzzy control of mobile robot. We have proposed a reactive navigation system based on the fuzzy logic, this system allows the robot to navigate terrain and reach its target with avoiding the obstacles. Also we have adjusted this system by an off-line neuro-fuzzy algorithm. And we have achieved a 3D platform of simulation for the validation of our approach. Also we have realized a real differential mobile robot, equipped with two odometers and three infrared sensors and one radio module for the communication between the robot and computer. The velocity and position of robot are controlled by a classical PID. And finally we have realized a software application that permits to send a target punctual coordinates toward the robot.

Keywords: Differential Mobile Robot, Mobile Robots Navigation, Localization and Perception, Odometery, Fuzzy reactive navigation, Mobile Robot Simulation, Fuzzy Logic, Neuronal Networks, Neuro-Fuzzy System, ANFIS.