UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
MAXIME BOISVERT

.

STRATÉGIE DE COMMUNICATION SANS FIL POUR LA GESTION DE VÉHICULES AUTONOMES

OCTOBRE 2009

Résumé

Un banc expérimental constitué d'une chaîne de véhicules qui devant se déplacer en convoi est en voie de réalisation. Des fonctions de mesure, de commande et de communication permettront de donner le comportement désiré aux véhicules. Le système de communication permettra aux voitures de communiquer les unes avec les autres. Les besoins en communication sont principalement la transmission et la réception de données spécifiant les consignes transmises aux véhicules. Sachant que plusieurs protocoles de communication permettent des communications bidirectionnelles entre une centrale et des mobiles, il s'agit d'identifier ceux qui répondent aux besoins de l'application. Plus particulièrement, ceux qui assurent le transport de petits paquets à de très grandes vitesses. Nous avons choisi la communication ad hoc spécifique à l'application basée sur le protocole Zigbee.

Une plateforme de simulation, programmée en C#, a été développée afin de visualiser la mise en application des stratégies de contrôle d'un convoi de véhicules. En fonction des consignes, il est possible de visualiser des ralentissements ou dépassements de véhicules. De plus, cette plateforme a été reliée à des cartes de développement CC2430 qui émulent les véhicules. Chacune de ces cartes dispose, entre autres, d'un microcontrôleur programmable, d'une puce exécutant le protocole Zigbee et une d'antenne. Ainsi, une émulation de communications entre véhicules a été réalisée.

Table des matières

Liste des tableaux

Liste des figures

Chapitre 1 - Introduction

1.1 Problématique :

La gestion de véhicules autonomes était à la base, un projet militaire. L'intérêt premier était d'assurer un convoi de véhicules sécuritaire et automatisé. Récemment, avec le développement de la technologie à coût abordable, plusieurs recherches sont en cours afin de proposer des stratégies de contrôle et des communications fiables entre véhicules. Un projet sur les autoroutes intelligentes est en cours de réalisation au département de génie électrique et génie informatique. Mon projet traite l'aspect communication entre les véhicules.

La communication sans fil entre les différentes voitures est une partie critique du projet. Le choix du dispositif dépend de plusieurs paramètres. Le prix, la distance couverte, le débit et la puissance disponible sont des exemples de caractéristiques qui doivent être étudiées. Sachant que les véhicules physiques considérés sont à une échelle réduite, la distance à couvrir n'est pas très grande, tout au plus quelques mètres. De plus, la puissance disponible sur les maquettes sera de beaucoup inférieure à ce que l'on peut retrouver sur une voiture conventionnelle. Enfin, le débit d'information à transmettre ne sera pas aussi important que pour une vraie autoroute, dû à un nombre de voitures limité sur la maquette (4 voitures maximum). Il sera donc nécessaire de faire un compromis sur le débit et la distance couverte afin de gagner une certaine autonomie.

Mis à part la communication sans fil, il faut prévoir une communication entre l'unité « sans fil » et l'unité centrale. L'unité centrale représente le dispositif qui contrôle la voiture. Puisque ces deux unités sont séparées, la communication devra passer par un protocole câblé. L'unité centrale n'étant pas précisée, il sera nécessaire d'utiliser un protocole présent dans la plupart des microprocesseurs, PC104 et autres équipements permettant l'automatisation des voitures. En plus de cette dernière problématique, le bon fonctionnement de cette communication ne sera assuré que si les données transmises sont validées et qu'il est possible de confirmer l'intégrité des données. Pour ce faire, un protocole de niveau supérieur devra être implanté.

La simulation sur une véritable maquette comporte quelques désavantages. En plus de confronter le programmeur aux différentes avaries mécaniques, elle lie son travail à la personne responsable des maquettes. C'est donc dire que le résultat de l'un dépend du travail de l'autre. Afin de limiter les délais, il sera nécessaire de faire un simulateur, qui permettra de simuler les voitures et de valider la communication sans fil. Ce qui permettra aux différents participants du projet d'être indépendants les uns des autres.

1.2 **Objectifs :**

1.2.1 Simulateur :

Les objectifs sont nombreux. En premier lieu, un simulateur sera créé afin de valider la communication sans fil. Ce simulateur devra reproduire le comportement des voitures. Afin de rendre la simulation réelle, nous devrons concevoir un algorithme pour la gestion du trafic, le contrôle de la vitesse et la direction des voitures. De plus, les différents contrôleurs

utilisés pour asservir les voitures pourront être simulés et validés. Il sera ainsi plus facile de visualiser et de documenter les avantages des différents contrôleurs. L'objectif est donc d'utiliser des contrôleurs de base comme le PI, PID et la logique floue afin de prouver le bon fonctionnement du simulateur.

1.2.2 Communication sans fil :

Puisque les caractéristiques de nos maquettes seront uniques, il sera nécessaire de prendre en considération la distance à couvrir, la puissance disponible et le débit maximal désiré. Comme mentionné plus haut, il sera nécessaire de faire un compromis sur le débit et la distance couverte afin de gagner une certaine autonomie. Une étude des différents moyens de communication sera effectuée, afin de choisir le plus approprié à notre projet.

1.2.3 Protocole de communication :

Afin d'assurer l'authenticité des données lors de la communication sans fil et câblée, nous devrons implanter un algorithme qui garantira qu'aucune donnée transmise n'est erronée. Cet algorithme devra être simple et efficace. Une étude sera donc faite afin de choisir ce dernier.

Chapitre 2 - Modélisation du Système

2.1 **Modèle mathématique :**

Le comportement dynamique d'un système physique peut être représenté par une équation, ou un ensemble d'équations. Cette dernière nous sert de modèle mathématique de ce système. Ce modèle peut être construit à partir des caractéristiques physiques du système. Par exemple, la masse pour un système mécanique ou la résistance pour un système électrique. De plus, il est possible de modéliser et valider un système par expérimentation. Les mesures prises lors de l'expérimentation permettraient entre autres de calibrer le modèle. Il est donc nécessaire d'ajouter des gains et des limites afin d'avoir un modèle final le plus proche possible de la réalité.

2.1.1 Modèle simplifié d'une voiture conventionnelle :

Le modèle le plus simple est celui d'une masse qui se déplace [1]. L'accélération, la vitesse, la position et la direction sont toutes fonctions des forces qui sont appliquées sur cette masse. Si nous prenons le modèle d'une voiture conventionnelle, la commande de vitesse, ou plutôt de l'accélération, est faite par la pédale d'accélération. Un modèle mathématique simple qui lie l'angle de la pédale d'accélération à la vitesse:

$$v(t) = a\varphi$$

Où v est la vitesse et φ l'angle de la pédale des gaz. Cette représentation est plutôt simpliste car, tout le monde sait qu'une voiture prend un certain temps avant d'atteindre une vitesse donnée. Un modèle mathématique qui représente le comportement dynamique des systèmes est construit en utilisant les équations différentielles. Une représentation plus exacte du véhicule pourrait être :

$$b\frac{dv}{dt} + cv = e\theta(t)$$

Si la vitesse est constante, c'est-à-dire que l'accélération est nulle ($b\frac{\partial v}{\partial t} = 0$), nous retrouvons la première équation ($cv = e\theta(t)$) où $a = \frac{e}{c}$.

Afin de bien représenter la voiture, il faudrait tenir compte de la puissance du moteur, de la traction des roues et de l'aérodynamisme. Sans oublier les contrôles dont les voitures sont équipées; système anti dérapage, anti patinage, boîte de vitesse automatique, … etc.

2.1.2 Représentation mathématique d'une masse :

En représentant la voiture comme une masse qui se déplace et en utilisant la deuxième loi de Newton, nous arrivons aux équations suivantes [1]:

$$f(t) = ma(t) = m\frac{\partial v}{\partial t} = m\frac{\partial^2 x}{\partial t^2}$$

$$\sum f = ma$$

Pour une masse tournante :

$$T(t) = I\alpha(t) = I\frac{\partial \omega}{\partial t} = I\frac{\partial^2 \theta}{\partial t^2}$$

$$\sum M = I\alpha$$

En représentant la voiture comme une masse qui subit une force, provenant du moteur, et de la friction, nous arrivons au système masse amortisseur représenté sur la figure 2.1:

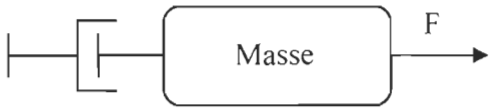


Figure 2.1 Système masse amortisseur

La force provenant de l'amortisseur est proportionnelle à la vitesse $f_a(t) = c\frac{\partial x}{dt}$ et la somme des forces est proportionnelle à l'accélération. $f(t) = m\frac{\partial^2 x}{\partial t^2}$.

$$F - c\frac{\partial x}{dt} = m\frac{\partial^2 x}{\partial t^2}$$

2.1.3 Système masse-ressort-amortisseur [1] :

Le modèle masse-ressort-amortisseur est un système du deuxième ordre [1].

$$k(x_i - x_0) - c\frac{\partial x_0}{dt} = m\frac{\partial^2 x_0}{\partial t^2}$$

$$m\frac{\partial^2 x_0}{\partial t^2} + c\frac{\partial x_0}{dt} + kx_0 = kx_i(t)$$

La représentation du système de deuxième ordre dans le domaine de Laplace, avec des conditions initiales nulles, s'écrit comme suit :

$$ms^2X_0(s) + csX_0(s) + kX_0(s) = kX_i(s)$$

$$(ms^2 + cs + k)X_0(s) = kX_i(s)$$

La fonction de transfert :

$$G(s) = \frac{X_0}{X_i}(s) = \frac{k}{ms^2 + cs + k}$$

Je reviendrai un peu plus tard sur ce système masse-ressort-amortisseur. Pour l'instant, concentrons-nous sur le moteur.

2.1.4 Modèle d'un moteur électrique:

Les voitures miniatures qui seront développées seront dotées de moteur à courant continu. Pour rendre la simulation plus réaliste, ainsi que pour tester d'éventuels contrôleurs pour les moteurs, voici une modélisation mathématique d'un moteur à courant continu. Le moteur à courant continu (cc) contient un enroulement inducteur ou un aimant permanent, une partie tournante ou rotor qui porte l'enroulement d'induit et des contacts frottant sur les lames du collecteur [1]. L'induit est alimenté par une tension U. Il a une résistance r et une inductance L. Le courant qui le traverse est I. le moteur tourne à une vitesse angulaire de Ω et la force électromotrice induite est E. Cette dernière est proportionnelle à la vitesse $E = K * \omega$.

La loi d'Ohm appliquée à l'induit s'écrit comme suit :

$$u(t) = e(t) + ri(t) + L\frac{\partial i(t)}{\partial t} = k\omega(t) + ri(t) + L\frac{\partial i(t)}{\partial t}$$

Dans le domaine de Laplace nous obtenons :

$$U(s) = K\omega(s) + rI(s) + LsI(s)$$

De plus en utilisant le modèle des corps en rotation et en utilisant la même logique que les équations obtenues en 2.1.2 et 2.1.3, nous arrivons à :

$$J\frac{\partial\omega(t)}{\partial t} + Cr = Cm$$

Où J est l'inertie totale ramenée sur l'arbre du moteur, Cm est le couple moteur qui correspond à : $Cm = Kj(t)$ et Cr est le couple résistant visqueux $Cr = c\omega$.

Cette dernière équation dans le domaine de Laplace devient :

$$Js\omega(s) = KJ(s) - c\omega(s)$$

Note : La simplification suivante n'a pas été faite dans le modèle final. Elle reste toutefois pertinente dans cette étude, c'est la raison pour laquelle elle est présentée ici.

En négligeant l'inductance L de l'induit, l'équation électrique du modèle s'écrit :

$$U(s) = K\omega(s) + rI(s)$$

En isolant le courant et en utilisant l'équation mécanique, nous arrivons à :

$$\frac{\omega}{U}(s) = \frac{K}{K^2 + Cr + Irs}$$

En posant : $K_0 = \frac{K}{K^2+Cr}$ et $\tau = \frac{Ir}{K^2+cr}$

$$\frac{\omega}{U}(s) = \frac{K_0}{1+\tau s}$$

Le résultat étant une équation du premier ordre.

2.1.5 Modèle d'un convoi :

Un convoi est une série de voitures. La voiture de tête sert de guide aux voitures qui la suivent. Tout au long de ce mémoire, je parlerai de la voiture de tête comme la voiture « maître » et des voitures qui la suivent comme les voitures « esclaves ». L'asservissement pour ces deux classes de voitures est complètement différent. La voiture de tête doit garder

une vitesse constante tout en s'assurent qu'elle n'entre pas en collision avec d'autres voitures, tendis que les autres voitures doivent garder une distance constante entre elles. C'est donc dire que le contrôle sur la voiture de tête doit être fait par rapport à sa vitesse de croisière, tandis que le contrôle pour les voitures qui la suivent doit être fait par rapport a la distance de la voiture qui la précède. Nous verrons plus en profondeur le modèle mathématique d'un convoi un peu plus loin.

2.1.6 Contrôleur en boucle fermée :

La figure 2.2 représente un système à boucle fermée. L'action du contrôleur vise à asservir le système en faisant tendre sa sortie vers une valeur qui se rapproche de l'entrée [1]. L'erreur E(t) finira donc par tendre vers zéro. La vitesse avec laquelle l'erreur diminue et la stabilité du contrôle sont toutes les deux dépendantes du contrôleur utilisé. Pour le contrôle des voitures, je me suis concentré sur les contrôleurs à une entrée, une sortie. Le proportionnel (P), le proportionnel intégré (PI), proportionnel intégré dérivatif (PID) et le contrôleur basé sur la logique floue sont les principaux contrôleurs que j'ai utilisés [1]. Le principe de base de ces contrôleurs est présenté ci-dessous.

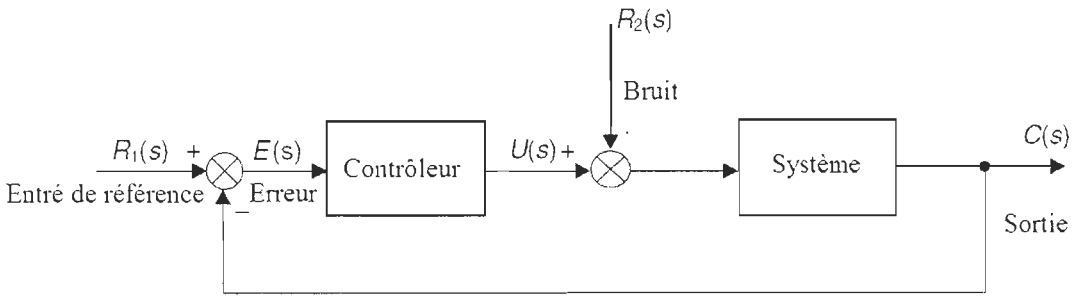


Figure 2.2 Système à boucle fermée

2.1.7 Contrôleur proportionnel :

Comme son nom l'indique, il s'agit d'un contrôleur qui commande le système proportionnellement à l'erreur. C'est donc un gain qui est ajouté à l'erreur qui forme l'entrée du système.

$$u(t) = K_1 e(t)$$

Ce contrôleur est très stable pour le modèle de premier ordre, mais il a le désavantage de laisser un écart entre la valeur désirée et la valeur voulue. Si nous utilisons une valeur de référence de UN comme entrée et un système du premier ordre :

$$\frac{K}{1 + T_s}$$

Nous arrivons à :

$$c(t) = \frac{K_1 K}{1 + K_1 K} \ , t \to \infty$$

Plus la valeur $K_1 K$ est grande, plus l'écart est petit.

2.1.8 Contrôleur proportionnel-intégral:

L'erreur produite par le contrôleur proportionnel peut être éliminée en ajoutant un terme intégral. Voici sa représentation mathématique :

$$u(t) = K_1 e(t) + K_2 \int e(t) dt$$

En utilisant la transformée de Laplace :

$$U(s) = \left(K_1 + \frac{K_2}{s}\right) E(s)$$

$$U(s) = K_1 \left(1 + \frac{1}{T_1 s}\right) E(s)$$

Où $T_1 = \frac{K_1}{K_2}$ et représente le temps d'action de l'intégrale.

Si nous utilisons une valeur unitaire comme entrée et un système du premier ordre, nous obtenons une réponse qui tend vers un. Pour un système de premier ordre, le PI produira une réponse du deuxième ordre. Il n'y aura pas d'erreur fixe pour une entrée constante.

2.1.9 Contrôleur proportionnel intégral dérivatif :

Le plus connu des contrôleurs et le plus utilisé est sans aucun doute le PID.

$$u(t) = K_1 e(t) + K_2 \int e(t)dt + K_3 \frac{de(t)}{dt}$$

En utilisant la transformée de Laplace :

$$U(s) = (K_1 + \frac{K_2}{s} + K_3 s) E(s)$$

$$U(s) = K_1 \left(1 + \frac{1}{T_1 s} + T_d s\right) E(s)$$

Ou encore :

$$U(s) = \frac{K_1 (T_1 T_d s^2 + T_1 s + 1)}{T_1 s} E(s)$$

Où T_d est appelé le temps d'action dérivative.

Ce contrôleur offre une bonne stabilité pour des systèmes d'ordre supérieur.

2.1.10 Modèle discret du système :

Comme la simulation sera faite en temps réel, il nous sera impossible d'utiliser les équations de Laplace directement dans notre code. Il nous faut utiliser la transformée en Z afin de passer d'un mode continu à un mode discret [1]. La transformée d'Euler et de Tustin donne de bons résultats

$$\text{Tustin :} \quad s = \frac{2(1-Z^{-1})}{T(1+Z^{-1})}$$

- P :
  Fonction de transfert :
  $$X_o(s) = KX_i(s)$$
  Transformée en Z :
  $$X_o(z) = KX_i(z)$$
- PD (Euler):
  Fonction de transfert :
  $$X_o(s) = K_1X_i(s) + K_2sX_i(s)$$
  Transformée en Z :
  $$X_o(z) = K_1X_i(z) + \frac{K_2}{T}(X_i(Z) - X_i(z-1))$$
- PI:
  Fonction de transfert :
  $$X_o(s) = K_1X_i(s) + \frac{K_2X_i(s)}{s}$$
  Transformée en Z :
  $$X_o(z) = K_1X_i(z) + K_2(X_o(Z-1) + (X_i(z) + X_i(z-1))\frac{T}{2})$$
- PID :
  Fonction de transfert :
  $$X_o(s) = K_1X_i(s) + \frac{K_2X_i(s)}{s} + K_3sX_i(s)$$
  Transformée en Z :
  $$X_o(z) = K_1X_i(z) + K_2\left(X_o(Z-1) + \left(X_i(z) + X_i(z-1)\right)\frac{T}{2}\right) + \frac{K_3}{T}\left(X_i(Z) - X_i(z-1)\right)$$

2.1.11 Logique floue :

Nous utilisons la logique floue exactement de la même façon que n'importe quel autre contrôleur. Dans l'exemple suivant, le modèle utilisé ressemble à un PI.

Les lois de commande utilisées sont représentées dans le tableau 2.1.

Tableau 2-1 Lois de commande

Numéros	Si E est :	Si ΔE est :	Alors ΔU
1	-	-	-
2	-	+	0
3	+	-	0
4	+	+	+

où E représente l'erreur, ΔE la variation de l'erreur, ΔU la variation de la commande et les signes – , + et 0 correspondent aux fonctions d'appartenance. Le lien entre les deux entrées (E et ΔE) est la logique ET (&) et le lien entre les sorties (ΔU) est le OU (||).

Nous pouvons écrire les 4 lois de commande suivantes :

1 : Si l'erreur est négative et la variation de l'erreur est négative, alors la variation de la commande doit être négative.

2 : Si l'erreur est négative et la variation de l'erreur est positive, alors la variation de la commande doit être nulle.

3 : Si l'erreur est positive et la variation de l'erreur est négative, alors la variation de la commande doit être nulle.

4 : Si l'erreur est positive et la variation de l'erreur est positive, alors la variation de la commande doit être positive.

L'appartenance à chacune des règles est « floue ». C'est-à-dire qu'elle peut varier selon la courbe d'appartenance. Ce calcul de l'appartenance est appelé la fuzzification. Voici un exemple qui explique bien ce concept (voir la figure suivante). Pour une certaine erreur E, la fuzzification donnera deux sorties; la réponse – et +. En supposant que les deux courbes varient de 0 à 1, la sortie sera d'environs 0.3 pour – et 0.8 pour +. La sortie correspond à la valeur à laquelle le pointillé rencontre la courbe.

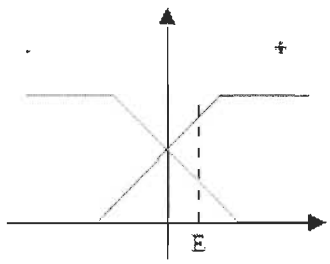
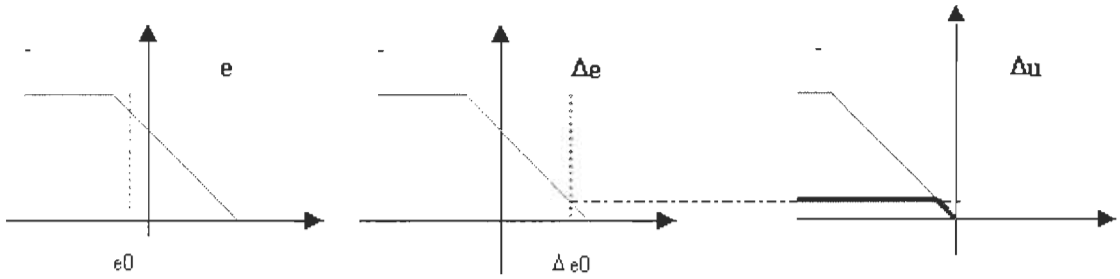


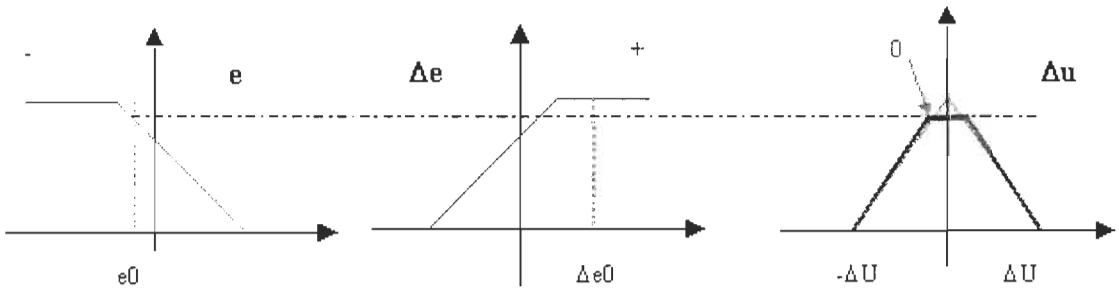
Figure 2.3 Fuzzification d'une variable

Une fois la fuzzification faite pour les deux entrées, il suffit de sommer les quatre règles pour obtenir une sortie. La figure 2.4 illustre bien cette étape. L'utilisation de la logique ET entre les deux entrées fait en sorte que nous devons utiliser la plus petite entrée afin de déterminer la sortie. Pour obtenir cette dernière, nous utilisons la surface qui reste lorsque nous traçons une ligne en dessous de la courbe de commande d'une hauteur égale à la plus petite fuzzification (voir la figure 2.4).

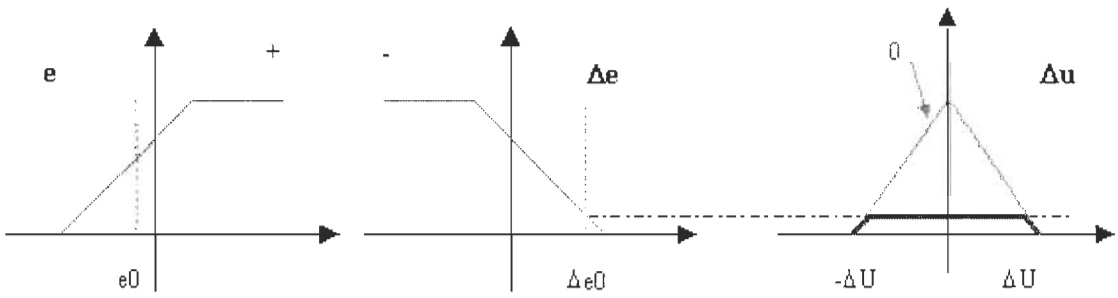
Règle 1 (- et – donne -):



Règle 2 (- et + donne 0):



Règle 3 (+ et – donne 0) :



Règle 4 (+ et + donne +) :

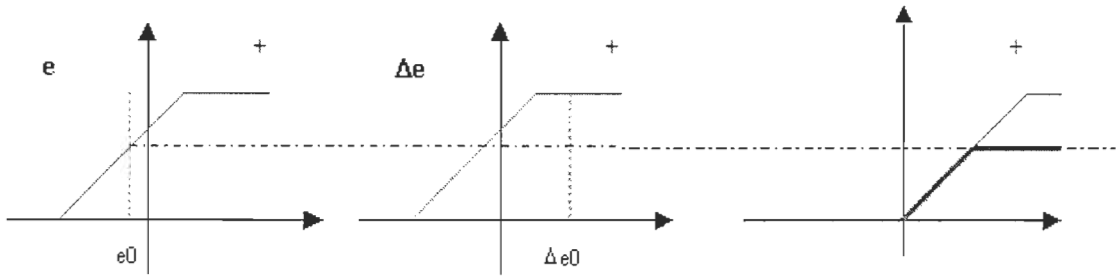


Figure 2.4 Règles floues

Puisque la logique qui unit les quatre sorties est OU, nous devons unir les sorties en utilisant la plus grande d'elles pour chaque point de la courbe (voir la figure suivante).

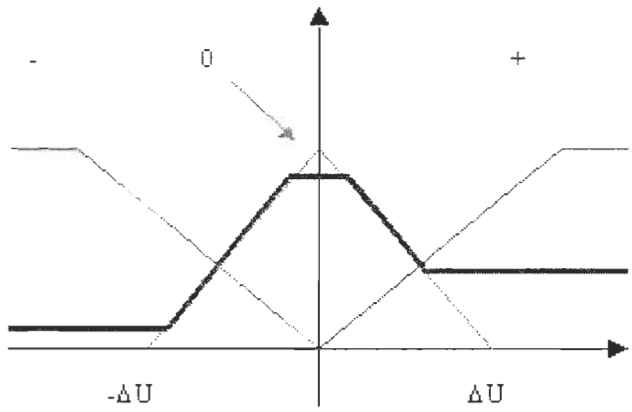


Figure 2.5 Logique du OU

Enfin, il faut procéder à la défuzzification barycentrique qui consiste à calculer l'abscisse du centre de gravité. Noter qu'il existe plusieurs autres méthodes de défuzzification.

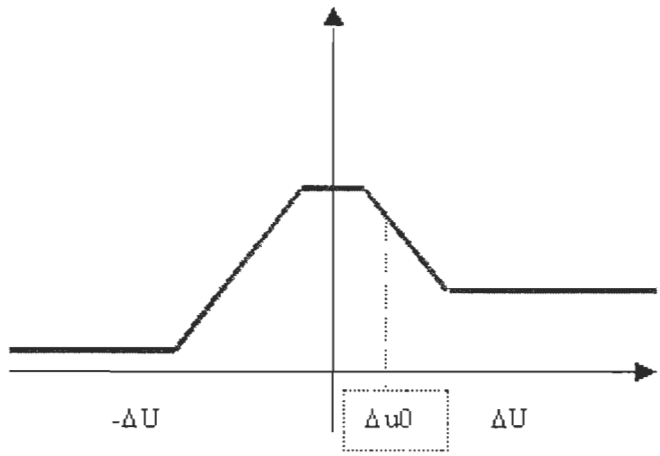


Figure 2.6 Défuzzification

2.2 **Communication :**

2.2.1 Communication sans fil :

Les techniques de transmission sont au nombre de quatre :

1. L'infrarouge
2. Le laser
3. La radio à bande étroite (fréquence unique)
4. La radio à spectre étalé

2.2.1.1 Infrarouge :

L'infrarouge est un faisceau de lumière. Les transmissions en infrarouge doivent être très intenses afin qu'il n'y ait pas de confusion avec les nombreuses sources de lumière qui existent dans une pièce (fenêtres, néons, télévision, ampoules, ...). La lumière infrarouge possède une large bande passante; les débits sont relativement importants, mais la portée est faible :

- 10 Mb/s.
- 30 mètres

Un réseau infrarouge est commode, rapide, mais sensible aux interférences lumineuses. Le faisceau ne doit jamais être coupé sinon la transmission est interrompue.

Il existe quatre types de réseaux infrarouges :

1. Les réseaux à visibilité directe (les émetteurs et les récepteurs doivent être proches les uns des autres)
2. Les réseaux infrarouges à diffusion (les signaux infrarouges se réfléchissent sur les murs et les plafonds sur une distance de 30 mètres, mais le débit est lent en raison des rebonds du signal)
3. Les réseaux réflecteurs (les « transceivers » des sources d'émission transmettent les signaux vers le même point lequel fait office de routeur en les redirigeant vers la source de réception)
4. Les réseaux à liaison optique à large bande (les performances sont comparables à un réseau câblé et permettent de transmettre des fichiers multimédias)

Se type de communication n'a pas été retenu puisque le faisceau ne doit jamais être coupé sinon la transmission est interrompue. Sur une piste, ou les voitures sont en mouvement, se type de communication ne semblait pas idéal.

2.2.1.2 Laser :

Le laser est une technologie semblable à l'infrarouge en ce sens qu'elle nécessite une visibilité directe. Le laser est aussi appelé « la lumière cohérente ».

2.2.1.3 Radio à bande étroite (à fréquence unique) :

La radio à bande étroite (à fréquence unique) fonctionne comme une radio commerciale; il faut régler l'émetteur et le récepteur sur la même fréquence. La radio à bande étroite (à fréquence unique) ne nécessite pas de visibilité directe. La portée de diffusion est importante, mais la vitesse de transmission est faible :

- 1650 mètres
- 4,8 Mb/s

La transmission par les ondes radio requière une licence ou une autorisation de la part des autorités locales.

2.2.1.4 Radio à spectre étalé

La technique de transmission par radio à spectre étalé diffuse des signaux sur une certaine plage de fréquence [6]. La bande passante est divisée en plusieurs canaux de communication. Les cartes réseaux pour « spectre étalé » sont réglées pour une durée prédéterminée sur un des canaux, puis passe sur un autre canal, c'est ce que l'on appelle des sauts de fréquence. Tous les ordinateurs sont synchronisés pour « sauter » en même temps. Pour intercepter de tels signaux, il faut connaître l'algorithme de changement de canal. La vitesse de transmission est faible, mais la portée est importante :

- 25 Kb/s à 4 Mb/s
- 250 mètres à l'intérieur et 3000 mètres à l'extérieur

La technique de transmission par radio à spectre étalé permet de créer un véritable réseau sans fil.

2.2.2 Étude des différents réseaux de communication :

Les types de réseau sont nombreux. Ceux-ci sont décrits brièvement dans cette section.

2.2.2.1 Réseaux PAN (PersonalArea Network)

Les protocoles LAN est le :

- IEEE 802.15 (WiMedia)
- 802.15.1 (Bluetooth)
- 802.15.3 (UWB)
- 802.15.4 (Zegbee)

Caractéristiques des PAN :

- Tout petit réseau permettant d'interconnecter des machines personnelles (PC portable, téléphone mobile, PDA, etc.)
- réseaux sans fil
- technologies émergentes : Bluetooth et Zigbee
- débits : quelques centaines à quelques Mégas bit/s

2.2.2.2 Réseaux LAN (Local Area Network)

Le protocole LAN est le :

- IEEE 802.11

Caractéristiques des LAN

- réseaux adaptés à un site d'entreprise dont la taille ne dépasse pas quelques kilomètres

- réseaux privés
- utilisés pour relier les PC ou les stations de travail à des ressources partagées
- quelques centaines d'ordinateurs
- débits : ~10 ~100 Mbit/s généralement
- repose sur un support partagé nécessitant un mécanisme d'arbitrage pour résoudre les conflits d'accès
- topologies : bus (Ethernet), anneau (Token Ring)

2.2.2.3 Réseaux MAN (Metropolitan Area Network)

Le protocole LAN est le :

- IEEE 802.16 (WiMax)

Caractéristiques des MAN :

- réseaux atteignant la taille d'un campus, d'une métropole
- réseaux publiques ou privées
- essentiellement des gros LAN
- débits : 100 Mbit/s
- utilisent un support de transmission auquel sont reliés tous les ordinateurs
- peuvent servir à interconnecter des LAN
- quelques centaines, quelques milliers de dispositifs
- topologie : double bus (DQDB)

2.2.2.4 Réseaux WAN (WideArea Network)

Le protocole LAN est le :

- IEEE 802.20

Caractéristiques des WAN :

- réseaux étendus sur plusieurs centaines voire des milliers de km (un pays, un continent, ...)
- réseaux publics ou privés
- composés de commutateurs et de liaisons entre eux
- des milliers de dispositifs y sont connectés

Voici quelques caractéristiques des différents réseaux:

Tableau 2-2 Différents types de réseaux

Nom Commun	WIFI			
Norme	802,11a	802,11b	802,11g	802,11n
Type de réseau	LAN	LAN	LAN	LAN
Zone de couverture	15m	30m	30m	plus de 30m
Gamme de fréquence	5 GHz	2,4 GHz	2,4 GHz	2,4 GHz
Débit théorique max.	54 Mbits/s	11 Mbits/s	54 Mbits/s	540 Mbits/s
QoS				
Mobilité				

Nom Commun	WiMAX		Bluetooth	UWB	Zigbee
Norme	802,16d	802,16e	802,15,1	802,15,3a	802,15,4
Type de réseau	MAN	MAN	PAN	PAN	PAN
Zone de couverture	10Km	10Km	Quelques mètres	Quelques mètres	Plusieurs mètres
Gamme de fréquence	2-66 GHz	2,3 GHz	2,4 GHz		2,4GHz
Débit théorique max.	72 Mbits/s	46 Mbits/s	3 Mbits/s		250 Kbits/s
QoS	oui	oui			
Mobilité		oui			

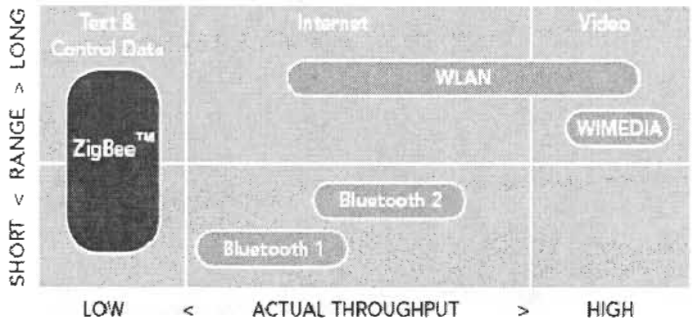


Figure 2.7 Réseaux par rapport à leur portée et leur débit

Il semble évident que nous allons nous diriger vers un réseau sans fil PAN. Dans ces réseaux, deux se distinguent par leur faible coût en consommation de puissance. Le Bluetooth et le Zigbee sont deux bons candidats. Il est donc indispensable de les comparer.

2.2.3 Bluetooth :

Le Bluetooth utilise une technologie radio courte distance. Il a été conçu afin de simplifier les connexions entre les appareils électroniques.

Il utilise la bande de fréquence située à 2.4GHz. Cette bande se nomme ISM (Industrial, Scientific, Medical). La norme Bluetooth définit 79 canaux espacés de 1Mhz chacun, à partir de 2402MHz (2402 - 2480). La modulation utilisée est une modulation en fréquence (dérivation positive = '1', dérivation négative = '0'). Le codage de l'information se fait par sauts de fréquence. La période est de 625 µs, ce qui permet I 600 sauts par seconde. Il existe trois classes de modules radio Bluetooth sur le marché ayant des puissances différentes et donc des portées différentes. Le tableau 2.3 les définies.

Tableau 2-3 Classe Bluetooth

Classe	Puissance	Portée
1	20 dBm (100mW)	100 mètres
2	4 dBm (2.5 mW)	10 mètres
3	0 dBm (1 mW)	1 mètre

Je n'ai pas été plus loin dans mes recherches pour se protocole. Sa complexité et la puissance qu'il demandait ne rencontrent pas les exigences du projet.

2.2.4 Zigbee:

Ce protocole ressemble un peu au Bluetooth. C'est un protocole de haut niveau. Il permet toutefois de communiquer avec une consommation réduite. Il est basé sur la norme IEEE 802.15.4 [7-9]. Cette norme est destinée au réseau à dimension personnelle (Wireless Personal Area Networks : WPANs). Comme le Bluetooth, le Zigbee propose une communication à courte distance. Il a l'avantage d'être moins cher et beaucoup plus simple. Le meilleur exemple est l'espace occupé par le code Zigbee comparativement à celui du Bluetooth [7].

2.2.5 Choix du protocole :

Zigbee ressemble beaucoup à Bluetooth, il est toutefois un peu plus simple, le débit est moindre et il passe une grande partie de son temps en dormance. Un réseau Zigbee peut, par exemple être opérationnel durant une période de 6 à 24 mois avec deux batteries AA.

C'est donc dire que sa consommation est minime. La portée du Zigbee est légèrement supérieure. Elle est d'environ 10 à 75 mètres. Pour le Bluetooth, elle est d'une dizaine de mètres, sans amplification. Un inconvénient important au Zigbee est sa vitesse de transmission. Elle est de 250 Kbps à 2.4 GHz comparativement à 1 Mbps pour le Bluetooth. Le protocole Bluetooth est plus complexe. Par contre, il est plus performant pour transférer de la voix, des images ou encore des fichiers en réseau ad hoc.

Un autre avantage du protocole Zigbee est qu'il prend très peu de temps au démarrage. Lorsqu'on l'allume, le Zigbee peut prendre son premier paquet après 15 msec, comparativement au Bluetooth qui peut prendre environ 3 secondes avant de répondre. De plus, le nombre maximal de personnes connectées au même nœud est de beaucoup supérieur pour le Zigbee. Ainsi, le protocole retenu pour la suite est celui du Zigbee.

2.2.6 Plateformes Zigbee :

Depuis quelques années, plusieurs circuits intégrés sont disponibles [10-19]. On distingue deux types de transceiver Zigbee. Des transceivers intégrés à un processeur, par exemple le CC2430 de Chipcon. Ou encore ceux qui ne le sont pas, par exemple le Xbee de MaxStream. Les deux ont leurs avantages et leurs inconvénients. Voici une liste de composantes qui utilisent le Zigbee :

- CC2430/31
- ETRX1DVK
- MC13191/92/93
- PICDEM Z

- Pixie
- Xbee
- ZS1319x
- ZMD44102

J'ai finalement choisi le modèle CC2430 [10-13] qui est disponible avec une plateforme de développement [13]. J'ai choisi ce modèle, car la littérature sur ce dernier est abondante. De plus, il intègre un processeur d'Intel qui a fait ses preuves.

2.2.7 Communication câblée

1. Le RS232 :

La communication série est l'une des toutes premières. C'est la première qui a permis aux ordinateurs de communiquer avec leurs interfaces (sourie, clavier, imprimante). Il est très facile d'utiliser se type de communication. De plus ces dernières années, le débit qu'elle propose est très intéressant. Un débit souvent utilisé est 125000 bps, mais il est possible d'aller jusqu'à un peu moins de 1 Mbps. Elle nécessite un fil pour l'envoi et un fil pour la réception. L'information se propage de manière série, c'est-à-dire un bit après l'autre. Voici le support physique le plus souvent utilisé. Seulement trois fils sont indispensables. Le Rx (réception), le Tx (envoi) et le gnd(ground).

Figure 2.8 Connecteur normalement utilisé pour le RS-323

Un autre point important du RS232, est qu'il disponible sur la plupart des microcontrôleurs. La communication série ne comprend aucun algorithme pour le contrôle de l'intégrité des données transférées et reçues. Il est aussi impossible de savoir si vous

être bel et bien connecté à moins d'avoir prévu un algorithme à cet escient. Malgré ses inconvénients, la communication RS232 reste l'une des plus utilisées dans le monde, grâce à sa grande simplicité.

Chapitre 3 - Ensemble de développement Zigbee

3.1 Introduction

Comme mentionné plus haut, le projet consiste à faire communiquer des voitures entre elles. Pour ce faire, nous avons créé une plateforme qui simule le comportement de plusieurs voitures sur une autoroute. Cette plateforme à été programmée en C# [4-5]. Comme la simulation est faite sur un ordinateur et que la communication est faite grâce aux kits Zigbee, il est indispensable d'avoir un support pour la communication entre les plaques Zigbee et l'ordinateur. Le choix du type de communication pour se projet est le RS232. Notre choix s'est arrêté sur le RS232 parce qu'il est très simple, flexible et surtout parce qu'il était disponible sur presque tous les microcontrôleurs.

3.2 CC2430DK :

Le Kit de développement inclut un grand nombre d'items [13]. Un avantage considérable est qu'il vient avec des applications qui nous permettent de tester le produit dès la sortie de la boite. Pour chaque bloc du microcontrôleur, il y a un exemple de programme qui nous aide à bien comprendre le fonctionnement et nous donne une idée de comment s'en servir.

Voici la définition des différents termes et composantes du kit :

- SmartRF04EB
    - C'est la plateforme d'évaluation. Elle contient le « LCD », l'interface USB, les LEDs, des potentiomètres, etc.
- SmartRF®04DK
    - C'est le terme qui inclut tout le matériel de l'ensemble de développement.
- CC2430EM
    - C'est un petit circuit qui comprend le CC2430, ainsi qu'une connexion pour l'antenne.
- USB MCU
    - Il utilise le C8051F320 MCU pour avoir l'interface USB sur la carte.
- SoC
    - « System on a Chip ». Ici on fait référence au CC2430.
- ICE
    - « In Circuit Emulator ». Comme le nom le dit, c'est un émulateur.

Le SoC CC2430 est un système très puissant, qui intègre un circuit RF à un processeur. Plusieurs Timers, ADC, DMA, Sleep mode… sont disponibles. De telle sorte qu'ils rendent le système très polyvalent. Voici une liste des différents outils du microcontrôleur.

- Port d'entrées/sortie (I/O PORTS)
- DMA
- Compteur 16-BIT (TIMER (TIMER1))
- Compteur MAC (TIMER (TIMER 2))

- Compteur du repos (SLEEP TIMER)
- Compteur 8-BIT (TIMER) 3 et Horloge (TIMER) 4
- ADC
- Générateur de nombre aléatoire
- Coprocesseur AES
- Détecteur d'alimentation et d'interruption
- « WATCHDOG »
- « TIMERUSART »

Le Kit complet intègre deux SmartRF04EB, deux CC2430EM, ainsi que deux antennes avec leur adaptateur. Voici une description des différentes composantes du smartRF04EB

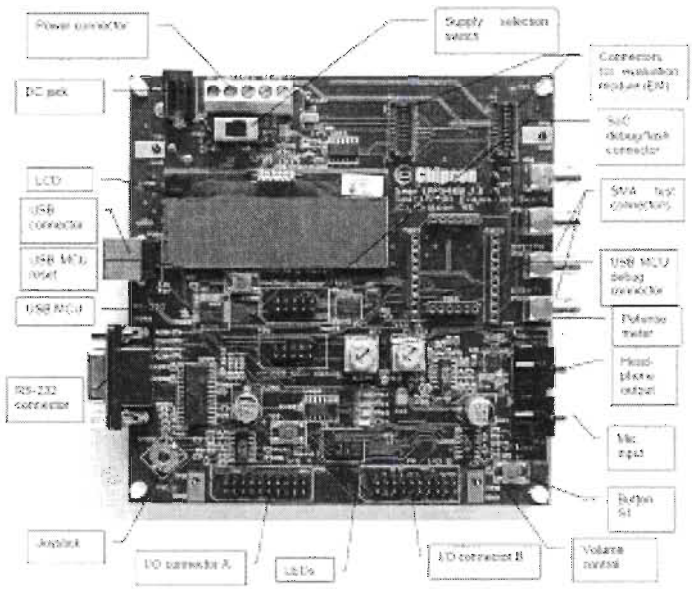


Figure 3.1 SmartRF04EB

Le smartRF04EM contient le cc2430, ainsi que l'antenne. Cette plaquette est le cœur du Kit.

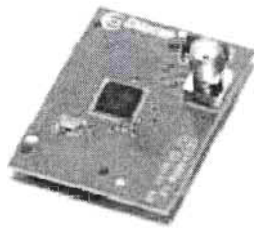


Figure 3.2 SmartRF04EM

3.3 **IAR System :**

Toute la programmation C a été faite avec l'éditeur d'IAR [14]. Mes références en programmation C sont nombreuses. La plus utilisée est le site internet : site du zéro [20]. Nous avons utilisé l'IAR Embedded Workbench. Se dernier comprend l'éditeur et le compilateur. Toutes les librairies nous ont été fournies par IAR, de telle sorte que la programmation à été de beaucoup simplifier. Le kit de développement comprenait le programmeur qui était reconnu automatiquement par le logiciel. De plus, les programmes fournis avec le kit ont été créés à l'aide d'IAR. Aucune modification n'a été nécessaire pour les faire fonctionner. Le programme comprend au total plus de 10 mille lignes de code, sans compter les librairies. Chaque partie du processeur à été testé. De plus, l'interface permet d'utiliser et de tester presque tous ces programmes (fonctions).

3.4 **Programme**

3.4.1 Communication série

Un protocole de communication à été implanté pour la communication UART (RS232). Chaque envoi se fait par bloc. Un bloc d'octets comprend au minimum cinq octets. Voici la signification octet par octet du paquet envoyé.

Octet 1 : 250, valeur fixe. Il définit le début d'un paquet

Octet 2 : Adresse du destinataire ou nom de l'expéditeur (voir 3.4.2 pour plus d'explication sur ses deux adresses)

Octet 3 : 1, 2 ou 10. Il commande l'envoi de données ou l'initialisation de l' adresse. Les commandes 1 et 2 sont pour l'envoi de données et la commande 10 est pour l'initialisation de l'adresse. Il existe donc plusieurs modes. Les neuf premiers (1-9) sont pour l'envoi ou la demande d'information de position, de vitesse et d'accélération (1 : Envoi, 2 : Demande). Par exemple en mode 2, je demande à une autre voiture ses données de position, de vitesse et d'accélération. En mode 1, c'est l'inverse, j'envoie ces informations. Les modes 10 à 19 sont des modes qui portent sur l'initialisation de la plateforme Zigbee. Par exemple, le mode 10 donne l'adresse.

Octet 4 : Longueur du paquet à envoyer

Octet 5 et les suivants : Les données à transmettre

Dernier Octet : « Check somme » : La somme de tous les octets doit être égale à 0. Ce denier octet sert donc à respecter cette consigne. Une fois l'acquisition des données terminées, la sommation est effectuée. Si le résultat est différent de 0, une erreur de transmission à eu lieu. Cet octet sert donc à valider l'information transmise et à s'assurer qu'il n'y a aucune erreur dans les données.

Par exemple, pour un mode 10, les octets envoyés de l'ordinateur à la plateforme Zigbee sont (hexadécimal) : FA 01 0A 00 FB.

3.4.2 Communication zigbee :

La communication Zigbee se concentre autour de deux fonctions :

1. *radioSend*(*sendBuffer*, *longeur*, *adrDenvoi*, *ACK*)
    a. *sendBuffer* est le paquet à transmettre.
    b. longueur est la longueur du paquet à transmettre.
    c. *adrDenvoi* est l'adresse de destination.
    d. *ACK* est un nombre boolien. S'il est égal à 0, c'est que nous ne voulons pas que le destinataire nous réponde. S'il est égal à 1, nous voulons une réponse afin de s'assurer que la communication est bonne.
2. *radioReceive*(*&receiveBuffer*, *&length*, *RECEIVE_TIMEOUT*, *&sender*)
    a. *&receive* Buffer est l'adresse du « buffer » où les données sont écrites.
    b. *&length* est l'adresse de la variable où la longueur du paquet reçu est écrite.
    c. *RECEIVE_TIMEOUT* est le temps d'attende. Nous avons utilisé 50 ms.
    d. &sender est l'adresse de la variable où est écrite l'adresse de la plateforme Zigbee qui nous a envoyé la demande.

Les données envoyées à une autre plateforme sont exactement les mêmes que celles reçues de l'UART, à l'exception de l'octet d'adresse qui est changée par l'adresse de l'expéditeur. Il est important de changer cette valeur avant de transmettre le paquet. Pour bien comprendre, lorsque l'ordinateur envoie un paquet à transmettre d'une plateforme à une autre, il écrit dans le deuxième octet l'adresse du destinataire. Une fois que la plateforme a reçu le paquet à envoyer, il doit changer le deuxième octet par son adresse. Ainsi, la plateforme qui reçoit le paquet n'a aucune modification à faire et l'envoie

directement à l'ordinateur. Ce dernier n'a qu'à lire le deuxième octet pour savoir de qui ce paquet provient.

3.4.3 Intégration de l'UART et du Zigbee :

En fait, l'idée était de faire un convertisseur RS232 vers Zigbee, Zigbee vers RS232. Les schémas 3.3 et 3.4 illustrent bien l'algorithme que nous avons développé. Premièrement, il est indispensable d'initialiser la communication UART. Le premier envoi de données (de l'ordinateur vers le cc2430) initialise l'adresse et le mode de transmission RF (voir le 2^{e} schéma et ses explications).

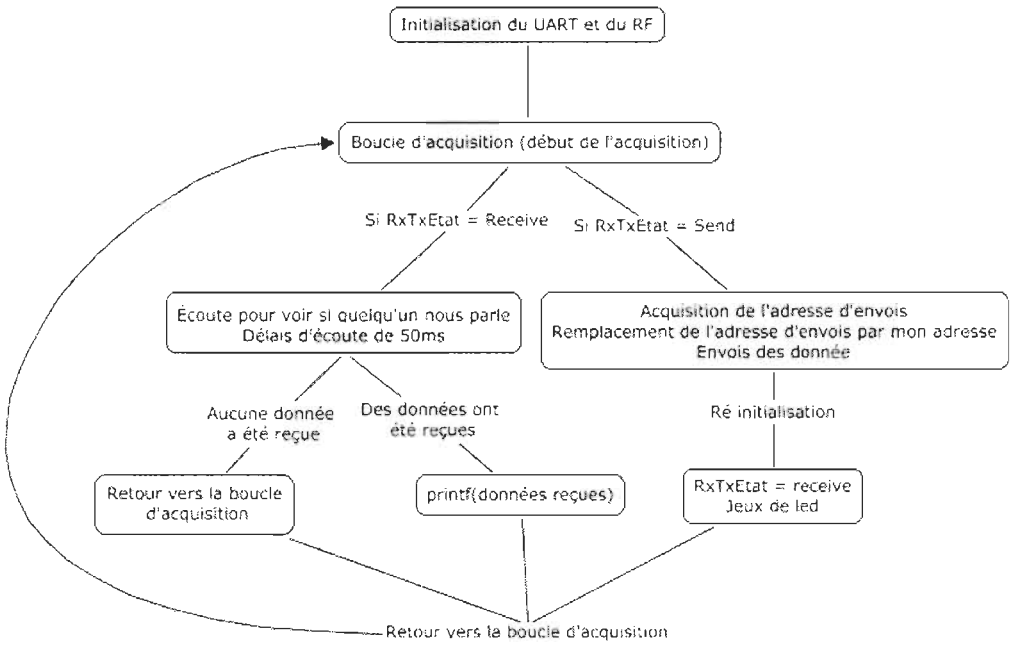


Figure 3.3 Convertisseur RS232 vers Zigbee, Zigbee vers RS232 (1-2)

Premièrement, en mode *Receive* (qui est le mode par défaut), la plateforme se met en mode attente. Elle attend qu'une autre plateforme entre en communication avec elle. Si après 50ms personne n'a tenté de communiquer avec la plateforme, elle retourne au début

de la boucle afin de voir si le mode a changé. Par contre, si des données ont été reçues, la plateforme doit les transmettre à l'ordinateur en utilisant le *printf*.

En mode *Send*, des données ont été reçues de l'ordinateur et doivent être transmises vers une autre plateforme. L'ordinateur communique avec la plateforme à l'aide du port Uart (voir la figure 3.3 pour plus d'explication sur l'acquisition des données UART). La section 3.4.1 explique le protocole de communication que nous avons implanté. L'adresse du destinataire est incluse dans le paquet.

Le schéma est une boucle sans fin. Il est toutefois possible d'arrêter le programme en utilisant le « Joystick ». Plus d'explication vous sera donnée dans la section 3.4.4.

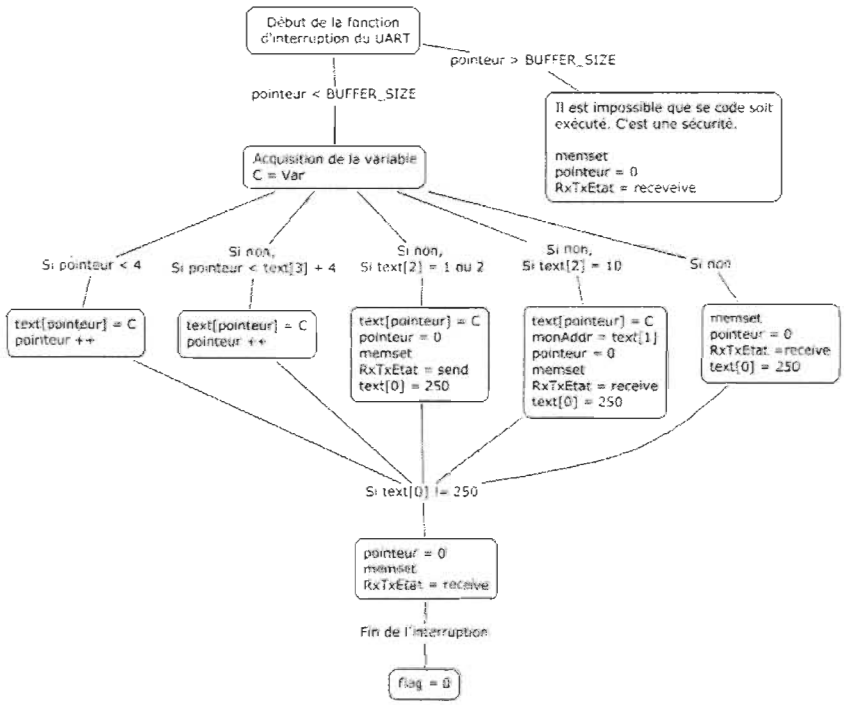


Figure 3.4 Convertisseur RS232 vers Zigbee, Zigbee vers RS232 (2-2)

Le schéma 3.4 illustre la logique utilisée lors d'une interruption due à un nouvel octet disponible à l'UART. Comme mentionné plus haut, chaque paquet contient la quantité

d'octets à transmettre. Ces informations se trouvent dans les octets trois et quatre. Il est nécessaire d'avoir au moins quatre octets acquis avant de pouvoir faire quoi que ce soit. Une fois le quatrième octet acquit, il reste (text[3] + 4) octets toujours manquants. Lorsque toutes les données ont été acquises, une vérification est faite afin de s'assurer que le premier octet vaut bien 250. Si ce n'est pas le cas, une réinitialisation est faite, un paquet est perdu, mais aucune donnée erronée n'est envoyée. Le test d'intégrité des données n'est pas fait ici. Il est fait dans l'algorithme de l'ordinateur.

3.4.4 Affichage (LCD), contrôle (joystick) et pointeur de fonction

Afin de simplifier l'utilisation de l'ensemble de développement, une interface a été développée. L'utilisateur peut compter sur un écran « LCD » et sur un « joystick » pour choisir et initialiser le programme qu'il veut utiliser.

Voici le nœud principal du programme :

```
for(;;)
{
  if(checkForInput())
    updateMenu();
}
```

C'est une boucle sans fin. Premièrement, nous devons vérifier si le « joystick » a été bougé. Pour ce faire, nous utilisons la fonction *checkForInput*(). Cette fonction est expliquée un peu plus bas. Si le « joystick » a été bougé, nous appelons la fonction *updateMenu*() qui comme son nom l'indique fait une mise à jour du menu affiché.

La figure 3.5 illustre le fonctionnement de la fonction *checkForInput*().

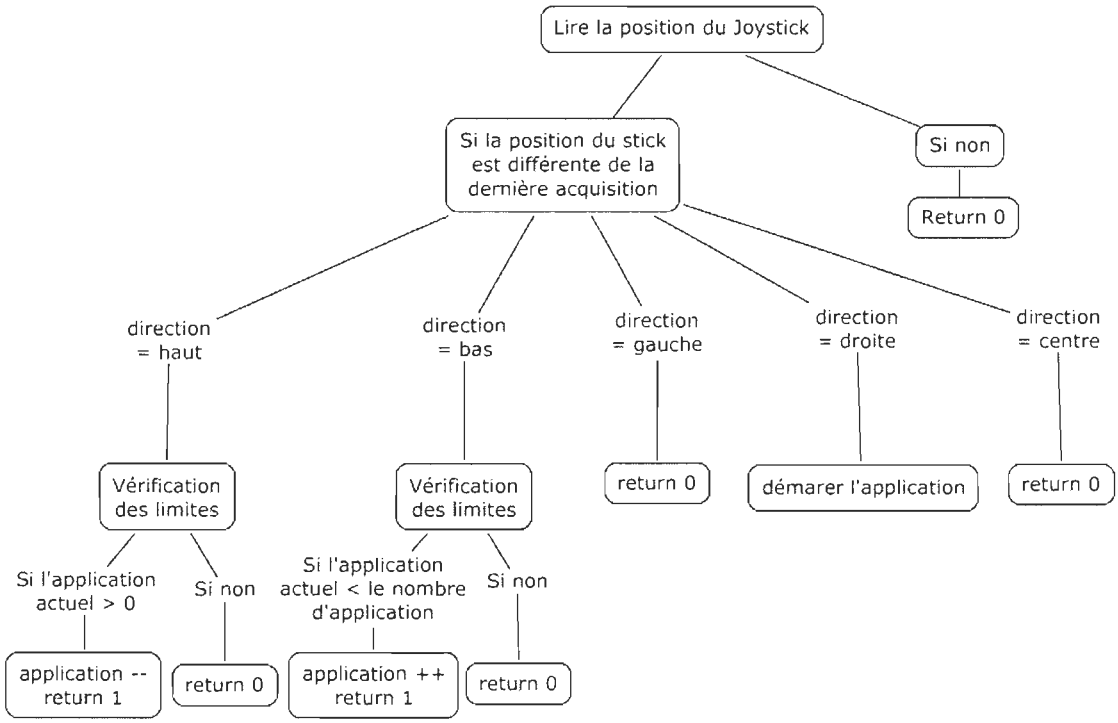


Figure 3.5 Utilisation du «joystick»

L'utilisateur peut visualiser toutes les applications simplement en déplaçant le «joystick» vers le bas ou vers le haut. Ainsi le nom des applications s'affichera tour à tour sur l'écran « LCD ». Une fois que son choix est fait, il doit déplacer le «joystick» vers la droite. Ainsi l'application débutera. Donc si l'utilisateur se contente de bouger le manche vers le haut et que l'application actuelle n'est pas 0, le compteur d'application se décrémente et la fonction retourne 1. La figure 3.6 illustre la fonction *updateMenu*().

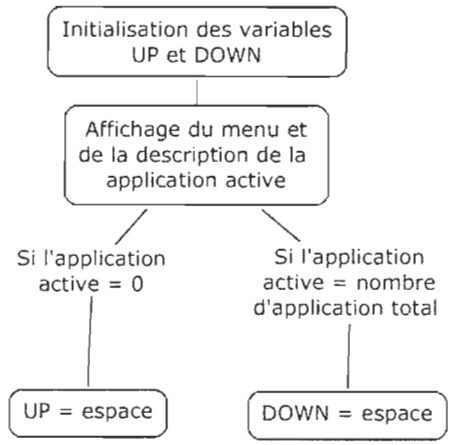


Figure 3.6 Affichage (LCD)

Voici le code utilisé :

```
char up = ARROW_UP;
char down = ARROW_DOWN;
lcdUpdate(apps[activeApp].menuText, apps[activeApp].description);

if (activeApp == 0){
  up = ' ';
}
else if (activeApp == NBR_OF_APPS - 1){
  down = ' ';
}
lcdUpdateChar(LINE1, LINE_SIZE - 1, up);
lcdUpdateChar(LINE2, LINE_SIZE - 1, down);
```

Si l'utilisateur à fait son choix et qu'il déplace le manche vers la droit, la fonction startApplication() sera appelée :

```
void startApplication(void)
{
  DISABLE_ALL_INTERRUPTS();
  SET_MAIN_CLOCK_SOURCE(CRYSTAL);
```

```
    apps[activeApp].main_func();

    CLR_GLED();
    CLR_YLED();

    DISABLE_ALL_INTERRUPTS();
  }
```

Cette fonction est fort simple. Elle s'assure que toutes les interruptions sont désactivées et elle appelle la fonction.

3.4.4.1 Structure et fonction de base :

Afin de réaliser cet algorithme, il a fallu créer une structure. Cette structure est au cœur du fonctionnement de ce programme. Son nom est App (pour Application). Voici sa définition :

```
typedef struct APP_S{
  char *menuText;
  char *description;
  void (*main_func)(void);
  void (*interrupts[NBR_OF_INTERRUPTS])(void);
} APPLICATION;
```

La structure contient quatre variables. La première sera affichée sur l'écran «LCD». C'est tout simplement le nom de la fonction. La deuxième sera aussi affichée sur l'écran. C'est une brève description de la fonction. Elle sera affichée sur la deuxième ligne de l'écran. La troisième valeur donne l'adresse de la fonction et la dernière donne l'adresse de l'interruption. Voici un exemple d'initialisation de fonction, précédé par l'appel de toutes les initialisations.

```
adc_init(&apps[ADC_CONV]);
adc_series_init(&apps[ADC_SERIES]);
```

```
stop_watch_init(&apps[STOP_WATCH]);
uart_init(&apps[UART]);
clockmodes_init(&apps[CLOCKMODES]);
rf_test_init(&apps[RF_TEST]);
...
```

Chacune de ces fonctions initialise une application. L'exemple suivant initialise l'UART.

```
void uart_init(APPLICATION *a)
{
  a->menuText = (char*)"UART <--> LCD";
  a->description = (char*)"57600 8-N-1";
  a->main_func = uart_main;
  a->interrupts[INUM_URX0] = int_UART_RX_IRQ;
}
```

Chapitre 4 - Simulateur du convoi

4.1 Introduction

Une plateforme qui simule le comportement de plusieurs voitures sur une autoroute a été réalisée dans le cadre de ce travail. Le langage C# a été utilisé [4-5]. L'ensemble de la plateforme a nécessité environ 1000 lignes de code. Nous allons décrire, dans ce chapitre, les composantes et le mode de fonctionnement de cette plateforme.

4.2 Affichage et interface utilisateur

L'utilisateur peut compter sur un programme simple à utiliser. La figure suivante illustre l'interface. Cette dernière permet d'initialiser et d'afficher toutes les données des voitures, d'ajouter et de retirer des voitures, d'afficher le déplacement des voitures lors de la simulation, de « zoomer » et de suivre une voiture en particulier, de démarrer et d'arrêter la simulation. De plus, il est possible d'enregistrer toutes les données des voitures, ainsi il est possible de simuler plusieurs fois avec les mêmes configurations. L'affichage est fait en 3 dimensions, de tel sorte qu'il est possible de placer la caméra n' importe où dans la scène, même à l'intérieur d'une voiture. Lors de la simulation, toutes les données sont mises à jour en temps réel. Il sera toutes fois possible pour l'utilisateur de changer certaines valeurs, tout dépendant du mode qu'il a choisi pour la voiture. Par exemple, pour une voiture de tête, il est possible de changer la vitesse de croisière. Ainsi, la voiture accélérera ou décélérera en fonction de ce que l'utilisateur désire.

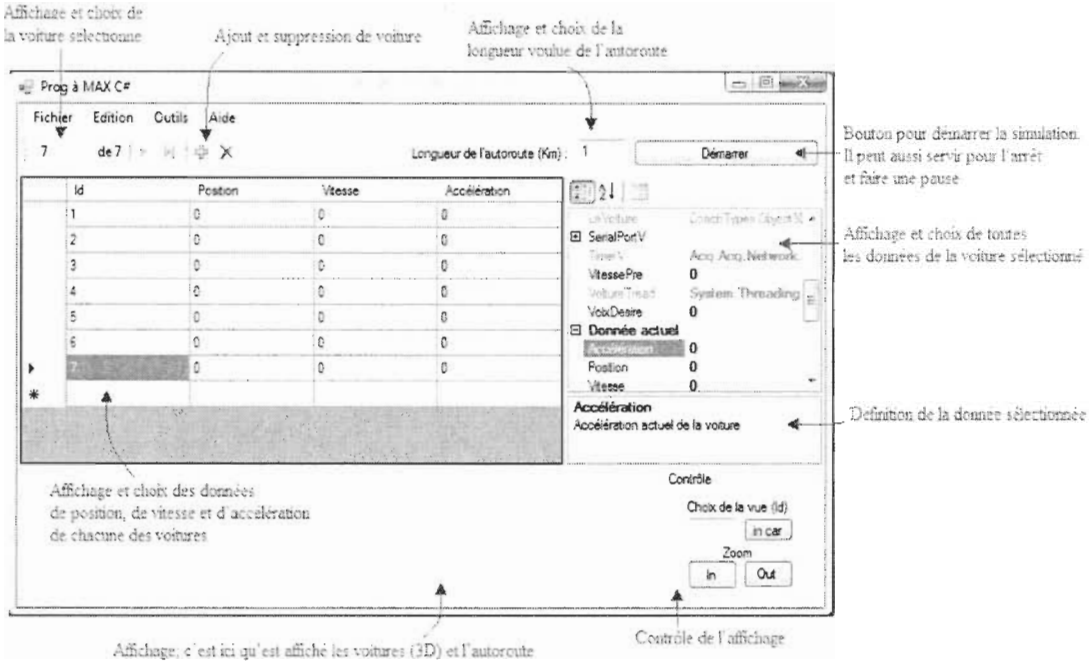


Figure 4.1 Interface utilisateur

4.3 **Caractéristiques d'une voiture :**

Chaque voiture contient des dizaines de variables. Elles sont utilisées pour contrôler la voiture, pour l'identifier et pour avoir une bonne idée de son comportement lors de la simulation. Voici une liste de ces variables.

- Identification :
    - Modèle : le modèle de la voiture
    - Numéro : Numéro d'identification de la voiture
    - Plaque d'immatriculation
    - Propriétaire
- Divers :
    - DistancePre : Distance qui sépara la voiture avec celle qui la précède

- ErreurPre : Erreur de positionnement entre la position désirée et la voiture qui la précède
- ErreurVitessePre : Erreur de vitesse de la dernière itération
- ErreurVoixPre : Erreur entre la position que la voiture occupe sur la route (position latérale) et la position désirée
- LaVoiture : Fichier .x (modèle 3d de la voiture)
- SerialPortV : Configuration de la communication série
- TimerV
- VitessePre : Vitesse de la voiture lors de la dernière acquisition
- VoitureTread : « Thread » particulier de la voiture
- VoixDesire : La voix (route) désirée

- Donnée actuelle :
  - Accélération
  - Position
  - Vitesse
  - Vitesse de croisière
  - Voie
- Statut :
  - Groupe
  - Statut : Voiture active ou inactive
  - Train : Si la voiture veut faire partie d'un convoi ou pas
- Uart :
  - Communication série

- Numéros de port

Affin de simuler le déplacement de chaque voiture en temps réel, un « thread » est nécessaire. Chaque voiture à son propre « thread ». Une autre solution aurait été possible, nous aurions pu simuler une après l'autre les voitures, par contre le temps réel aurait été plus difficile à atteindre. La figure qui suit illustre l'algorithme général d'un «thread» de voiture.

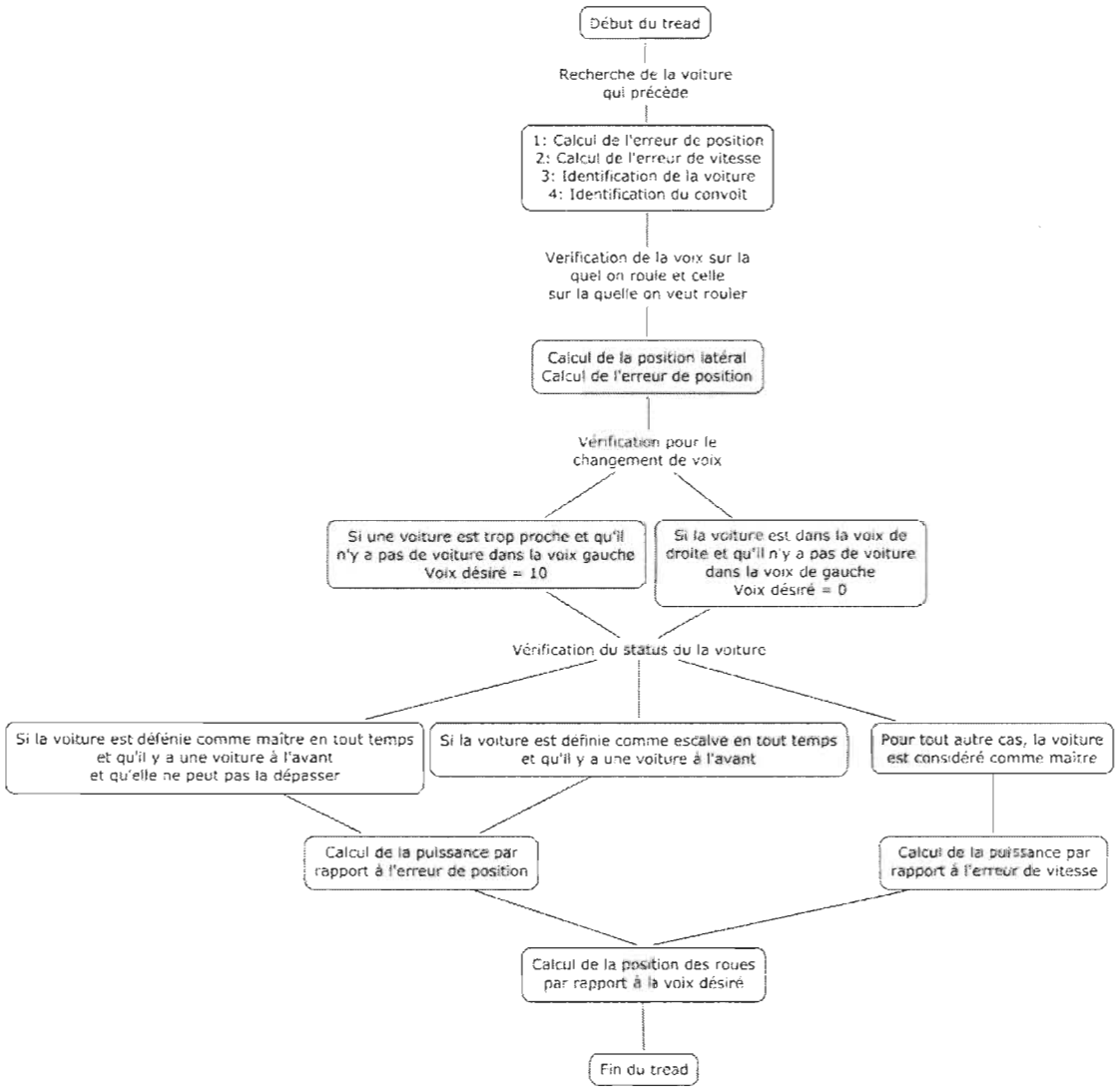


Figure 4.2 Algorithme général de la voiture (thread)

Deux parties importantes manquent à ce schéma; les contrôleurs et la modélisation mathématique du comportement des voitures.

4.3.1 Modélisation des voitures et des contrôleurs

La puissance demandée aux voitures vient des contrôleurs. Le modèle réagit à cette demande. Le modèle utilisé est très simple. Afin de simuler le déplacement des voitures et des contrôles, il a fallu simuler la réponse des voitures à une fréquence plus élevée que celle des contrôleurs. Une fréquence 10 fois plus élevée a été utilisée. Voici le modèle de base des voitures :

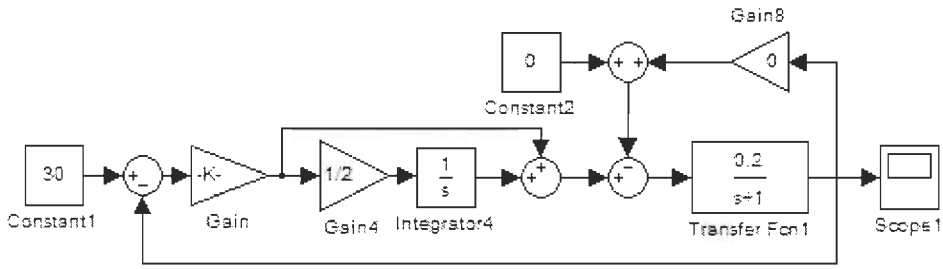


Figure 4.3 Modèle de base d'une voiture

Différents contrôleurs ont été utilisés. Le P et le PI pour le contrôle de la vitesse et de la position sur la voie, le PID pour le contrôleur de position. La logique floue à été utilisé dans différentes conditions. La figure 4.3 est un contrôleur de vitesse. Il est utilisé pour les voitures de tête. Il asservit la vitesse en utilisant la différence entre la vitesse de la voiture et la vitesse désirée. Dans ce schéma la vitesse désirée est de 30km/h.

Deux modèles de voitures ont été utilisés afin de tester les contrôleurs. Le modèle masse-amortisseur et le modèle du moteur électrique. Finalement, le modèle choisi pour

le programme est le système masse-amortisseur. La stabilité de ce modèle et la facilité à le contrôler rendent la simulation (la partie visuelle) d'une très grande réalité. Le modèle du moteur électrique (voir chapitre 2) à aussi été utilisé, mais malgré sa plus grande complexité, il n'a pas rendu la simulation plus réelle (ici je fais allusion à l'impression visuelle). De plus, il rendait la simulation un peu moins stable. Puisque le but final n'était pas de parfaitement simuler la voiture, j'ai opté pour la simplicité.

Les contrôleurs de type P, PI et PID se passent d'explication. Pour avoir plus d'explication sur c'est contrôleur, référez-vous au chapitre 2. Par contre, la logique floue utilisée demande des explications.

4.3.2 Logique floue sur les voitures :

Plusieurs configurations ont été utilisées. La logique utilisée entre les entrées est le ET (voir chapitre 2). Celle utilisée entre les règles est le OU. Comme mentionnés plus haut, trois types de logiques floues ont été réalisés; une pour le contrôle de la vitesse (voiture de tête), une pour le contrôle de la position (entre les voitures) et une autre pour le contrôle de la position sur la voie (voie de gauche ou voie de droite). Voici l'algorithme utilisé pour la logique floue.

4.3.2.1 Contrôle de la vitesse :

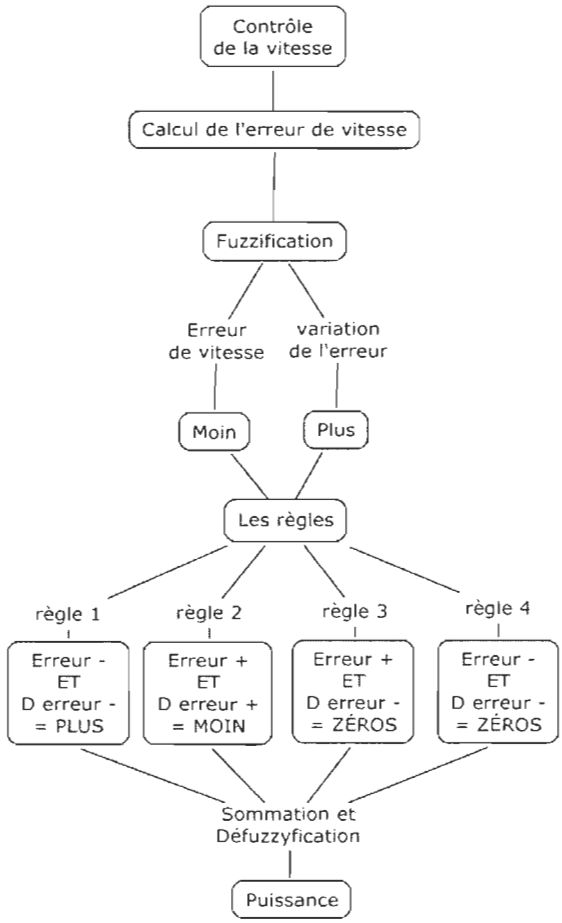


Figure 4.4 Contrôle de la vitesse (logique floue)

Le contrôle de la vitesse se fait exactement de la même façon que l'exemple du chapitre deux. La fuzzification se fait sur deux variables; l'erreur de vitesse (vitesse de croisière – vitesse réelle) et la variation de cette erreur (Err(z) – Err(z-1)). Cette fuzzification se fait grâce aux courbes de la figure 4.5.

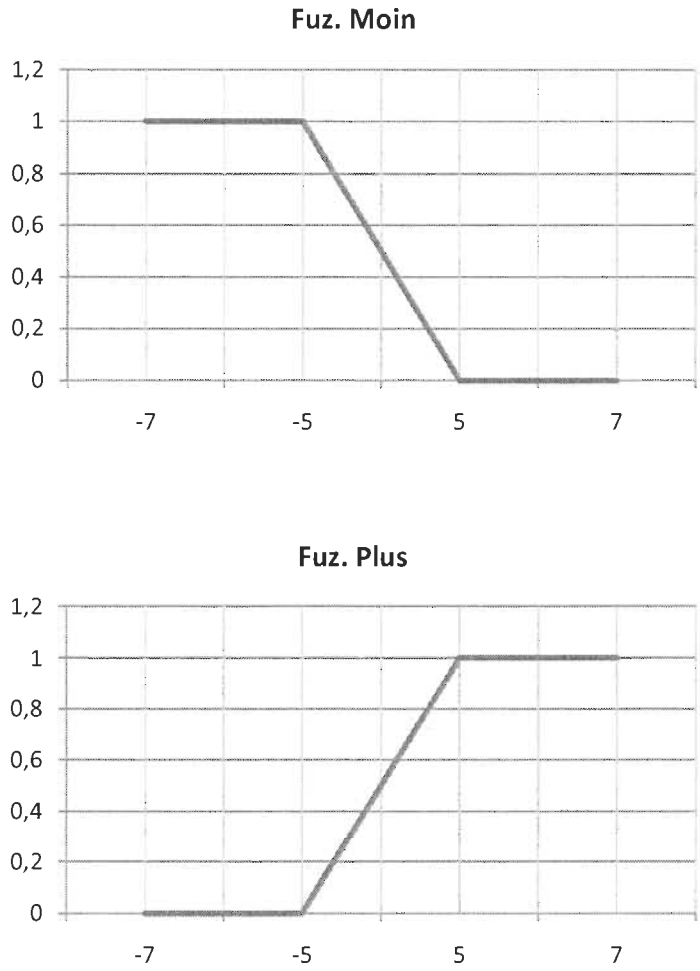


Figure 4.5 Courbes d'appartenance d'entrée (fuzzification)

La logique ET a été utilisée entre les variables de telle sorte que la plus petite des deux valeurs fuzzifiées est gardée.

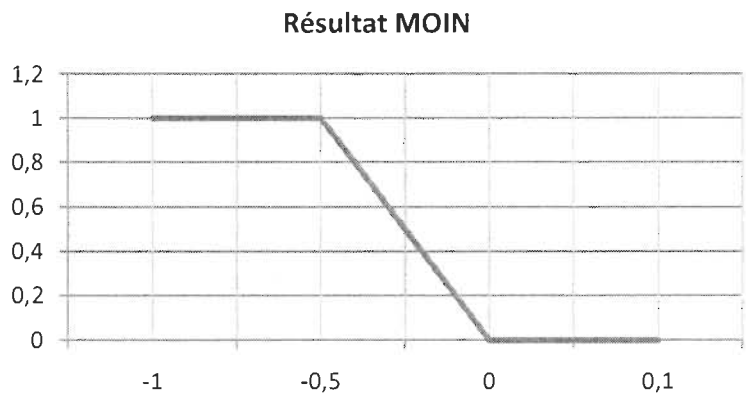
Quatre règles sont nécessaires

Tableau 4-1 Lois de commande

Règle	Erreur	Variation de l'erreur	Résultat
1	MOINS	MOINS	PLUS
2	PLUS	PLUS	MOIN
3	MOIN	PLUS	ZEROS
4	PLUS	MOIN	ZEROS

La logique qui unit ces quatre règles est le OU, de telle sorte qu'il ne reste que les maximums.

Voici les graphiques qui représentent les résultats.



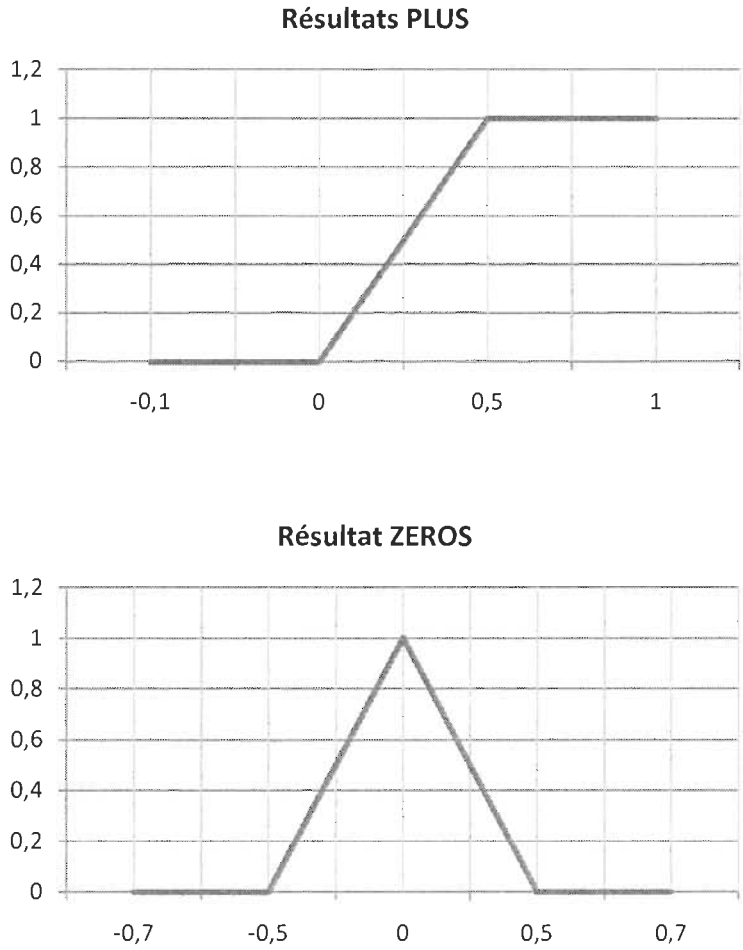


Figure 4.6 Courbes d'appartenance de sortie

4.3.2.2 Contrôle de la position

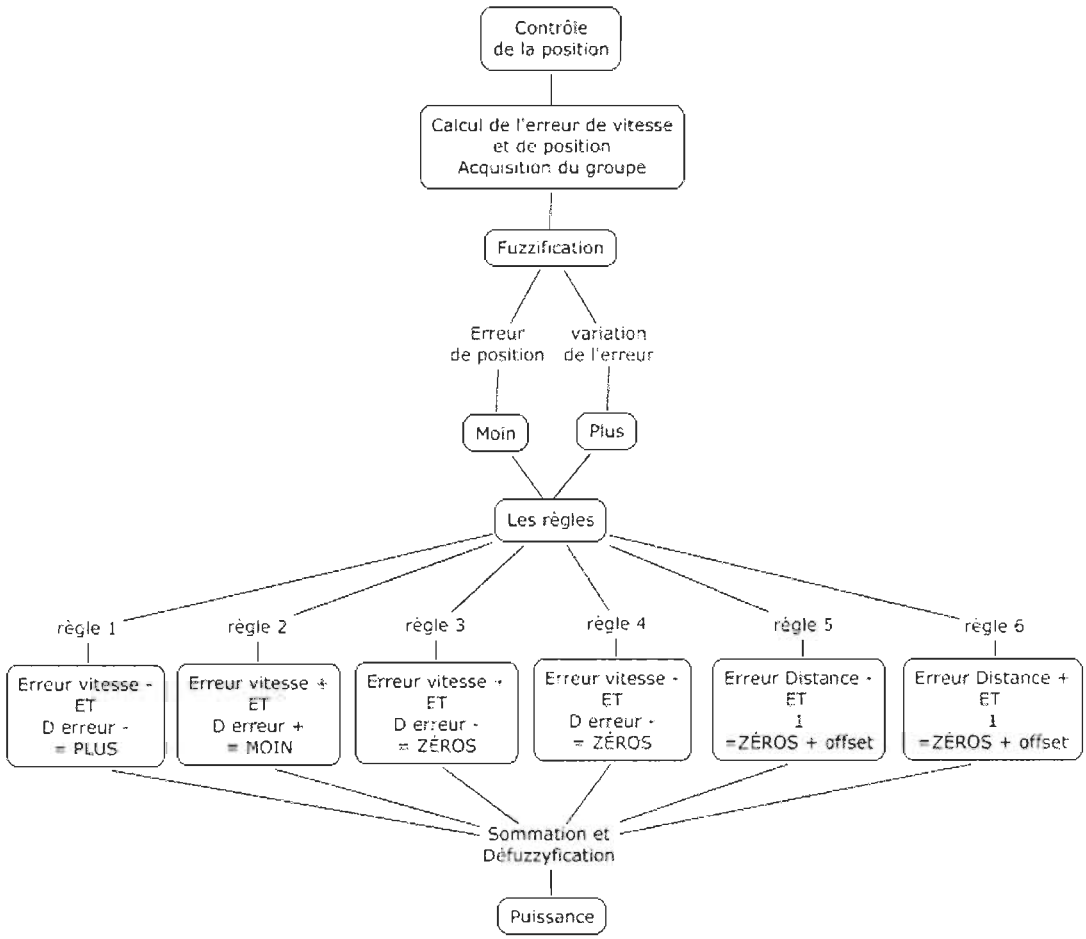


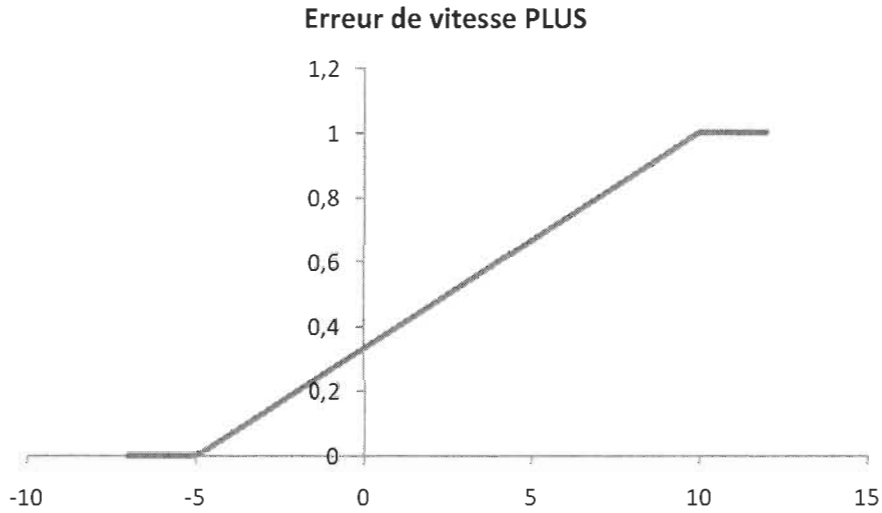
Figure 4.7 Contrôle de la position (logique floue)

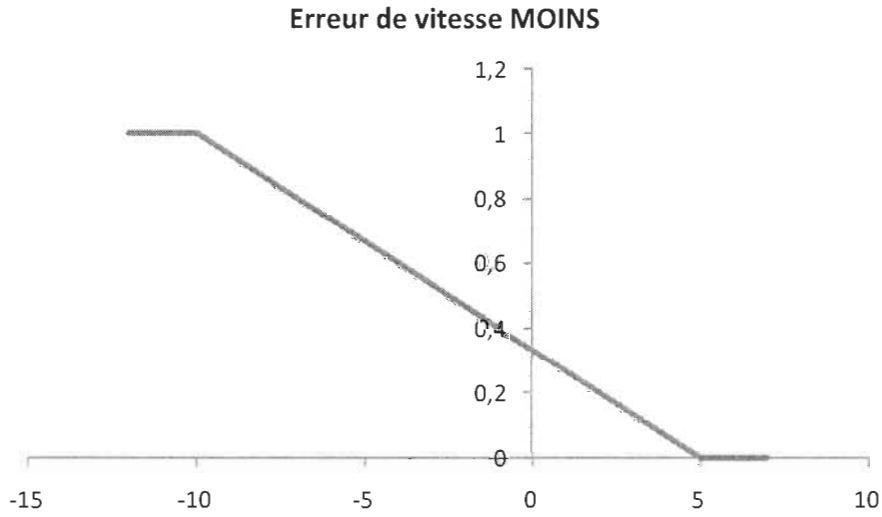
Le contrôleur de position est un peu différent du contrôleur de vitesse. Les variables « fuzzifiées » sont les suivantes :

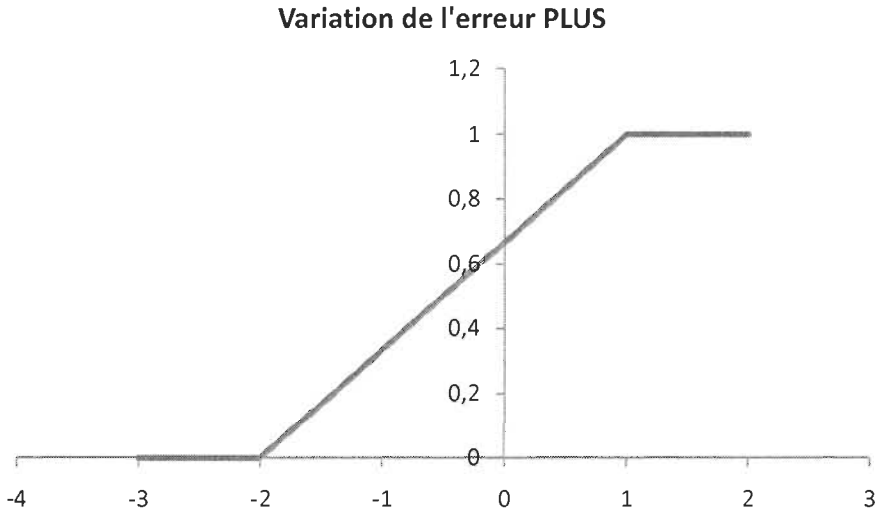
1. Différence entre la vitesse de la voiture et la vitesse de la voiture qui est suivie (Erreur vitesse)
2. Variation de cette vitesse (D erreur)

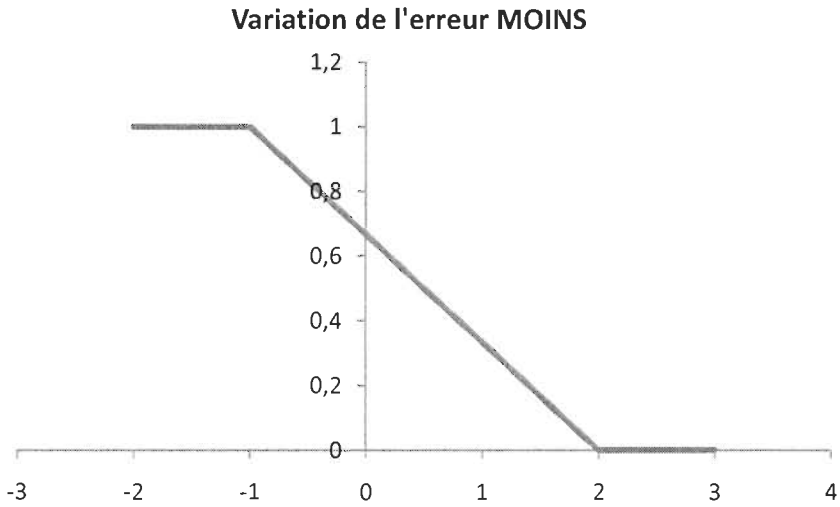
3. Distance qui sépare les voitures moins la distance désirée entre ces deux voitures (Erreur Distance)

Les courbes de fuzzification sont les suivantes :






Variation de l'erreur PLUS
1,2
1
0,8
0,6
0,4
0,2
0
-4
-3
-2
-1
0
1
2
3


Variation de l'erreur MOINS
1,2
1
0,8
0,6
0,4
0,2
0
-3
-2
-1
0
1
2
3
4

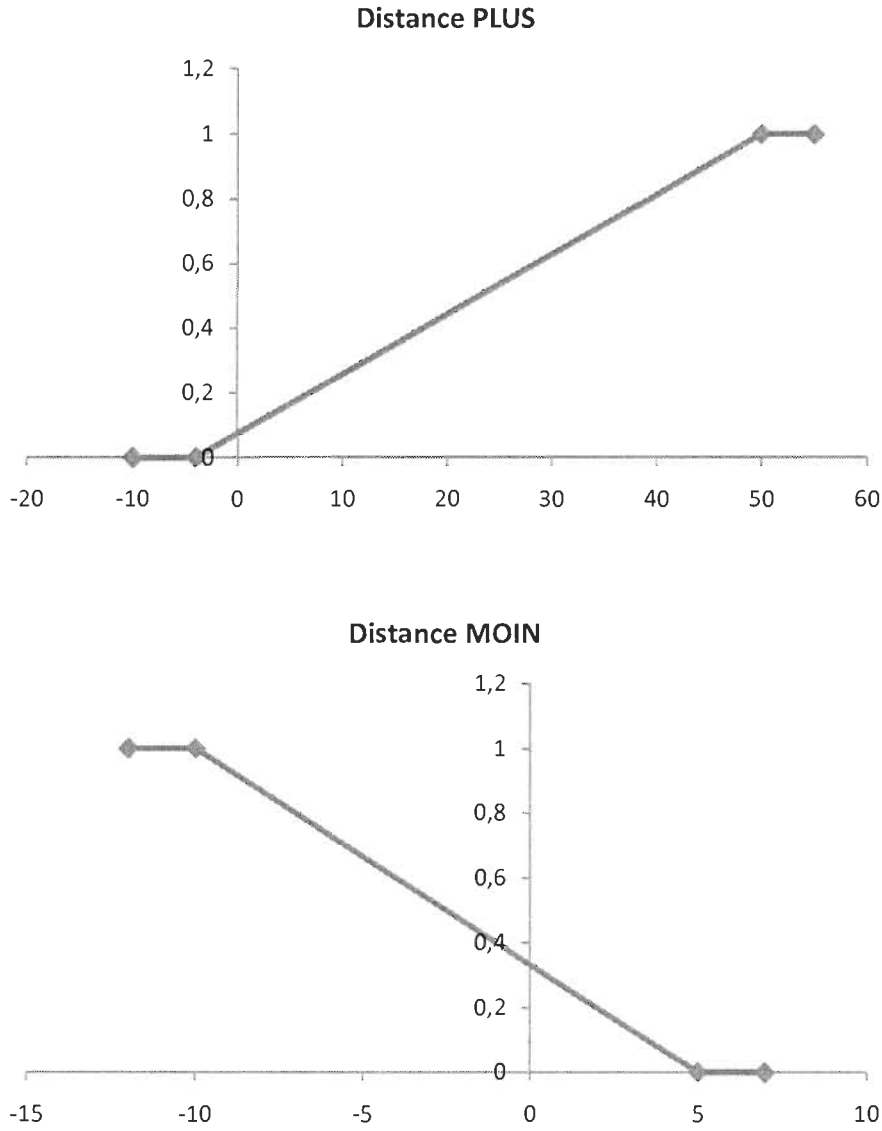


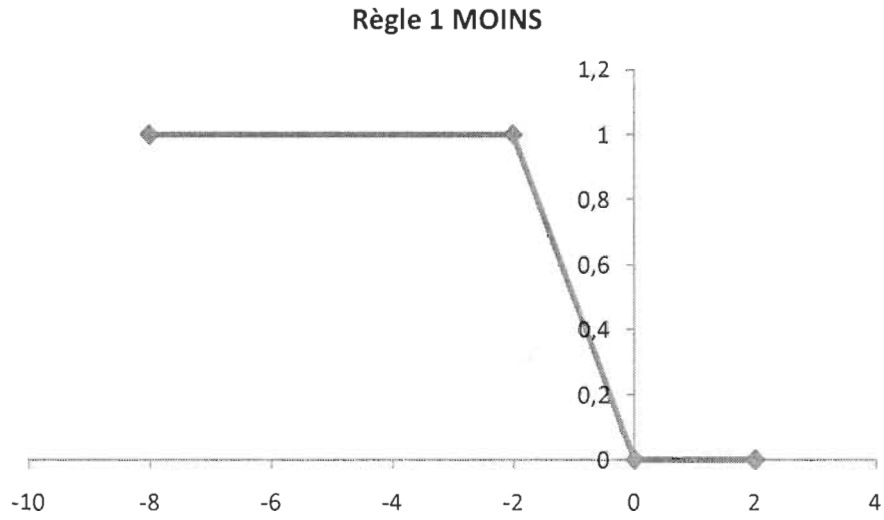
Figure 4.8 Courbes d'appartenance d'entrée (fuzzification) pour le contrôle de la position.

Les règles qui régissent ces courbes sont présentées au tableau 4.2:

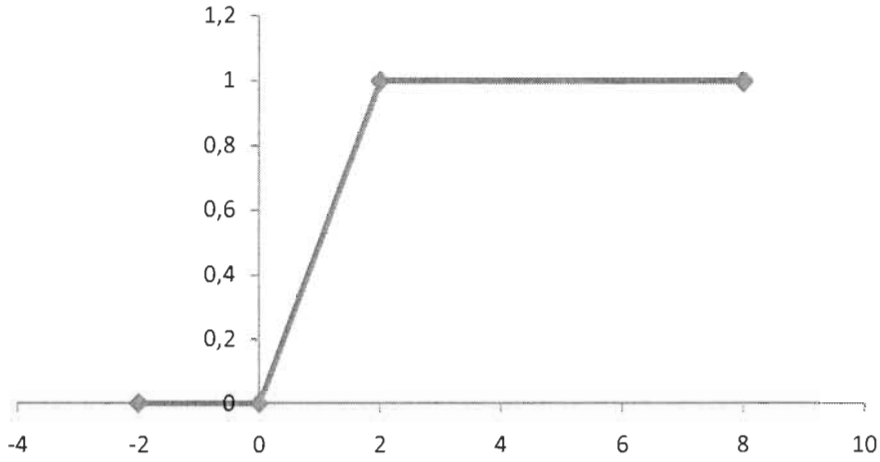
Tableau 4-2 Règles floues pour le contrôle de la position.

Règle	Erreur de vitesse par rapport à la voiture d'en face	Variation de l'erreur de vitesse	Erreur de position	Résultat
1	MOINS	MOINS	x	MOINS
2	PLUS	PLUS	x	PLUS
3	PLUS	MOINS	x	ZEROS
4	MOIN	PLUS	x	ZEROS
5	x	x	MOINS	ZEROS
6	x	x	PLUS	ZEROS

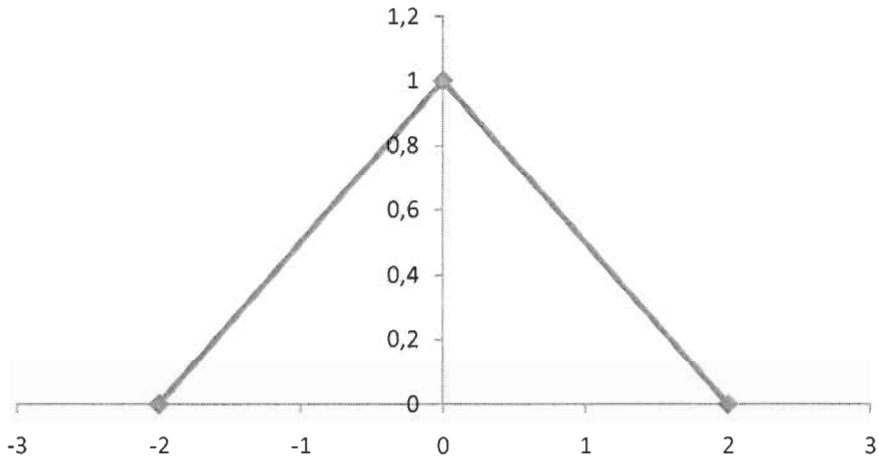
Les résultats sont illustrés sur la figure 4.9:



Règle 2 PLUS



Règle 3 et 4 ZEROS



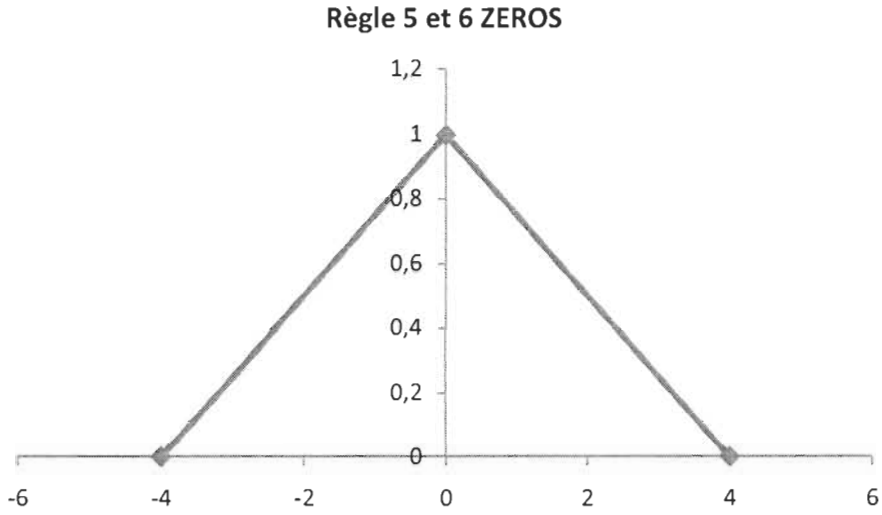


Figure 4.9 Courbes d'appartenance de sortie pour le contrôle de la position.

Les courbes 3 et 4 sont exactement les même. Les courbes 5 et 6 sont toutes fois très différentes. Un décalage (offset) est ajouté au centre de gravité.

Pour la courbe 5 le décalage est de :

$$-25*\text{Vitesse} / 27$$

Pour la courbe 6 :

$$4 * \text{Vitesse} / 27$$

Ce décalage permet de bien calibrer la vitesse de réaction.

4.3.2.3 Contrôle de la voie :

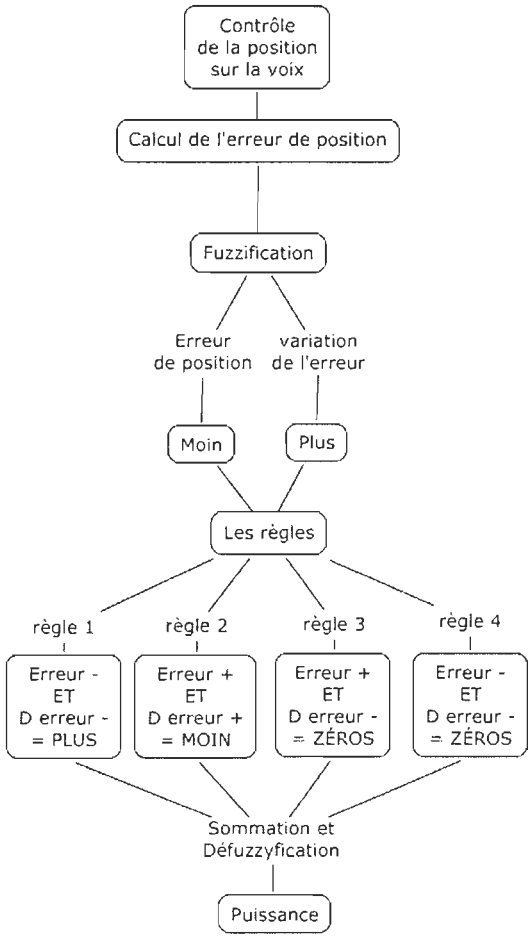


Figure 4.10 Contrôle de la voie

Ce contrôle est exactement le même que celui du contrôle de la vitesse. Ici afin de modéliser le changement de voie, le modèle d'une masse en accélération a été utilisé. Évidemment, ce modèle est moins réaliste. Afin de rendre la simulation plus réelle, il a été nécessaire de changer l'angle de la voiture sur la route. De telle sorte que si la voiture tourne vers la gauche, la voiture aura un angle positif par rapport à la route. Le calcul de l'angle est fort simple.

$$\theta = \frac{V_y}{V_x} * 90^{\circ}$$

Cette simplification nous a permis d'avoir une simulation proche de la réalité, tout en simplifiant les calculs. Il est toutefois impossible d'utiliser ce contrôleur dans la réalité.

4.3.2.4 Les threads :

Chaque voiture à son propre «thread». Un «thread» est une partie du code qui s'exécute en « parallèle » avec les autres «threads». L'idée est de rendre les voitures indépendantes les unes des autres. Alors si une voiture attend une information critique et qu'elle ne peut rien faire avant d'avoir cette donnée, la simulation continue pour les autres et les résultats ne sont pas affectés par ce délai. Ces «threads» (celles des voitures) s'occupent de l'acquisition des données, de la logique de contrôle et des contrôleurs. Ces «threads» s'exécutent à toutes les 100ms. Afin de mettre à jour la position, la vitesse et l'accélération, un autre « thread » a été utilisé. Ce dernier met à jour toutes ces données pour toutes les voitures. Ce « thread » s'exécute à toutes les 10 ms, donc à 100Hz. Il est nécessaire d'avoir une exécution beaucoup plus rapide, car puisque les «threads» des voitures ne sont pas cadencés, nous aurions l'impression que les voitures oscillent lors de l'affichage.

4.4 Convoi :

L'algorithme de la formation des convois est illustré à la figure 4.11. Chaque voiture a la possibilité d'entrer dans un convoi à la première occasion ou de rester en tout temps la voiture de tête. La variable « Train » donne l'état de la voiture. Lorsqu'elle est initialisée à VRAI, la voiture entrera dans un groupe à la première occasion. Elle suivra le groupe en gardant une vitesse de consigne, et une distance de 10 mètres entre elle et la voiture qu'elle

suit. Afin de savoir dans quel groupe la voiture est, elle doit demander à la voiture qu'elle suit le groupe qu'elle occupe. Le numéro du groupe est l'identité de la voiture de tête, de telle sorte que si une voiture veut contacter la voiture maîtresse, elle n'a qu'à utiliser le numéro de son groupe.

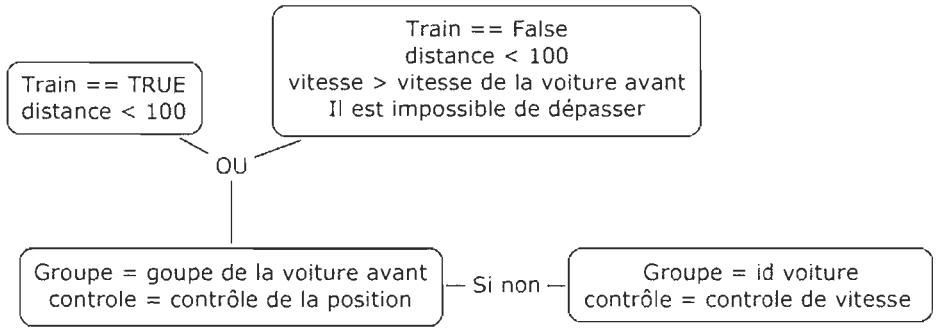


Figure 4.11 Logique du choix du contrôleur dans le cas d'un convoi

4.5 **Communication série :**

La communication série est l'une des parties importantes pour ce projet. Chaque voiture a une variable qui se nomme « uartCom ». Cette variable booléenne est initialisée par l'utilisateur. Lorsqu'elle est Vrai, la voiture utilise la communication série afin de communiquer avec les autres voitures. Évidemment, la voiture avec laquelle elle veut communiquer doit aussi utiliser ce type de communication pour que ça fonctionne. Donc, l'utilisateur doit initialiser uartCom, ainsi que le port de communication qu'elle utilise. Le port est défini dans la variable « comNum ». Afin de faciliter la simulation, nous avons utilisé un convertisseur USB à 4 ports RS232. Peut importe l'ordinateur utilisé, il est ainsi possible d'avoir 4 ports séries à notre disposition. Les numéros de ces ports sont COM10, COM11, COM12 et COM13. Lors de l'envoi des données, le deuxième octet est initialisé à l'adresse du destinataire. La plateforme Zigbee change cette adresse par l'adresse de la

voiture qu'elle représente et l'envoie à l'adresse du destinataire. Une fois que la plateforme « destinataire » reçoit le paquet, elle l'envoi directement à l'ordinateur. C'est donc dire que le deuxième octet représente l'adresse du destinataire lors de l'envoi et l'adresse du correspondant lors de la réception. C'est lors de la réception que l'ordinateur vérifie l'intégrité des données en utilisant le dernier octet « vérificateur de sommation ». Pour avoir plus d'information sur la communication série, référez-vous au chapitre 3.

4.6 **Performances du simulateur**

4.6.1 Interface utilisateur

L'interface utilisateur a rempli toutes les exigences. L'utilisateur peut créer, ouvrir et enregistrer toutes ses configurations. Ces dernières sont enregistrées dans des fichiers xml. Il est donc possible d'enregistrer la position initiale, ainsi que la vitesse et l'accélération.

Il est possible de changer les commendes en temps réel. Une voiture esclave peut ainsi devenir maîtresse en un clic. Ce qui rend la simulation plus réaliste en ajoutant des imprévus. Le nombre de scénarios est ainsi infini, ce qui facilite le débogage et la validation des contrôleurs et des différents algorithmes pour les deux plateformes.

4.6.2 Voitures

Il est difficile de montrer les résultats de la simulation des voitures. Le modèle choisi est du premier ordre. Les résultats de simulations sont présentés ci-dessous.

L'utilisateur a le choix d'utiliser plusieurs types de contrôleurs. La logique floue, le P, PI et le PID. Chacun de ces contrôleurs a ses avantages et ses inconvénients. Le contrôleur

qui m'a donné les meilleurs résultats est la logique floue. Une fois la structure en place, il est très simple de reproduire la conduite humaine. Pour ce qui est des autres contrôleurs, puisqu'ils réagissent à une seule entrée, il est très difficile de modéliser la conduite humaine.

4.6.2.1 Contrôleur de vitesse

La voiture de tête compare sa vitesse avec la vitesse de croisière désirée. La différence entre ces deux variables donne l'erreur. Cette dernière est utilisée et donne la puissance d'entrée au véhicule. Les résultats pour ce contrôleur sont excellents. Pour visualiser les résultats, voir l'exemple AVI de la simulation.

4.6.2.2 Contrôleur de position

Le contrôleur de position utilise la distance entre les voitures et la compare avec la distance désirée (10 mètres). Comme j'ai mentionné plus haut, la logique floue à donné de très bons résultats. Les contrôleurs de type PID ont été plus difficiles à modéliser. Des limites ont dû être ajoutées afin de rendre la simulation plus réaliste.

Plus le nombre de voitures est grand dans un convoi, plus le convoi devient instable. Le modèle de base prévoyait qu'on se base sur la distance et la vitesse de la voiture précédente. Ce modèle est très simple. De plus, il réduit de beaucoup la communication entre les voitures puisque celles-ci sont munies d'un capteur de position. Il n'est donc pas nécessaire de communiquer avec la voiture qui précède toutes les fois ou nous voulons savoir sa vitesse et sa position. Par contre, ce système a amené beaucoup d'instabilité. Puisque chaque voiture réagit un peu plus lentement que la voiture qui la suit, la voiture de

queue oscille beaucoup plus que la première voiture du train. La figure 4.12 montre bien cette oscillation.

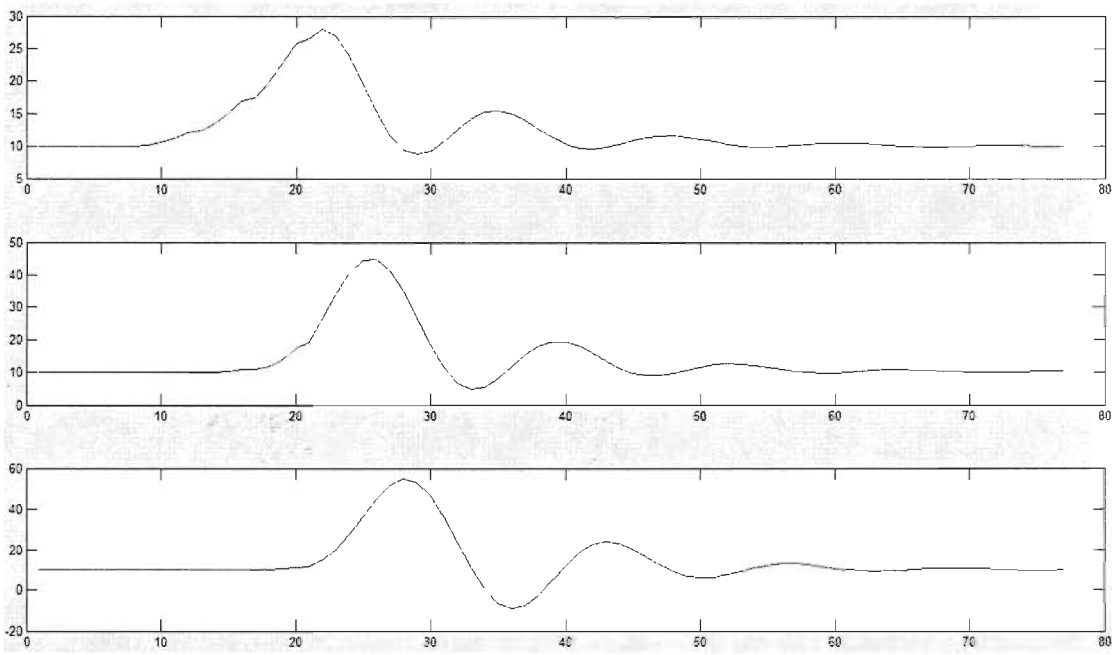


Figure 4.12 Phénomène d'oscillation dû au contrôleur de position

La valeur du gain a été diminuée afin d'augmenter cette oscillation et de la rendre plus visible sur la figure 4.12. La figure 4.12 représente la distance qui sépare les voitures. Le premier est la distance entre la première voiture et la deuxième et le dernier la distance entre la troisième voiture et la quatrième. Puisque cette dernière distance est inférieure à 0, il y a eu un accident.

Ainsi pour les contrôleurs de type P, PI et PID j'ai limité la longueur d'un tain à 4 voitures. Pour la logique floue, ce nombre a augmenté à 8.

Nous aurions pu régler ce problème en utilisant la position de la voiture de tête comme commande. Il est toutefois beaucoup plus difficile de faire l'acquisition de cette valeur. Il aurait été possible pour la voiture de tête d'envoyer sa vitesse à son groupe en utilisant la

communication RF. Une autre façon de régler ce problème aurait été d'utiliser la vitesse de la voiture de tête comme référence et de baser les contrôles sur cette dernière. Les résultats auraient été beaucoup plus stables. Par contre, une logique supplémentaire aurait dû être mise en œuvre afin de valider la position (ou la vitesse) de la voiture d'en avant. De plus, pour éviter tout décalage entre la position de la voiture et celle qui la précède un deuxième contrôleur aurait dû être mis en parallèle.

4.6.2.3 Contrôleur de la voie

Les résultats pour les changements de voie sont très satisfaisants. Plusieurs raccourcis on été pris, simplifiant de beaucoup les calculs, tout en gardant la simulation réaliste (voir la vidéo pour un aperçu de ces résultats).

4.6.3 Affichage et visuel

L'utilisation d'une animation 3d a rendu la simulation plus facile et plus réelle [2-3]. Voir les voitures se déplacer en temps réel a été un grand défi. Initialement, l'affichage était en une dimension. Celui-ci ressemblait à un vieux jeu de course de voiture des années 80. L'utilisation de la librairie DirectX à apporter beaucoup au programme. Présentement l'affiche est 3d, mais il reste toujours des améliorations à apporter aux textures. Quoi qu'il en soit, le résultat obtenu a de loin dépassé les attentes.

L'utilisateur peut naviguer sur la carte, « zoomer » sur les voitures et prendre la vision du conducteur.

4.6.4 Améliorations futures

Nous avons identifié deux améliorations à apporter au simulateur. Premièrement, la texture des voitures n'est pas finalisée. Les voitures sont pour ainsi dire transparentes. Même si ce n'est qu'une amélioration mineure et purement esthétique, il sera important de finaliser cet élément. Le deuxième point porte sur l'acquisition des données. Pour l'instant, il est impossible de faire l'acquisition de données de la simulation. Le seul support est le visuel. Il sera donc nécessaire de prévoir un algorithme pour l'acquisition et l'entreposage des données de simulation.

4.7 **Plateforme Zigbee**

La plateforme Zigbee a bien répondu aux attentes. Sa grande simplicité et les outils qu'elle incorporait nous a permis de réaliser la communication sans fil. Deux outils ont été principalement utilisés. La communication câblée série et le Zigbee. Les protocoles implantés ont tous très bien fonctionné.

4.7.1 Communication série

Après plusieurs heures d'essais, aucune erreur de communication n'a été détectée. Il a donc fallu simuler des erreurs afin de tester le protocole. Lors de ses tests, le protocole a détecté les erreurs et a permis le renvoi des données. Après plusieurs centaines d'heures de communication série, aucune erreur n'a fait en sorte que la simulation s'est arrêtée. Certes quelques erreurs d'acquisition ont pu subvenir, mais le protocole les a détectées et les données erronées n'ont pas affecté la simulation. De plus, aucune erreur sur le destinataire n'est subvenu. Nous pouvons donc dire que cette partie est un franc succès.

4.7.2 Communication Zigbee

La vitesse de communication est bonne. Le défi est tout ce qui doit être fait avant et après la communication Zigbee. Lors de l'envoi d'un paquet, nous devons au préalable passer par la communication série (ordinateur vers plateforme). Par la suite la plateforme doit décoder l'information et l'envoyer au destinataire. Ensuite, le destinataire, à son tour, décode l'information et l'envoie vers l'ordinateur. Enfin, l'ordinateur la décode et l'utilise. Le temps utilisé par la communication RF ne représente qu'une fraction du temps total. De plus, lorsque la plateforme reçoit ou envoie des données par le port série, elle ne peut pas écouter si quelqu'un lui parle. Ainsi, si une plateforme reçoit des données RF en même temps qu'elle fait l'acquisition de données, les données RF sont perdues. Il faut donc les renvoyer une fois de plus. Ceci augmente le temps de communication et retarde les envois des autres voitures. Avec le protocole et l'algorithme utilisé, il serait impossible de garantir une bonne communication entre plus de 6 voitures. Comme le nombre maximal de voitures est quatre, aucun problème n'est survenu. Par contre, un nouvel algorithme devra être mis en place si nous devions utiliser plus de voitures.

4.7.3 Interface utilisateur de la plateforme matérielle

Les résultats obtenus avec l'affichage sont très satisfaisants. L'utilisateur a un choix impressionnant de programmes à tester. De l'acquisition analogique jusqu'à la communication série, en passant par les essais RF, il est très facile pour l'utilisateur de choisir l'application qu'il désire. Voici une liste des différentes applications disponibles.

- RF_TEST
- ADC_CONV

- ADC_SERIES
- STOP_WATCH
- UART
- CLOCKMODES
- RANDOM
- AES
- FLASH
- DMA
- POWER
- TIMER_INT
- EXTERNAL_INT

Celle qui nous intéresse plus particulièrement est RF_TES. Elle utilise le Uart et le Zigbee. Les autres permettent à l'utilisateur de voir les différentes applications du CC2430. Un exemple de conversion analogique à décimale, les principales interruptions, les horloges (timer), le DMA (direct memory access), le réglage de l'horloge, ..., sont disponibles.

L'utilisateur doit utiliser le «joystick» afin de faire dérouler le menu. Les petites flèches à droite donnent les choix possibles à l'utilisateur. Ainsi s'il se trouve à la fin de la liste, seule la flèche pointant vers le haut sera visible. L'utilisateur navigue de haut en bas dans le menu en montant ou descendant le «joystick». Une fois sont choix fait, il bouge le «joystick» vers la droite. En fonction du choix de l'utilisateur, une application démarre ou un autre menu apparait. Dans le deuxième cas, il s'agit du menu de l'application. Par

exemple pour la communication UART, l'utilisateur doit choisir le débit avant le début de l'application. Il doit donc choisir cette valeur en la sélectionnant avec le «joystick». Lorsqu'il bougera le «joystick» vers la droite, l'application débutera avec la vitesse (baud rate) choisie. L'utilisateur peut mettre fin à l'application en bougeant le «joystick» vers la gauche. Il est ainsi très facile pour l'utilisateur de naviguer dans les différentes applications. C'est donc dire que tous les résultats ont été obtenus pour cette partie.

Chapitre 5 - Conclusion

Pour conclure, le simulateur m'a permis de pleinement réaliser mes objectifs. La simulation d'une chaîne de véhicules qui se déplace en convoi, la simulation des mesures, des commandes et des communications m'ont permis de donner et simuler les comportements des véhicules. L'utilisation de plusieurs contrôleurs a été effectuée. Il devient ainsi possible de valider n'importe quel contrôleur en l'implantant dans le simulateur.

L'interface utilisateur m'a permis de bien initialiser les voitures et l'affichage des données. L'affichage des voitures m'a permis de valider les différentes simulations.

Le système de communication Zigbee a permis aux voitures de communiquer les unes avec les autres. Le choix du protocole Zigbee et des plateformes CC2430 m'a permis d'avoir des communications bidirectionnelles. De plus, elles ont assuré le transport des paquets à une vitesse respectable. La distance couverte par la plateforme est suffisante. La consommation de cette dernière est telle que j'ai utilisé des batteries carrées 9 Volts, sans me préoccuper de les changer ou de les recharger durant plusieurs mois. Les outils que le CC2430 comprend (UART, ADS, DMA, ...) m'ont donné toute la latitude dont j'avais besoin pour réaliser ce projet. Les protocoles implantés pour garantir l'intégrité des données transmises ont tous bien fonctionné.

Le débit et le temps nécessaire entre les communications ont toutefois limité la quantité d'information que je pouvais transmettre. Puisque le nombre de plaques disponibles était de

quatre, je n'ai rencontré aucun problème. Mais il semble évident qu'un nombre limité de voitures pourrait se connecter à un tel réseau. De plus lors de tests où le nombre de paquets à transmettre était plus grand, ou lorsque la fréquence de communication entre les voitures était plus grande, les pertes de communication étaient plus fréquentes. Il faudrait donc penser à un nouveau protocole qui diminuerait la grosseur des paquets à transmettre si plus de voitures devaient être simulées.

Bibliographie (ou Références)

[1] Roland S. Burns, « Advanced Control Engineering », Control Engineering, Department of Mechanical and Marine Engineering, University of Plymouth, UK, 2001

[2] 3ds Max 8 Bible, Kelly L. Murdock, Wiley Publishing, Inc., 2006

[3] Advanced 3D Game Programming Using DirectX 9.0, Peter Walsh, Wordware Publishing, Inc., 2003

[4] Beginning C# Game Programming, Andy Shafran, Thomson Course Technology PTR, Thomson Course Technology PTR, 2005

[5] Microsoft C# Programming for the Absolute Beginner, Andy Harris, Premier Press, 2002

[6] Pesant Jérôme, ''Technologies de réseaux de télécommunication sans fil de proximité, Étude de positionnement', industries des technologies de l'information et des communications, Ministère du Développement économique et régional, France, Octobre 2002

[7] Sinem Coleri Ergen, ''Zigbee/IEEE 802.15.4 Summary'', 10 Septembre 2004

[8] Evans-Pughe, C.; ''Bzzzz zzz [ZigBee wireless standard'', Volume 49, Issue 3, March 2003 Page(s):28 - 31

[9] Gang Ding, Sahinoglu, Z., Orlik, P., Jinyun Zhang, Bhargava, B. ''Tree-Based Data Broadcast in IEEE 802.15.4 and ZigBee Networks Mobile Computing'', IEEE Transactions Volume 5 on, Issue 11, Nov. 2006

[10] Texas Instruments, ''Low-Power RF Selection Guide'', 2Q 2005

[11] Texas Instruments, ''Wireless Infrastructure Solutions Guide'', 2Q 2005

[12] Cipcon producs for Texas Instrument, ''CC2430 A True System-on-Chip solution for 2.4 GHz IEEE 802.15.4 / ZigBee™'', 22 Decembre 2006

[13] Cipcon producs for Texas Instrument, ''CC1110DK CC2430DK CC2510DK Development Kit User Manual'', Rev. 1.7, 12 Octobre 2006

[14] Cipcon producs for Texas Instrument, ''IAR IDE User Manual'', Rev 1.2, 16 février 2006

[15] Freescale semiconductor, ''MC13191/92 Developer's Starter Kit'', 2004

[16] Microchip Technology Inc, ''PICDEM Z Demonstration Kit User Guide'', DS51524A, 2004

[17] FlexiPanel, ''PixieTM PIC microcontroller with 2.4GHz IEEE 802.15.4 transceiver and ZigBee stack'', 29 Janvier 2006

[18] MaxStream, Inc. ''Quick Start Guide XBee™/XBee-PRO™ OEM Development Kit'', 2005

[19] ZMD AG, ''ZMD44102SKB Starter Kit Bundle'', 23 mars 2006

[20] M@teo21, Karamilo, http://www.siteduzero.com/, Version 3.0, 23 novembre 2005