2014-2015

MOVE YOUR ROBOT

RAPPORT DE PROJET – EI4 AGI

Projet réalisé par :
Alexis TEIXEIRA
Mathieu COLAS
Thomas DORLEANS
Yves FIERVILLE

Projet encadré par :
Rémy GUYONNEAU
Sébastien LAGRANGE

Remerciement

Nous tenons à remercier dans un premier temps nos professeurs référents Monsieur Rémy Guyonneau ainsi que Monsieur Sébastien Lagrange pour leur aide pendant ce projet.

Nous remercions également L'équipe de l'entreprise NAIO Technologies pour l'aide et les conseils concernant nos questions sur la programmation et sur son fonctionnement en particulier Joan Andreu.

Nous remercions aussi l'équipe de l'ISTIA de nous avoir proposé ce projet qui a été si instructif et qui nous a permis d'évoluer dans le milieu de la robotique, ce qui nous a conforté dans nos choix professionnels. Nous voulons aussi les remercier pour les aides financières qu'elle pourra nous apporter pendant le voyage pour nous rendre au concours.

Table des matières

Introduction

L'agriculture qui est un des premiers métiers du monde, a toujours connu des évolutions technologiques croissantes. Cependant, certaines tâches pénibles se font encore de manière manuelle, les technologies développées sont encore trop imprécises et polluantes. Afin d'améliorer les conditions de travail, la rentabilité et la protection de l'environnement, les avancées technologiques se font de plus en plus dans le domaine de la robotique. Certaines entreprises prennent les devants et créent des projets innovants afin de satisfaire ce marché encore trop méconnu. Ces projets provoquent un réel engouement auprès des professionnels.

C'est le cas de l'entreprise NAIO technologies, fondée en 2011. Elle regroupe des passionnés de technologies dont le but est d'aider les agriculteurs dans les tâches les plus pénibles toute en fournissant une aide efficace et en rejetant le moins de déchets possible. Cette entreprise connaît une expansion depuis ses débuts. Le succès est au rendez-vous puisque l'année dernière elle a été récompensé lors du concours Inn'Ovations en remportant le prix de l'innovation technologique grâce à son produit phare le robot Oz. Ce robot est un robot agricole, permettant de désherber et de biner les champs de manière autonome. Cette avancée technologique est une prouesse car elle permet de transformer une tâche pénible et rébarbative en une tâche simple. De plus, le robot assure un suivi du travail via des sms. A chaque tâche accomplie le robot envoie au professionnel un sms afin de le prévenir de l'avancement de son travail, permettant à l'agriculteur de vaquer à ses occupations en toute confiance.

Dans le cadre de notre année scolaire, nous devons effectuer un projet d'une durée minimale de 80 heures. L'ISTIA, ayant fait l'acquisition d'un robot Oz, nous a proposé un projet sur celui-ci en partenariat avec l'entreprise NAIO Technologies. Ce projet consiste à élaborer un programme permettant de faire fonctionner le robot de façon autonome. Il était aussi demandé d'établir un programme permettant de faire fonctionner un simulateur. Ce simulateur représentant le fonctionnement du robot dans différents types de champs. Le troisième impératif demandé était d'établir un programme innovant sur le robot. Ces trois points entraineront la participation à un concours inter-écoles dans la ville de Toulouse et les trois meilleures équipes seront récompensées par l'entreprise ambassadrice de ce concours. Notre équipe est composée de quatre élèves ayant une attirance pour les nouvelles technologies ainsi que pour la robotique.

Ce rapport a pour but d'informer les professeurs référents sur notre travail effectué, mais aussi sur l'organisation des tâches, les outils utilisés, les solutions retenues ainsi que les difficultés rencontrées lors de la réalisation de ce projet. Ce document pourra aussi servir de documentation technique pour les futurs élèves qui travailleront sur le robot.

Ce rapport se décompose en six parties, la première sera sur la présentation du concours, ensuite nous vous parlerons des différentes actions effectuées sur le robot, suivi du simulateur. La quatrième partie abordera les innovations effectuées sur le robot, enfin, les deux dernières parties traiteront de la communication autour de notre projet et de sa gestion.

1 Présentation du concours « Move your robot »

Dans le cadre de notre projet, il était stipulé que nous avions la possibilité de participer à un concours si la programmation du robot et celle du simulateur étaient jugés satisfaisantes. Il se déroulera à Toulouse et sera organisé par la société NAIO Technologies pendant deux jours du 29 au 30 mai. Ce concours regroupe plusieurs étapes, trois pour être précis, que nous détaillerons dans les parties suivantes. La première étape concernera le simulateur, la seconde se portera sur le robot en lui-même et la dernière sera sur une innovation ou amélioration dans le but de mettre en compétition notre créativité. Les trois meilleures équipes recevront un prix pour récompenser leurs efforts.

1.1 Présentation des épreuves

1.1.1 Simulateur :

Le simulateur est à la base un logiciel fourni par l'entreprise NAIO Technologies permettant de prévoir les effets des algorithmes qui seront injectés dans le robot, afin d'éviter d'endommager Little Oz. De plus, la compilation des programmes sur le simulateur est plus simple et plus rapide. Le fonctionnement du simulateur sera plus détaillé dans la troisième partie de ce rapport. Dans le logiciel le robot est représenté par un rectangle gris, les plantes par des carrés verts et la perception du LIDAR par des lignes rouges. Le but pour cette partie du concours est de faire traverser le champ à notre robot sans passer sur les plantes simulées.

Pour le concours L'entreprise nous a fourni le simulateur avec ses interfaces graphiques et le code des capteurs simulés. Nous possédions aussi un premier niveau de test du simulateur. Pour ce niveau les rangs sont de mêmes longueurs et les plantes sont toutes de mêmes tailles. La difficulté principale n'a pas été le parcours du champ en soi, mais la compréhension du fonctionnement du code du simulateur .Une fois le level 1 opérationnel l'entreprise nous a envoyé un second niveau plus complexe**.** Dans celui-ci les rangs sont de tailles différentes, les plantes n'ont plus la même taille et on peut voir des obstacles représentés par les petits carrés verts. Cette partie a été plus complexe à coder à cause de la différence de taille des rangs et de la présence d'obstacles. Nous n'avons pas reçu le niveau trois, car ce niveau sera donné lors du concours.

1.1.2 Plein champ :

Pendant cette épreuve c'est le code du robot qui sera testé. Nous aurons à notre disposition un robot Little Oz prêté par NAIO Technologies ainsi qu'un champ de plusieurs mètres carrés. Nous ne connaissons pas exactement la taille du champ ni la taille des rangs. Cette épreuve sera similaire à celle du simulateur, le but étant de parcourir le champ sans endommager les plantes et en évitant les obstacles. Cette épreuve est la principale épreuve de ce concours. Elle va pouvoir tester nos capacités à anticiper les problèmes et les imprévus et notre niveau de programmation. Les informations concernant le robot ainsi que sa programmation seront données dans la deuxième partie du rapport.

1.1.3 Innovation :

La troisième et dernière épreuve consistera en une amélioration du robot. Lors de cette épreuve nous n'avons pas de contraintes. Nous devons seulement apporter une ou plusieurs améliorations au robot et qui permettra une avancée technologique. Pour cette épreuve l'entreprise NAIO Technologies mettra à notre disposition un champ vierge avec le robot utilisé lors de l'épreuve plein champ. Nous avons eu plusieurs idées qui seront détaillées dans la quatrième partie du rapport.

1.2 Présentation de l'entreprise NAÏO TECHNOLOGIE

1.2.1 Historique et Localisation :

Mai 2010 : Lors d'une fête de l'asperge, Gaëtan Séverac rencontre un producteur avec qui il échange sur les problèmes liés à la production d'asperges.
Il a de suite l'intime conviction que les robots agricoles représentent l'avenir.
Aymeric Barthes, stagiaire dans une entreprise informatique, commença à penser à créer une entreprise « différente ». Il avait envie de tenter des expériences nouvelles, de faire des activités différentes, nouvelles tous les jours, d'utiliser sa tête mais aussi ses mains.
Après avoir terminé leurs études respectives sur les bancs de l'école d'ingénieurs IMERIR de Perpignan, ils décidèrent de créer une entreprise avec deux autres de leurs amis.

Eté 2011 : Après avoir fait mûrir leur projet en échangeant avec des professionnels du milieu agricole, mais également avec des structures d'accompagnement telles que l'incubateur Midi-Pyrénées et des représentants d'AGRIMIP Sud-Ouest Innovation (pôle de compétitivité agricole). Ils achetèrent un garage leur servant à prouver la faisabilité technique de leurs prototypes.

Novembre 2011 : Ils créèrent NAIO Technologies SAS, avec l'intime conviction de parvenir un jour à révolutionner l'agriculture. Depuis la création, les problèmes se sont enchainés ainsi que des ralentissements, des mauvaises surprises et des échecs. Cependant il y a aussi eu des moments privilégiés, de joie et de réussite. Leur crédo est simple « On avance, on avance ! »

1.2.2 Domaine d'activité :

NAIO technologies est spécialisée dans la robotique agricole. Les fonctions principales de ces robots reposent sur l'assistanat lors de la récolte et le désherbage mécanique des exploitations agricoles plus particulièrement les exploitations légumières.

1.2.3 Les objectifs :

NAIO Technologies estime que les agriculteurs utilisent des méthodes néfastes pour l'environnement et que nous détruisons de plus en plus le patrimoine animal et végétal de la planète. C'est à cause de cette pensée que Gaëtan Séverac et Aymeric Barthes ont décidé de créer l'entreprise. Cette structure accueille des personnes attachées aux valeurs des deux fondateurs et effectuant des travaux de recherche et de production au profit de l'agriculture durable. L'objectif principal de cette nouvelle société est de démocratiser l'utilisation des nouvelles technologiques à l'ensemble de l'agriculture.

1.2.4 Les produits :

Oz : Ce robot est un robot autonome de désherbage et d'assistance. Entièrement automatisé il permet le désherbage mécanique des rangs. Il possède une faible consommation en énergie grâce à ses batteries. Il permet d'éviter de tasser des sols et d'améliorer les conditions et le temps de travail.

Figure 1: Robot Oz

COSI : Ce porte-outil électrique est idéal pour les petites surfaces, il permet un désherbage rapide, efficace et sans effort. De nombreux accessoires de travail de la terre, de semis et de transport sont compatibles afin d'avoir un porte-outil multi usages.

Figure 2: Robot COSI

Little Oz : C'est la version « recherche » du robot agricole Oz. Lors de la création de ce robot l'objectif de la société était de promouvoir la robotique et l'expertise en intelligence artificielle auprès des étudiants. NAIO Technologies propose aux écoles et aux laboratoires un robot mobile roulant, simple, robuste et sans danger, que les étudiants pourront programmer sans perdre de temps ni prendre de risque sur la partie mécanique. C'est un véritable outil pédagogique, modulaire, permettant aux étudiants de mettre en valeur leurs compétences en programmant un robot autonome.

Figure 3: Robot little Oz

2 Little Oz

2.1 Présentation de robot

2.1.1 Composition

Le robot Little Oz est alimenté par deux batteries, les mêmes que nous pouvons trouver sur les voitures. Le robot possède trois cartes programmables, une Odroïd-XU pour les codes de haut niveau comment les intelligences artificielles, et deux cartes Arduino qui gèrent les composants du robot comme les capteurs et les moteurs. Ainsi, le Little Oz est équipé d'un écran LCD accompagné de multiples boutons, d'un contacteur de sécurité à l'avant, de quatre moteurs pas à pas mais également d'un grand nombre de capteurs tel que le GPS et le Lidar. Dans le tableau qui suit, nous avons détaillé les principaux composants présents dans le robot.

Composants	Information
4 moteurs	Alimentation : 24V Vitesse maximale : 0.4 m/s
Batteries au plomb	Tension : 2 x 12 V Capacité : 80 A/H
2 cartes Arduino	ArduinoCMD gère: - senseurs internes (accéléromètres) - actuateurs (commande des moteurs)
	ArduinoIHM gère : - écran LCD - Boutons - GSM
Odroïd-XU	OS : XUbuntu 13.10 4 cœurs cadencés à 1.7 GHz Gère les IA (Intelligence Artificielle)
Lidar	Capteur infrarouge (pour plus de détail cf Simulateur)
Contacteur de sécurité	Ce déclenche quand un objet heurte la face avant du robot

2.1.2 Technologies et outils de développement

L'ensemble des codes sont écrit avec le langage C++. Une fois le code transcrit, il mous faut le compiler sur le robot. Pour ce faire, nous utilisons un système Linux dans lequel nous ouvrons un terminal de commande. Premièrement, il faut envoyer notre code (le dossier OzCore) à l'intérieur du robot en utilisant la commande suivante :

```
scp –r OzCore odroid@IP:.    (IP = adresse ip du robot)
```

Le robot va demander un mot de passe : « odroid ».

Ensuite, nous nous connectons au robot avec la commande :

```
ssh odroid@IP
```

Une fois connecté, allez dans le dossier OzCore (cd OzCore) et vérifiez que le dossier build existe, si ce n'est pas le cas, il faut le créer (mkdir build). Nous pouvons maintenant lancer la compilation des codes avec les deux commandes suivantes :

```
sudo cmake CMakelists.txt

sudo make installAll –j5
```

Le robot redémarre, le code est à jour.

2.1.3 Fonctionnement du robot

Le robot fonctionne d'après l'architecture suivante :

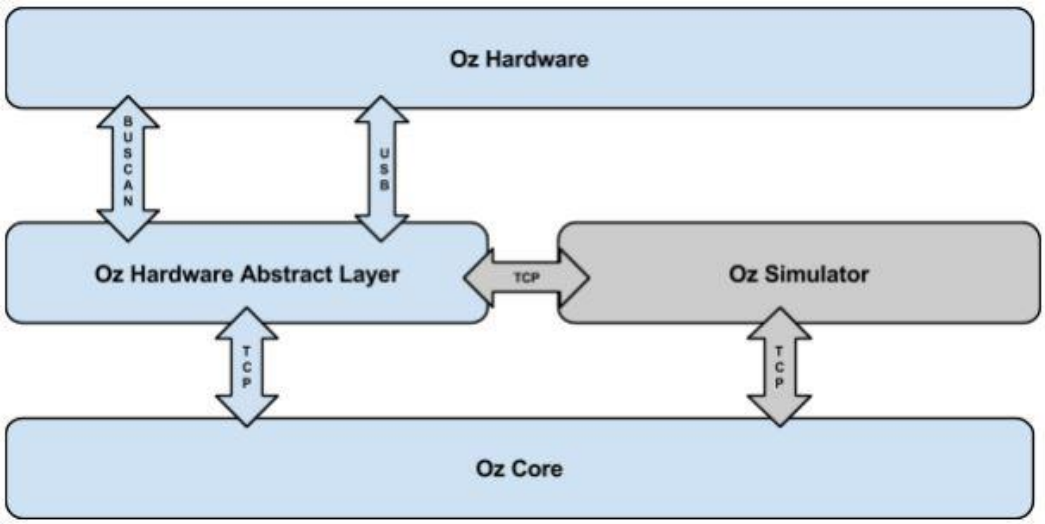


Figure 4 : Architecture du robot

Ainsi, les différentes cartes du robot communiquent entre elles via une liaison série. Les programmes s'envoient donc des messages pour récupérer des informations qui sont présentes sur une autre carte.

Le robot possède deux modes distincts de fonctionnement, le mode manuel et le mode automatique. Dans le mode manuel, l'utilisateur peut prendre le contrôle du robot via une manette de PS4 qui se connecte au robot avec le Bluetooth.

Dans le mode automatique, l'utilisateur choisit l'action que doit réaliser le robot parmi les modes proposés, binage d'une parcelle, mode suivi. En mode automatique, le robot se repère dans son environnement en utilisant le lidar.

Le robot ayant 4 roues pouvant se mouvoir séparément, il est possible de le faire tourner sur lui-même en faisant tourner les roues d'un côté dans le sens inverse de celles de l'autre côté. Pour faire un virage, le robot ralentit la vitesse de rotation des roues du côté du quel il veut tourner.

2.2 Travail réalisé

2.2.1 Analyse et fonctionnement

Avant de nous lancer dans le développement des fonctionnalités que nous voulions implémenter, il a fallu étudier la segmentation des codes qui nous ont été fournis par NAIO Technologies. Ainsi, nous avons déterminé la structure générale du code présent sur l'Odroïd. Le code qui permet de démarrer l'ensemble des services du robot est Core.cpp, ce code vérifie que les cartes Arduino sont démarrées et il lance la boucle infinie.

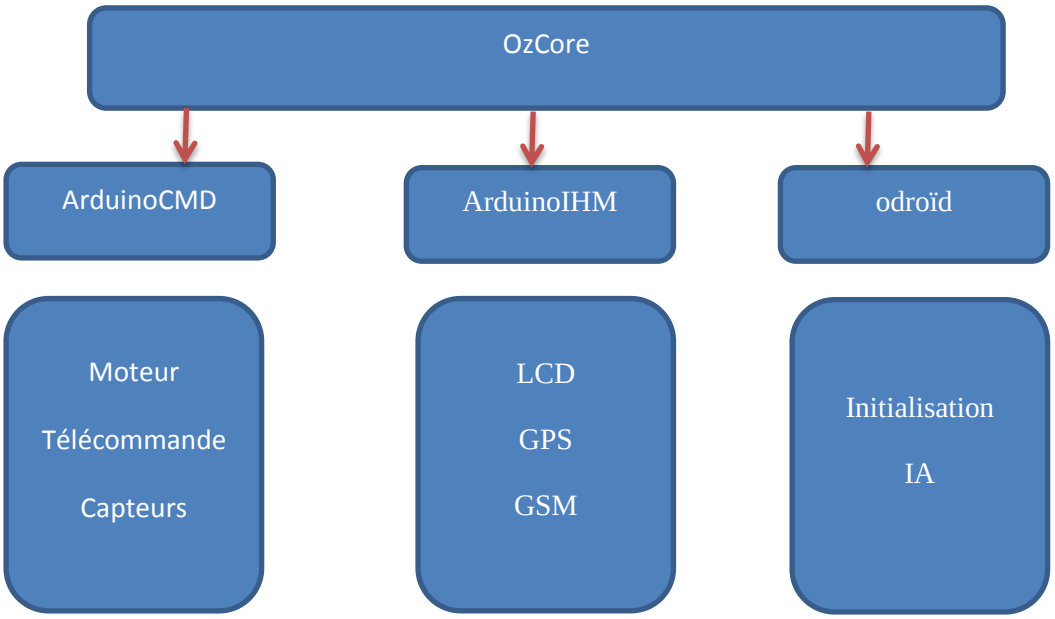


Figure 5: Organisation du code

Dans la boucle infinie du Core.cpp, il y a un appel de la fonction guidage() de la classe Decision, cet appel permet de vérifier s'il y a quelque chose à faire. C'est donc dans le code Decision.cpp que les intelligences artificielles sont lancées, c'est le code qui va permettre de lancer le binage d'une parcelle selon les informations rentrées par l'utilisateur.

Le binage d'une parcelle est séparé en deux grandes actions, parcourir les rangées et changer de rangs en fin de rangée. Le parcours d'une rangée et le changement de rangée sont des codes qui étaient initialement présent dans le code fournis.

Pour parcourir une rangée, nous faisons appel à la fonction guidageRangee() de la classe Decision, dans celle-ci, le robot récupère les infos du lidar pour déterminer sa position dans l'environnement, avec ces informations, nous allons réguler la position du robot afin qu'il ne touche pas les plantes.

Le changement de rangs se fait en appelant la fonction guidageDemiTour() de la classe Decision. Ce changement de rang est séparé en trois phases, un premier quart de tour, une marche arrière et un second quart de tour pour rentrer dans le rang souhaité.

2.2.2 Conception

L'objectif du robot dans notre projet est de parcourir un champ composé de plusieurs rangs et ceci le plus rapidement possible. Pour cela, nous nous sommes inspirés des agriculteurs qui parcourent généralement leur champ en faisant un rang sur deux. Cela permet d'avoir une plus grande vitesse dans le virage et aussi d'éviter d'avoir à faire une manœuvre pour rentrer dans le rang.

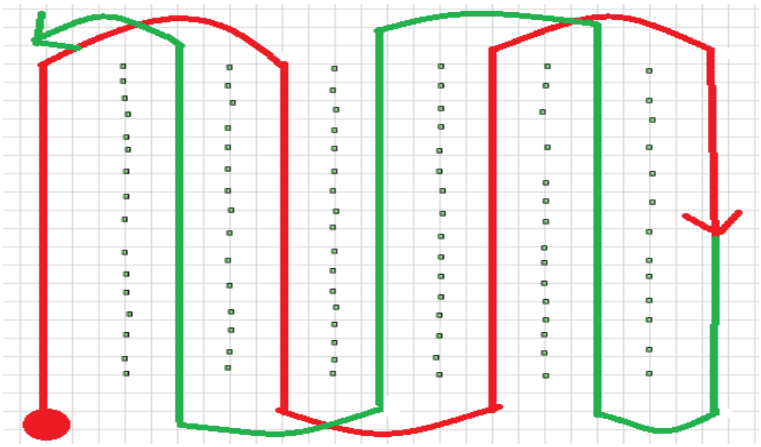


Figure 6 : Stratégie binage

Outre l'ordre dans lequel nous parcourons le champ, il nous faut modifier la méthode de changement de rang du robot pour optimiser le temps de parcours.

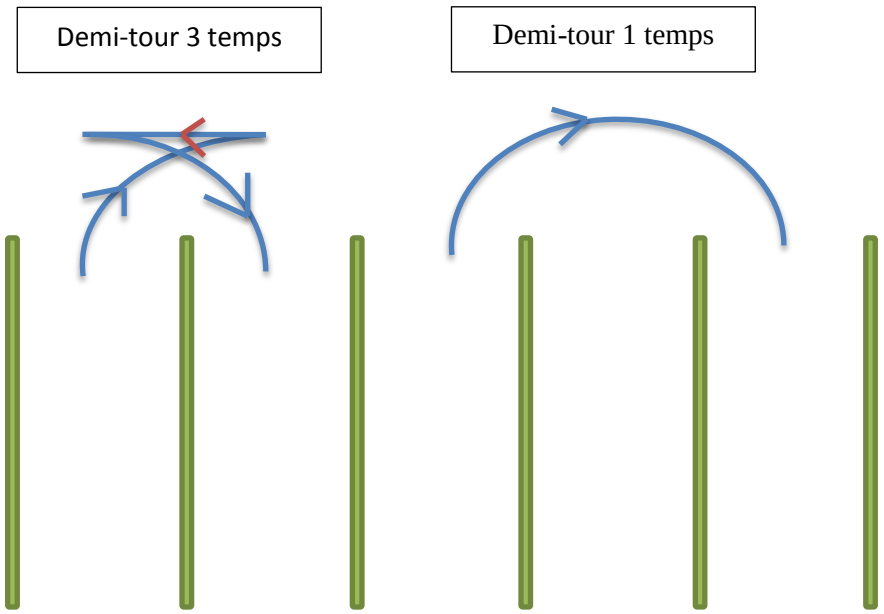


Figure 7: Stratégie du demi-tour

Pour réaliser le demi-tour en un temps, il faut calculer la vitesse de rotation des roues afin de réaliser l'arc de cercle qui nous permettra d'arriver directement dans le rang désiré. Nous avons donc réalisé le calcul suivant :

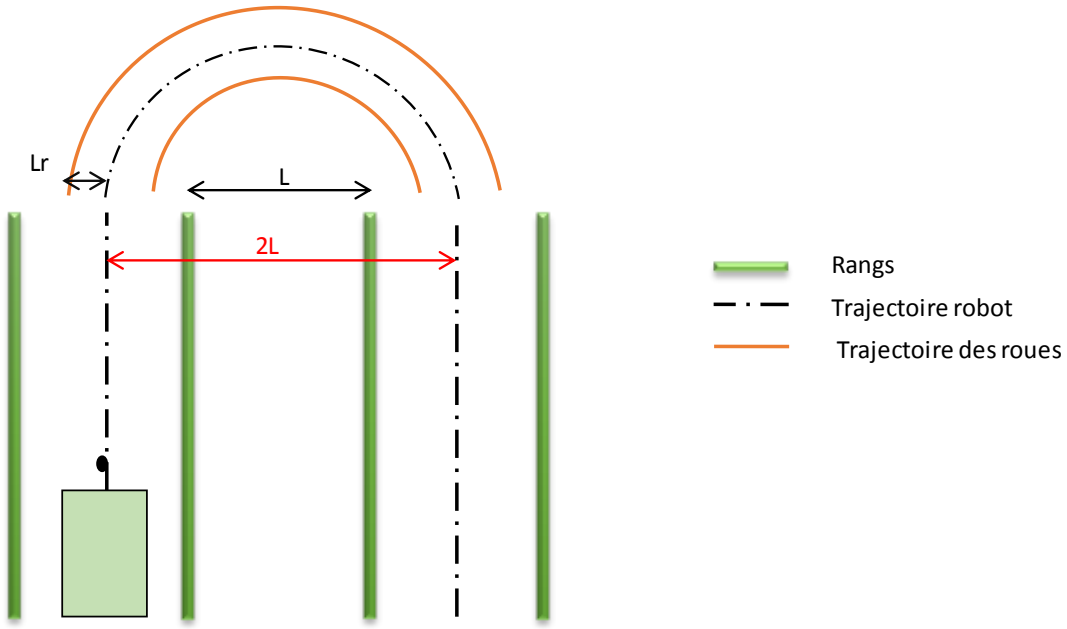


Figure 8 : Illustration calcul

Nous calculons les distances que doivent parcourir les roues internes (Di) et externes (De) :

Di=(2L-2Lr) x π

De=(2L+2Lr) x π

Nous cherchons ensuite le rapport entre les deux distances : $\frac{Di}{De} = \frac{L-Lr}{L+Lr}$

Nous calculons maintenant la vitesse pour chaque roue :

Vitesse roues externes = Vitesse maximale

Vitesse roues internes = $\frac{Di}{De}$ x Vitesse maximale

D'après le schéma de la stratégie de binage, nous observons qu'il faudra obligatoirement faire un demi-tour deux fois plus petit que ceux que nous venons de calculer.

Nous avons également réfléchis au cas où le rang serait trop large pour être biné correctement, il faut alors se serrer d'un côté lors du premier passage puis de l'autre au second passage.

2.2.3 Mise en œuvre des idées

2.2.3.1 *Lancer le binage d'une parcelle avec les codes fournis*

Dans un premier temps, nous avons testé le bon fonctionnement du parcours d'une parcelle sans modifier les codes liés aux demi-tours ainsi que pour le parcours de la rangée. Pour lancer une parcelle depuis l'IHM, il faut rentrer un certain nombre de paramètres (nombre de rangs, longueur des rangs, écart avec le rang suivant, …) et à la fin l'IHM demande validation avant de lancer le binage. C'est à cet instant qu'il faut lancer le binage de la parcelle, ce code est présent dans la classe Parametrage, à la fonction paramAttenteValid(int act). Dans ce code, nous ajoutons l'appel de la fonction startBinage() présent dans la classe Decision, d'où le code suivant :

```
case K_Valid:
        switch (p_dec->modeAutoAct){
                case MA_Binage://auto
                        p_dec->chkFin = 1;
                        p_dec->startBinage();//Prépare les variables nécessaires pour faire le binage
                        break;
```

Ensuite dans la classe Decision, dans la fonction guidage(), nous ajoutons la gestions des outils (simulé puisque nous n'avons pas d'outils), le lancement de la rangée à biner ainsi que le lancement du demi- tour : (code complet du case MA_Binage en annexe 1)

```
switch (modeAutoAct){
        case MA_Binage: // Mode binage sélectionné
                switch (phaseAct){ //phase actuel du robot
                        case P_OutilUp: // Lève l'outil avant de faire le demi-tour
                                        /* … */
                                        break;
                        case P_OutilDown: // Baisse l'outil  avant de commencer le rang
                                        /* … */
                                        break;
                        case P_Rangee: // Lance la rangée une fois que l'outil est baissé
                                        /* … */
                                        break;
                        case P_DemiTour: // Lance le demi-tour dès que l'outil est levé
                                        /* … */
                                        break;
                        default:
                        }
        break;
```

2.2.3.2 *Modifications du demi-tour*

Maintenant que nous avons un système qui permet de biner une parcelle, nous allons l'améliorer en changeant la façon de faire les demi-tours. Nous voulons à présent que le demi-tour se réalise en un seul mouvement et que le robot parcours le champ un rang sur deux. Pour cela, nous allons modifier la fonction guidageDemiTour() de la classe Decision. Dans cette fonction, il nous faut déterminer si il faut faire un grand virage, se rendre dans le rang +2, ou faire un petit virage, se rendre dans le rang juste à côté. Nous obtenons donc le code suivant : (code complet annexe 2)

```
if (((p_param->nbRangees)%2==0 && alleeCur== ((p_param->nbRangees)/2)+1))
//Petit Virage test si paire
{
	Move(p_autom->turnPID(cons,(double)p_param->largAllee[alleeCur]));
}
else if ((p_param->nbRangees)%2==1 && alleeCur== ((p_param->nbRangees)/2)+2)
//Petit Virage test si impaire
{
	if(cons.side==C_DROITE)
	{
		cons.side=C_GAUCHE;
	}
	else cons.side=C_DROITE;
	Move(p_autom->turnPID(cons,(double)p_param->largAllee[alleeCur]));
}
else
{
	//Grand virage
	Move(p_autom->turnPID(cons,2.*(double)p_param->largAllee[alleeCur]));
}
```

Pour être en mesure de réaliser ce code, nous avons également modifié la fonction turnPID de la classe Autom. Cette fonction fournis la vitesse de rotation des moteurs lors d'un demi-tour, pour que le demi-tour se fasse en une seule phase et que le robot arrive bien dans le bon rang :

```
ORDRE_MOTEUR consigne;
double coef=((dist*100.)-40.)/(2*((dist*100.)+40.)); // Calcul du rapport des vitesses
int minSpeed = 63*coef;
int maxSpeed = 63;
consigne.md = (int)(cons.speed);
consigne.mg = (int)(cons.speed);

if (cons.side == C_DROITE && cons.speed > 0){ //Virage à droite
	consigne.md = minSpeed;
	consigne.mg = maxSpeed;
}
if (cons.side == C_GAUCHE && cons.speed > 0){ //Virage à gauche
	consigne.md = maxSpeed;
	consigne.mg = minSpeed;
}
```

2.2.3.3 *Gestion du double passage*

Nous avons maintenant une fonction binage qui parcours les rangs un sur deux et qui ne fait plus de marche arrière lors de ces demi-tours. Il nous faut maintenant gérer le fait qu'une allée puisse être trop grande pour réaliser un bon binage. Dans le parcours d'une rangée, le double passage est déjà implémenté mais comme nous avons changé le demi-tour, il nous a fallu l'adapter. La position où doit se placer le robot est géré par la fonction calc_centrage() de la classe Decision :

```
COTE resultat = C_CENTRE;
if (p_param->largAllee[alleeCur] < p_param->min2rang || p_param->nbPassages == 1){ //un seul passage
        resultat = C_CENTRE; //placera le robot au centre du rang
} else if (((p_param->largAllee[alleeCur] >= p_param->min2rang) && (p_param->largAllee[alleeCur] < p_param->min3rang)) || (p_param->nbPassages == 2)){ // deux passages
        switch (passNumber){ //Nombre de passages effectué
                case 0: // premier passage, nous sommes du côté opposé au prochain demi-tour
                        resultat = (COTE)(-1*(int)(p_param->sensDemiTour));
                        break;
                case 1: //deuxième passage, nous somme du côté où on fera demi-tour
                        resultat = p_param->sensDemiTour;
                        break;
        }
}
```

Ensuite, dès que nous avons parcourue une première fois la parcelle, il faut recommencer. Mais pour cela, il faut pouvoir retourner dans la première allée qui est juste à côté de la dernière allée parcourue, nous avons donc rajouté à la fonction guidageDemiTour() ce code :

```
if (alleeCur==1 && passNumber==1) // Permet de retourner dans le premier rang
{
        if ((p_param->nbRangees)%2==0) // nombre de  rangée paire
        {
                cons.side=sensPre; // Tourne suivant le sens précédent
        }
        else if(sensPre==C_DROITE)
        {
                cons.side=C_GAUCHE; // Tourne à gauche
        }
        else cons.side=C_GAUCHE; // Tourne à gauche
        Move(p_autom->turnPID(cons,(double)p_param->largAllee[alleeCur])); //fait bouger le robot
}
```

2.2.4 Tests et améliorations

Pendant notre projet, nous avons réalisé un grand nombre de tests sur le robot. Or tester un programme sur un robot n'est pas la même chose que de tester un programme sur PC. Sur PC, il est possible de mettre des points d'arrêts et d'observer le fonctionnement du code pas à pas, ceci n'est pas possible sur le robot. Pour observer dans quelle fonction passe le robot, nous avons décidé de lui faire envoyer un SMS et d'afficher un message sur l'écran du robot quand le programme passe dans une certaine partie du code.

Lors du développement du parcours d'une parcelle, nous avons commencé par tester le parcours d'une rangée en modifiant la longueur et la largeur de celle-ci. Nous avons également fait passer des obstacles devant le robot pour vérifier qu'il ne rentre pas dedans. Ainsi, nous avons augmenté la distance d'arrêt entre le robot et l'obstacle pour éviter tout incident.

Ensuite, nous avons testé le bon fonctionnement des demi-tours que ce soit pour les grands et les petits virages. Une fois cela fait, nous avons vérifié le bon fonctionnement du binage sur une parcelle en modifiant le nombre de rangs et en mettant un nombre de passage dans les rangs à 1 puis à 2.

2.2.5 Bilan du travail réalisé

Au final, le robot est prêt pour passer l'épreuve en plein champ. Hélas, il est prêt que si l'épreuve se passe sans l'outil car n'ayant pas l'outil, nous n'avons pas eu l'occasion de tester si l'outil se lève et se baisse aux moments désirés.

Nous aurions aussi aimé avoir le temps de modifier la façon dont les rangs sont incrémentés. Pour l'instant, ils sont incrémentés un par un même si le champ est parcourue un rang sur deux. Il aurait aussi intéressant de tester le robot dans un vrai champ pour être sûr que le comportement du robot en intérieur soit le même qu'en extérieur.

2.3 Difficultés rencontrées

Lors de la préparation de l'épreuve en plein champ, nous avons dans un premier temps rencontré des problèmes pour nous connecter au robot en utilisant le réseau ad-hoc du robot. En effet, avec windows8, les réseaux ad-hoc sont difficilement accessibles. Nous avons donc utilisé un routeur pour récupérer l'adresse IP du robot et ainsi nous y connecter.

Un autre problème qui nous a ralentis dans le développement, est le grand nombre de code qui nous a été fournis. Etant pas habitués à récupérer des codes existants, nous avons eu du mal à repérer l'architecture générale de la couche odroïd.

Mais l'événement qui nous a le plus ralentit dans notre avancement fut la compilation. En effet, nous ne pouvions pas directement coder sur le robot. Il fallait donc dans un premier temps éditer le code, se connecter au robot, copier le code et compiler. Or la compilation est plutôt longue et lorsqu'il y a une erreur, il faut recommencer la manœuvre. Cela étant fait plusieurs fois par jours, nous pouvions facilement perdre plusieurs dizaines de minutes sur une journée de travail.

3 Oz Hardware Simulator

3.1 Présentation Oz Hardware Simulator

Pour la première étape du concours, « Simulation », l'entreprise NAIO nous a fourni un logiciel nommé « Oz Hardware Simulator ». C'est un simulateur entièrement programmé par leur entreprise simulant le robot Little Oz et tous les composants qui lui sont rattachés.

L'objectif de ce simulateur est de pouvoir implémenter des programmes interagissant avec les différents composants du robot Little Oz et de pouvoir les tester rapidement. De plus, le simulateur est capable de charger différents cas typiques dans lesquels le robot réel pourrait être confronté, rendant ainsi les tests très proches de la réalité.

L'un des points forts de l'Oz Hardware Simulator résident dans la méthode de connexion et les protocoles utilisés pour l'échange de données. En effet, la connexion à l'Oz Hardware Simulator s'établie grâce à différents sockets TCP classiques (un pour chaque composant) et les échanges de données, par le biais d'un protocole personnalisé de NAIO, nommé *NAIO01*. Ces deux points clés nécessaires à la communication avec le simulateur sont rigoureusement identiques sur le robot réel. De ce fait, il est possible de piloter le robot Little Oz, avec des programmes implémentés sur simulateur, par simples connexions TCP, au lieu d'utiliser l'Oz Core.

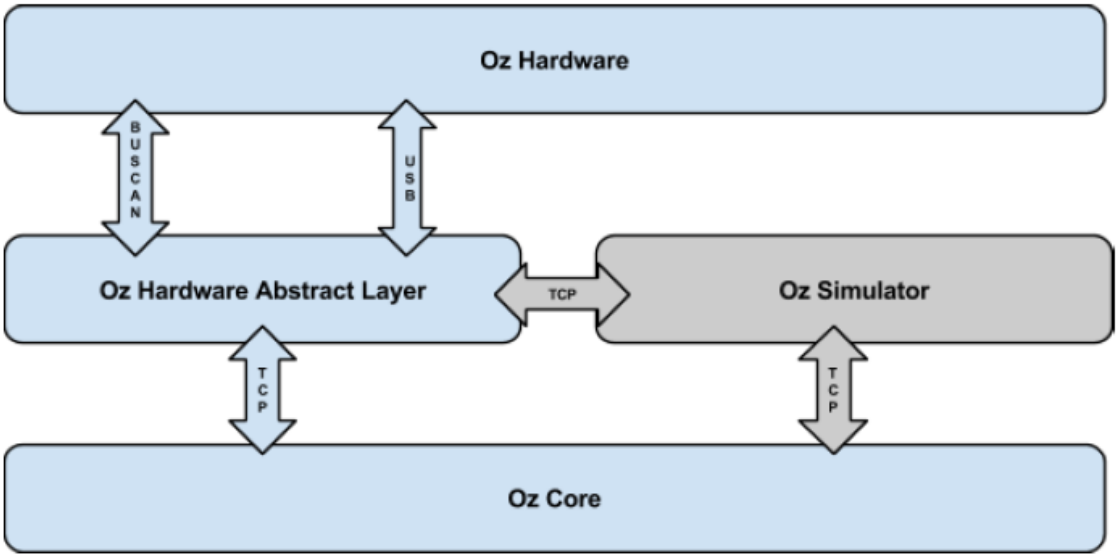


Figure 9, Diagramme d'architecture du robot Little Oz

Ce simulateur est implémenté en C#.net Framework 4.5 et évolue constamment. Lors de la première réception du logiciel, celui-ci était en version v0.0.3 et a évolué en cinq versions, durant le déroulement du projet, pour atteindre la v0.0.8. Ces mises à jour ont permis principalement l'ajout de différentes fonctionnalités et améliorations de sa stabilité.

3.1.1 Structure et fonctionnalités de l'Oz Hardware Simulator

Pour la présentation de ce simulateur, nous prendrons la version, actuellement, la plus récente, ici la v0.0.8. Cette dernière version bénéficie d'un bouton « Simulation All » permettant de lancer l'écoute de tous les ports correspondants à chaque composant du robot (dans les versions précédentes, il fallait les faire un par un).

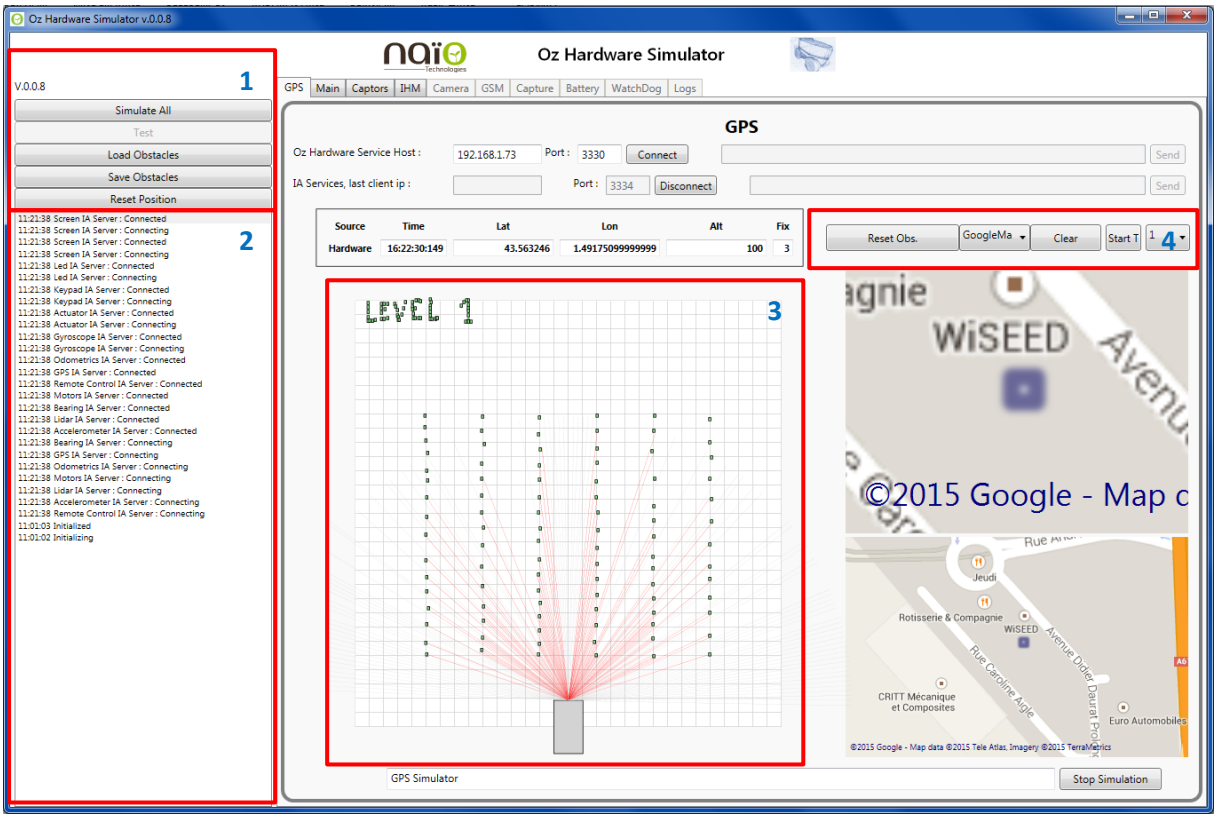


Figure 10, Oz Hardware Simulator en version v0.0.8

1 – Initialisation

Ces premiers boutons sont des boutons nécessaires à l'initialisation et à l'utilisation du simulateur.
Simulation ALL : Lance l'écoute de tous les ports correspondants aux composants simulés
Load/Save Obstacle : Charge/sauvegarde la configuration de rangs simulés appelés « obstacles »
Reset Position : *Positionne le robot à sa position par défaut (voir figure2)*

2 – Textes d'états

Dans cette « Listbox » latérale ce trouve les différents états des connexions. Ces indications permettent de visualiser si la connexion avec les différents composants du simulateur est correctement réalisée.

3 – Visualisation

Dans cette espace nous pouvons avoir un aperçu des rangs simulés. Cet écran de contrôle nous permet observer les trajectoires du robot commandées par nos différents algorithmes. De plus, il offre une représentation du LIDAR embarqué par le robot pour mieux comprendre son fonctionnement.

4 – Outils de création et visualisation

Reset Obs. : Supprime les obstacles préalablement chargés de la visualisation
Start T : Démarre le mode « tracé » permettant de mémoriser les trajectoires réalisées par le robot sous forme de traces. Celles-ci permettent de mieux analyser les performances de nos algorithmes
Clear : Supprime les traces mémorisées de la visualisation

3.1.2 Les niveaux de l'épreuve « Simulation »

L'épreuve de « Simulation » du concours se déroule en trois étapes, ou plutôt trois niveaux. Chaque niveau, de difficultés croissantes, possède ses propres caractéristiques afin de pouvoir tester les performances de notre programme (IA). Pour gagner cette épreuve, nous devons réaliser le plus précisément et le plus rapidement possible chaque niveau successivement.

3.1.2.1 *Level 1*

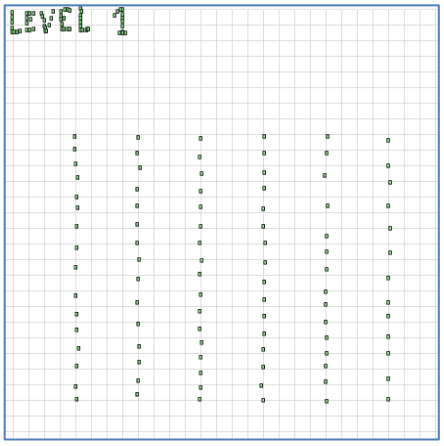


Figure 11, level 1 « Simulation »

Caractéristiques :

Nombre de rang : **6**
Type de rang : **égaux**
Longueur des rangs : **3400 mm**
Largeur entre deux rangs : **800 mm**

Ce level 1 représente un cas idéal.

Objectif de ce niveau : Réaliser la base de l'IA permettant de faire le binage de rangs.

3.1.2.2 *Level 2*

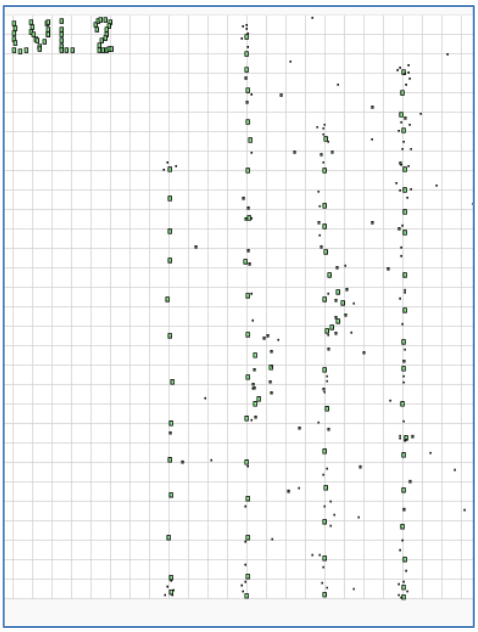


Figure 12, level 2 « Simulation »

Caractéristiques :

Nombre de rang : **4**
Type de rang : **inégaux**
Longueur des rangs :
R1 =**4400 mm**
R2 = **6000 mm**
R3 = **4800 mm**
R4 = **5400 mm**
Largeur entre deux rangs : **800 mm**

Ce level 2 représente un cas proche de la réalité, car l'environnement est parasité.

Objectif de ce niveau : Améliorer l'IA du level 1 afin traiter les données des capteurs plus intelligemment pour ne pas prendre en compte les parasites.

3.1.2.3 *Level 3*

Le level 3 n'a pas encore été communiqué par l'entreprise NAIO et ne sera dévoilé que le jour du concours. L'objectif est de nous pousser à réaliser une IA la plus performante possible afin que celle-ci soit capable de s'adapter à tous types de rangs, notamment ceux du level 3.

3.2 Travail Réalisé

3.2.1 Analyses et fonctionnalités

3.2.1.1 *Trames NAIO*

Dans nos premières recherches nous avons constaté que pour interagir avec le simulateur notre programme allait devoir utiliser la norme d'encapsulation imposé par NAIO Technologies. Cette norme impose un format de trame spécifique, détaillée ci-dessous.

Trame exemple : **LIDAR**

Header						ID	SIZE				DATA…				CRC			
N	A	I	O	0	1	0x07	0x00	0x00	0x03	0x3C	x	x	x	…	CRC	CRC	CRC	CRC

NAIO01 : L'entête de trame ou « header » NAIO01 permet au destinataire de reconnaitre le format de trame NAIO01 afin de pouvoir traiter la trame correctement.

ID : l'ID correspond à un composant du robot simulé, ici 0x07 correspond à l'identificateur du LIDAR.

SIZE : 4 octets avec pour valeur, la longueur total de la trame (Hearder+ID+Size+DATA+CRC)

DATA : Données transférées.

CRC : Contrôle de Redondance Cyclique, 4 bytes de comparaison pour s'assurer de la conformité de la trame (avant/après transfert)

Caractéristiques des trames NAIO : MOTEUR / LIDAR

Moteurs simulés id : 0x01

- 1 byte : LCM, sbyte value,
  Vitesse gauche [-127;127]

- 1 byte : RCM, sbyte value,
  Vitesse droite [-127;127]

Vitesses utilisées :
0 : stop
127 : Vitesse max avant
-127 : Vitesse max arrière

Lidar simulé id : 0x07

- 2 * 271 bytes : distance[271], integer16[271]
Distance : distance auquel se trouve l'obstacle

- 1 * 271 bytes : Albedo, byte value[271]
Albedo d'un obstacle : pouvoir réfléchissant

Caractériques des composants simulés détaillés « AnnexeX ».

3.2.1.2 *Communication TCP*

Pour échanger des trames NAIO avec le simulateur, il est nécessaire d'ouvrir une connexion TCP pour chaque composant (et donc sur chaque port correspondant). Le simulateur est capable de jouer le rôle du client TCP aussi bien que celui du serveur. Nous avons donc décidé que le simulateur serait le serveur TCP et notre programme le Client, pour plus de simplicité.

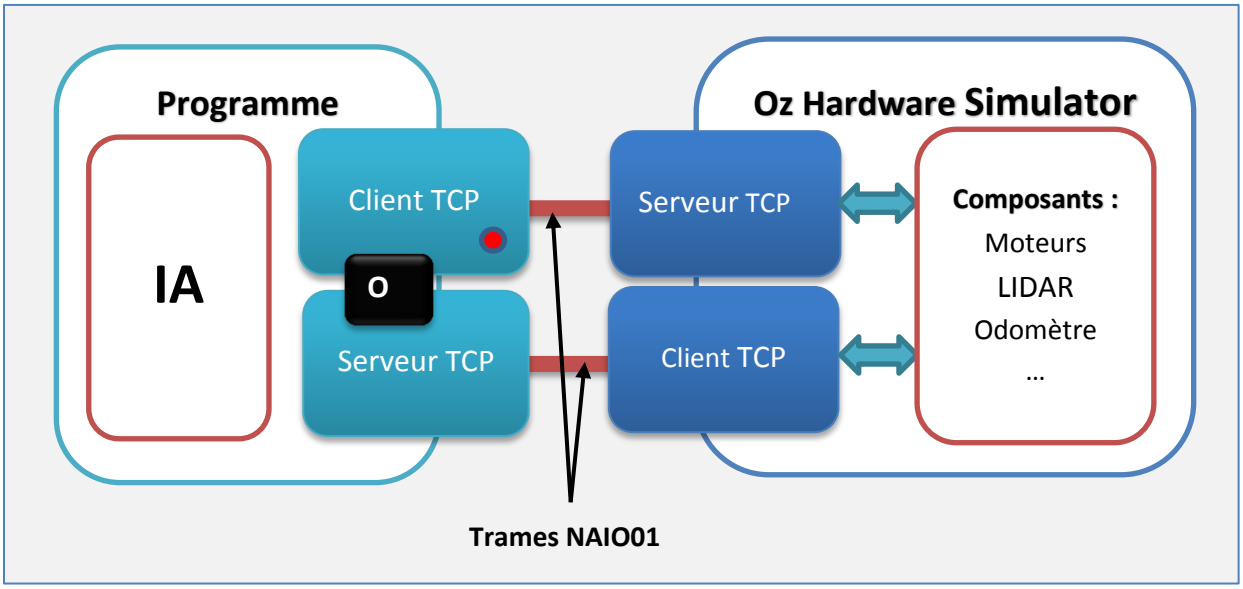


Figure 13, Schéma connexion de notre programme avec le simulateur

3.2.1.3 *Les moteurs*

Les moteurs simulés, de type moteur pas à pas, sont commandés de la même façon que le sont ceux du robot Little Oz. C'est-à-dire qu'ils réalisent un nombre de pas demandé dans un intervalle de temps fixe. Ces nombres de pas sont limités à -127 pour la marche arrière et 127 pour la marche avant pour le simulateur tandis que sur le Little Oz ceux-ci sont limités respectivement à -64 et 64.

Pour maximiser les performances sur le simulateur, nous avons décidé d'utiliser le plus souvent possible la vitesse maximum sans perdre en précision pour autant.

Valeurs dans rangées:

Ligne droite	Evitement gauche	Evitement droit	Détection fin de rangée
Moteurs de Gauche 127 Moteur de Droite 127	Moteurs de Gauche 127 Moteur de Droite 72	Moteurs de Gauche 72 Moteur de Droite 127	Moteurs de Gauche 38 Moteur de Droite 38

Valeurs changements de rangs :

Grand virage à gauche	Grand virage à droite	Petit virage à gauche	Petit virage à droite
Moteurs de Gauche 38 Moteur de Droite 127	Moteurs de Gauche 127 Moteur de Droite 38	Moteurs de Gauche 0 Moteur de Droite 127	Moteurs de Gauche 127 Moteur de Droite 0

3.2.1.4 *LIDAR*

Le LIDAR est le capteur le plus important du robot. C'est ce composant qui permet de détecter les obstacles face à lui sur 270°. Cette technologie de capteur a été produite par l'entreprise SICK spécialisé dans les capteurs industriels.

Principe de fonctionnement

Le LIDAR est un capteur qui utilisant une multitude de faisceaux lasers infra-rouges pour détecter des obstacles à 270°. Chaque laser émit est renvoyé par réflexion dès lors qu'un obstacle est à portée et les données traitées de la réflexion ter apportent trois informations :

- **La distance** : en mm, de 0 à 4000 mm
- **Le degré** : en °, de 0° à 270°, aveuglé à 180° sur le robot et le simulateur
- **Qualité de la réflexion** : en %, de 0% à 100% (pour le simulateur 100%, cas idéal)

Toutes ces données sont stockées dans un tableau où chaque case représente un degré. Cependant la distance maximale (4000) dépasse la taille limite d'un octet (255). De ce fait, chaque case du tableau correspondant à un degré est dédoublée pour acquérir correctement la distance. Au final, nous obtenons un tableau d'une taille de 813 octets (271*2+271).

Exemple : un obstacle détecté à 120° pour une distance de 3600 mm

118	119	120	121	122	123
0x00	0x00.	**0x0E**	**0x10**	0x00	0x00

Les cases 120 et 121 représentent le degré 120°.

120 : **bits de poids fort**

121 : **bits de poids faible**

Ce qui nous donne 0x0E10 pour la distance à 120°, $0E10_{(16)} = 3600_{(10)}$, donc bien 3600 mm.

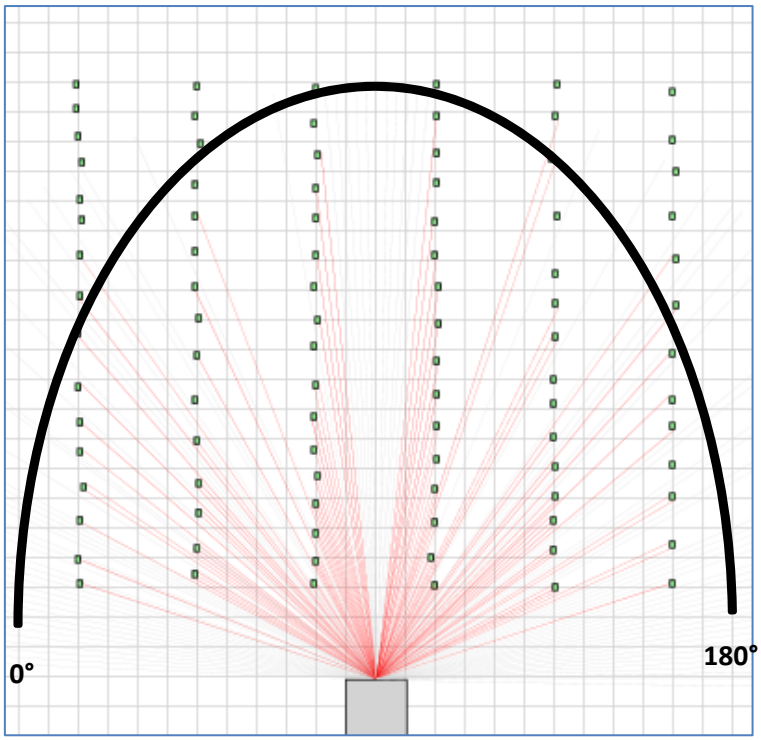
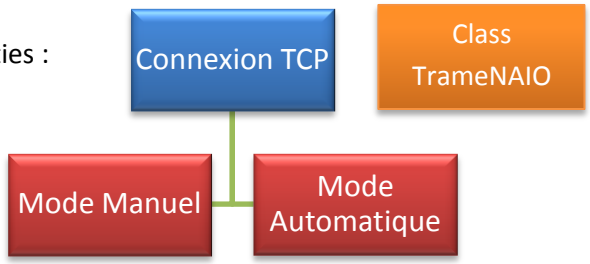


Figure 14, Visualisation du LIDAR

3.2.2 Conception

Notre programme est composé de quatre principales parties :

- Class TrameNAIO
- Connexion TCP
- Mode Manuel
- Mode automatique (IA)



3.2.2.1 *Class TrameNAIO*

Les trames de communication avec simulateur suivent un format spécifique à NAIO, nous avons donc décidé de réaliser une classe TrameNAIO. Cette classe permet d'initialiser et manipuler les trames simplement, par le biais de différentes méthodes afin de ne pas avoir à se soucier du format NAIO01. Ce format est automatiquement généré dès lors qu'une TrameNAIO est instanciée avec un ID correspondant à un composant. Cette classe ne sera pas détaillée dans la partie « 3. Développement » mais est disponible en annexe 3.

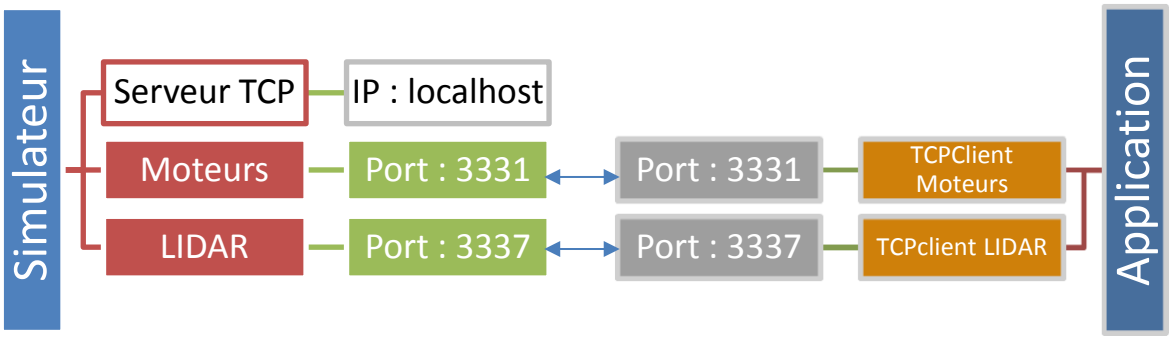
TrameNAIO
- _id : byte - _matrame : byte[] - _taille : int
- InitTrame (byte[] trame) : void + TrameNAIO (byte ID) + TrameNAIO (byte ID) + TailleTrame : int + TailleData: int + ValeurOctet (int index) : byte + ValeurData (int index) : byte + GetTrame () : byte[] + SetTrame(index :int, value : byte) : void + Affiche (void) : string

3.2.2.2 *Connexion TCP*

La connexion TCP est essentielle au bon fonctionnement des autres parties du programme. En effet, si la connexion entre l'application et le simulateur n'est pas réalisée, le mode manuel et le mode automatique deviennent inutiles, ne pouvant plus envoyer de commandes ni recevoir des données. Pour ce faire, les différents modes ne sont disponibles que si et seulement si la connexion TCP est établie.

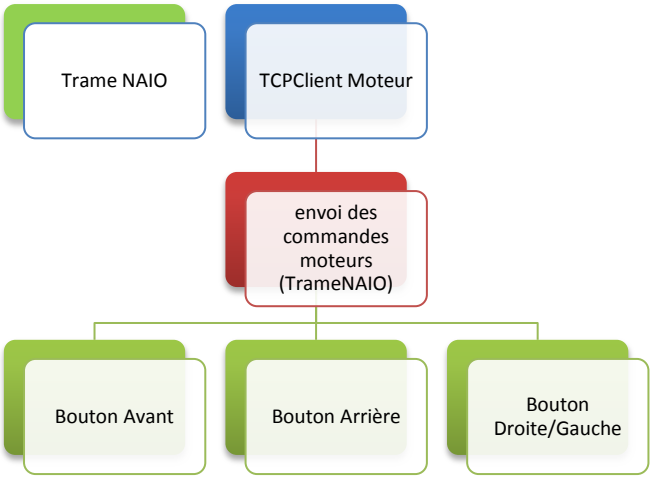
La connexion TCP est dîtes établie si les composants nécessaires au fonctionnement du mode manuel et du mode automatique sont tous connectés. C'est-à-dire les que les deux composants simulés (Moteurs et LIDAR) sont bien connectés sur leur port respectif.

- Les moteurs, disponible sur le port 3331 et l'adresse IP de la machine sur laquelle se trouve le simulateur
- Le LIDAR, disponible sur le port 3337 et la dîtes adresse IP



3.2.2.3 *Mode Manuel*

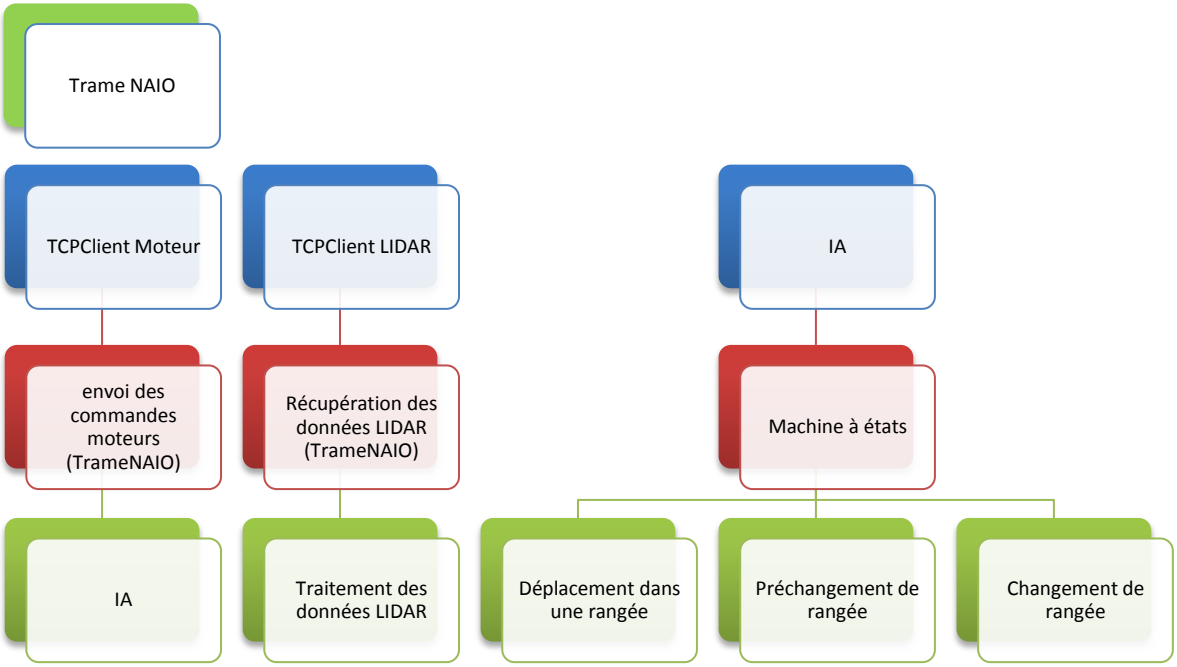
La mode manuel est un mode où l'utilisateur peut envoyer des commandes des déplacements via des simples boutons (Av, Ar, gauche et droite) et des champs numériques pour spécifier le nombre de pas à réaliser. Le programme traite le nombre de pas demandé afin que celui-ci corresponde à l'intervalle limite des moteurs ainsi qu'au format des trames NAIO.



3.2.2.4 *Mode Automatique*

Le mode automatique permet d'obtenir un comportement autonome du robot simulé grâce à une IA contrôlant les moteurs et analysant les données du LIDAR. Cette IA fonctionne sur le principe d'une machine à états. Cette méthode permet d'obtenir un code séquentiel très structuré ressemblant aux grafcets en automatisme.

Déplacement dans une rangée : *le robot entre dans un mode de régulation afin de rester le plus au centre possible lors du binage*
Pré changement de rangée : *le robot entre dans un mode de fin de rangée et s'assure de bien sortir du rang pour ne pas heurter les plantes avec sa bine.*
Changement de rangée : *le robot change de rangée, la direction (gauche/droite) et le type de virage (grand/petit) sont déterminés en fonction de l'avancement du binage et des virages précédents*

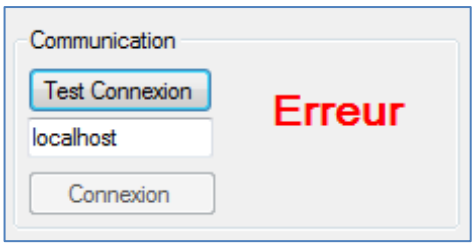

Trame NAIO
TCPClient Moteur
TCPClient LIDAR
IA
envoi des commandes moteurs (TrameNAIO)
Récupération des données LIDAR (TrameNAIO)
Machine à états
IA
Traitement des données LIDAR
Déplacement dans une rangée
Préchangement de rangée
Changement de rangée

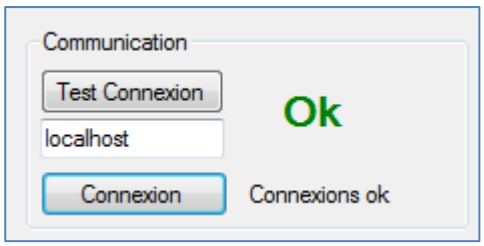
3.2.3 Développement

Nous avons développé l'application en langage C#.net Windows application, sous Microsoft Visual Studio. Quelque mois avant le début du projet, nous avons appris à utiliser ce langage et donc nous avons simplement décidé de l'approfondir avec ce projet.

3.2.3.1 *Connexion TCP*

Pour réaliser la connexion TCP sur chaque composant, le programme test au préalable si la connexion est possible, grâce à un test de connexion sur les moteurs. Si le test de connexion est réussi, alors la connexion de tous les composants peut s'exécuter.





La classe utilisée pour les connexions TCP est la classe : TCPClient, cette classe fournit des connexions clientes pour les services réseau TCP, parfaite donc pour notre application.

Les codes correspondants aux connexions TCP sont en annexe 4.

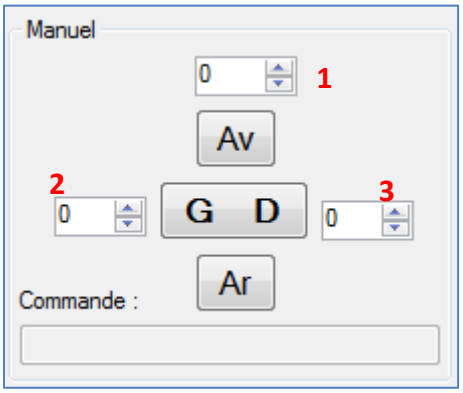
3.2.3.2 *Boutons manuels de commande*

Pour piloter le robot simulé en mode manuel, une IHM est présente et permet d'envoyer facilement des commandes au simulateur.

Les boutons « Av » et « Ar », respectivement avant et arrière, permettent de faire avancer ou reculer le robot simulé d'un nombre de pas choisit en 1.

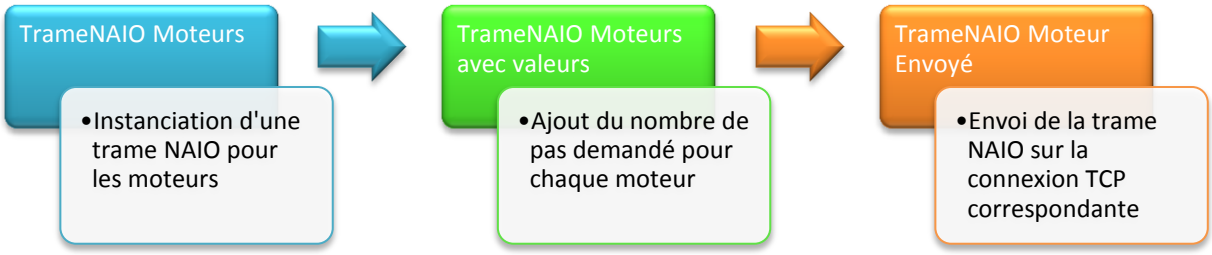
Le bouton central « G D » permet la rotation du robot. Pour réaliser une rotation il est nécessaire que les champs 2 et 3 soient différents, sous peine de simplement avancer ou reculer.

Les intervalles de valeurs sont les mêmes pour les trois champs, [-127 ; 127] (les valeurs négatives sont traitées dans le programme pour correspondre aux déplacements souhaités)



Les codes correspondants au mode manuel sont en annexe 5.

3.2.3.3 *Envoi de commandes moteurs*



Les codes correspondants à l'envoi des commandes moteurs sont en annexe 6.

3.2.3.4 *Récupération des données LIDAR par Thread*

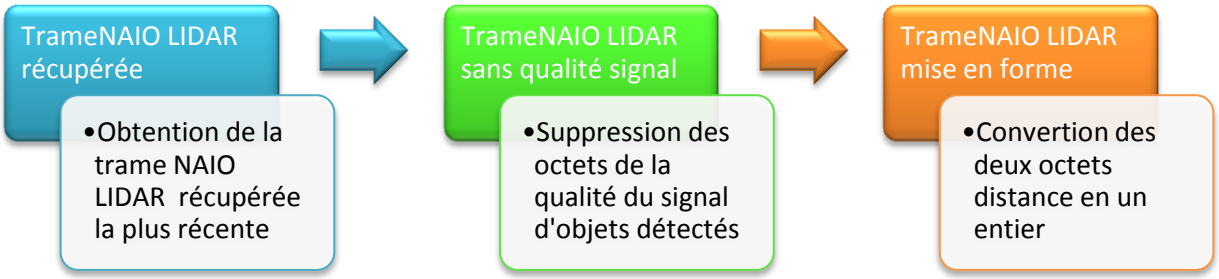
Dès lors qu'une connexion est établie avec le LIDAR, le simulateur envoi les trames de données toutes les 20 ms (fréquence de rafraichissement du capteur). Cependant si ces trames ne sont pas lues par le client connecté (accusé la réception (TCP)), celles-ci se stockent les unes après les autres sur le simulateur en attendant d'être envoyées. De ce fait, si le client ne traite pas les trames du LIDAR au fur et à mesure des envois, les données reçues ne seront pas les données en temps réels.

Pour résoudre ce problème, nous avons décidé d'utiliser un « thread » de récupération des données LIDAR afin de lire en tâche de fond les trames envoyées mais sans les traiter. Avec cette méthode, dès lors que nous voulons connaitre les données réelles du LIDAR, il suffit simplement de récupérer la dernière trame lue.

Les codes correspondants à la récupération des données LIDAR sont en annexe 7.

3.2.3.5 *Mise en forme des données LIDAR*

La trame NAIO des données du LIDAR sont dans un format de tableau difficilement exploitable. De ce fait, une mise en forme de ces données est requise afin de simplifier les algorithmes qui suivent pour l'IA. Comme expliqué dans l'analyse du LIDAR, les valeurs des distances sont dédoublées sur deux octets et la qualité de réflexion du faisceau est inutile sur un simulateur car fixe. La mise en forme des données consistes donc seulement à réduire celles-ci en un tableau de 181 cases (0° à 180°) avec comme valeurs des entiers correspondants aux distances des obstacles détectés.



Les codes correspondants à la mise en forme des données LIDAR sont en annexe 8.

3.2.3.6 *Traitement des données LIDAR*

Le traitement des données du LIDAR a pour rôle d'analyser si les obstacles détectés doivent d'être évités, suivis ou encore ignorés. Ces choix sont déterminés par des seuils fixés par la géométrie des rangs et celle du robot. Les obstacles correspondant aux seuils sont stockés dans des tableaux pour être ensuite utilisés par l'IA. Le seuil d'ajout est défini sur la largeur des rangs afin de ne prendre en compte exclusivement les obstacles appartenant au rang dans lequel se trouve le robot.

Exemple :

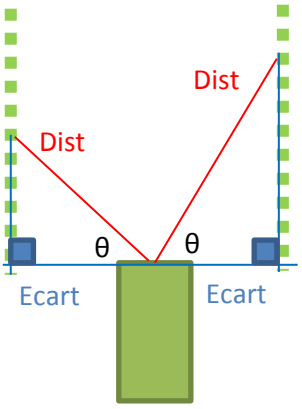
SI : un obstacle est détecté entre 0° - 75° et que la distance est inférieur à mon seuil
ALORS : ajouter cette obstacle dans mon tableau des obstacles à prendre en compte à gauche
SINON : ignorer cette obstacle

Ensuite pour savoir si le robot se trouve bien au centre, il faut homogénéiser les valeurs correspondantes aux obstacles. Pour ce faire, chaque valeur va être modifiée par un calcul trigonométrique simple pour obtenir la distance entre le centre du robot et les rangées latérales.

Ce calcul est le suivant :

$Ecart_{Robot/rang}$ = Distance x cos(Angle)

Ecart *= écart entre le robot et le rang*
Distance *= distance de l'obstacle détecté*
Angle *= angle(°) correspondant au faisceau*



Les codes correspondants au traitement des données LIDAR sont en annexe 9.

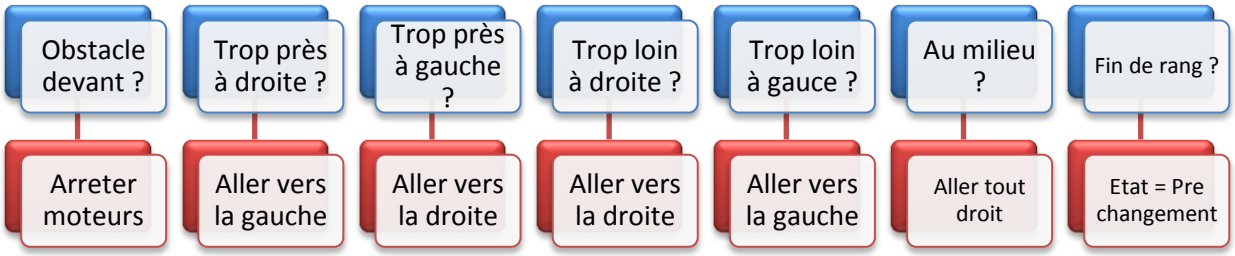
3.2.3.7 *Machines à états*

Afin de piloter le robot simulé correctement, nous avons décidé d'utilisé le principe de la machine à états. Les états du robot pendant un binage peuvent être très simplement divisés en trois états, dans une rangée, pré changement et changement de rang comme expliquer dans la conception.

```
void ModeAutomatique()
    {
      switch (Etat)
      {
        case "Initiale": break;
        case "DansRangée": DeplacementRangee(); break;
        case "SortieDeRangée1": PreChangementRangee(); break;
        case "SortieDeRangée2": PreChangementRangee(); break;
        case "ChangementDeRangée": ChangementRangee(); break;
        default: break;
      }
    }
```

3.2.3.8 *Déplacement dans une rangée*

Le déplacement dans une rangée est la première étape dans le pilotage autonome du robot. Dans ce mode ou état, l'objectif est de rester le plus au centre entre les deux rangées afin de réaliser un binage le plus rectiligne possible. Le principe de fonctionnement est simple, si le robot se rapproche trop près de la rangée de gauche celui-ci se décale vers la droite et inversement. De même que si le robot s'éloigne trop d'une rangée, il devra s'en approcher. De plus comme sur le robot réel, si un obstacle est détecté en face du robot, alors cela entraine un arrêt des moteurs tant que l'obstacle est présent. Pour sortir de cet état, il est nécessaire que le LIDAR ne détecte plus d'obstacle après trois déplacements lents.



Les codes correspondants du déplacement dans une rangée sont en annexe 10.

3.2.3.9 *Pré changement de rangée*

Le pré changement n'est présent que pour préserver les plantes. Il permet de faire avancer suffisamment le robot en dehors du rang pour que lors du virage, la bine n'heurte pas les plantes comme celle-ci est installer à l'arrière du robot.

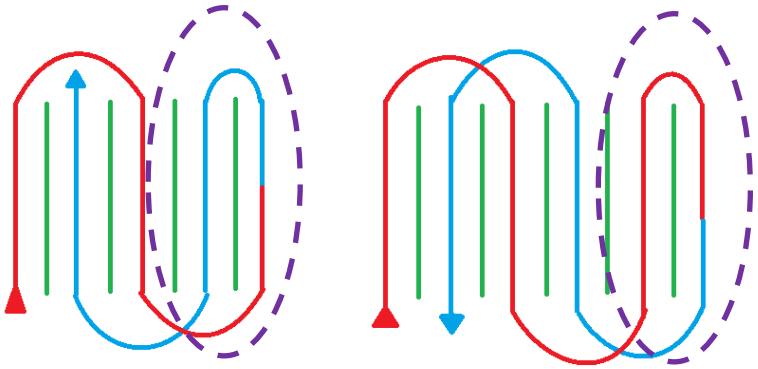
Son code est le suivant :

```
void PreChangementRangee(){
   envoi_commande_moteur(byte.Parse("127"), byte.Parse("127")); //tourner gauche petit
   if (Etat == "SortieDeRangée1") // si avancé une fois
   {
     Etat = "SortieDeRangée2"; //changement d'état en "Prechang2"
   }
   else // si avancé deux fois
   {
     Etat = "ChangementDeRangée"; // Changement d'état en "changement"
   }
```

```
}
```

3.2.3.10 *Changement de rangée*

Le changement de rangée est l'algorithme le plus complexe de notre IA. Du fait de notre stratégie est de réaliser un rang sur deux, notre programme est capable de déterminer à quel moment il doit réaliser son demi-tour ainsi que son virage serré. En effet, ce moment varie en fonction de la parité des rangs à biner.



Rangs pairs rangs impaires

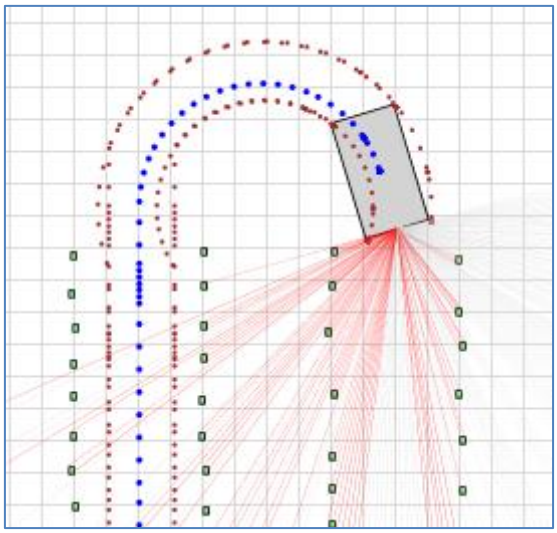
Si les rangs sont au nombre pair, alors dès que le robot engage le dernier binage avant demi-tour, il doit le faire par un petit virage et ensuite réaliser le demi-tour avec un grand virage.

Si les rangs sont au nombre impair, alors dès que le robot engage le dernier binage avant demi-tour il doit le faire normalement (grand virage) et réaliser sont demi-tour par un petit virage.

Les codes correspondants au changement de rangée sont en annexe 11.

3.2.3.11 *Est dans rangée ?*

Une fois dans le mode « Changement de rangée », pour en sortir il est nécessaire que le robot soit en face d'une nouvelle rangée à biner. Pour le savoir, nous avons implémenté deux fonctions appelées « EstDansRangée » et « EstDansRangée1 », respectivement utilisée pour les grands virages et les petits virages. Ces fonctions se basent sur le nombre d'obstacle qu'elles détectent à droite ou à gauche du robot et quand ce nombre dépasse celui du seuil fixé, alors le robot est considéré dans une nouvelle rangée à biner et le mode passe en « Dans Rangée ».



Les codes correspondants à EstDansRangéee EstDansRangée1 sont en annexe 12.

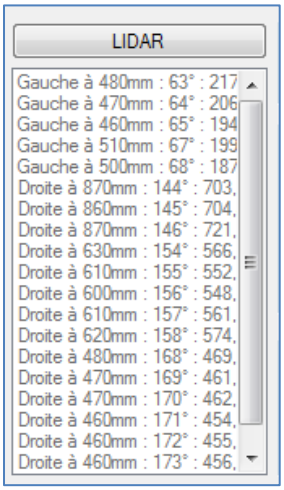
3.2.4 Tests et amélioration

3.2.4.1 *Affichage des données lidar*

Afin d'obtenir les informations du LIDAR en temps réel, nous avons décidé d'ajouter à notre application une fenêtre d'information affichant les données LIDAR reçues et traitées pour nous aider lors de la programmation de notre IA.

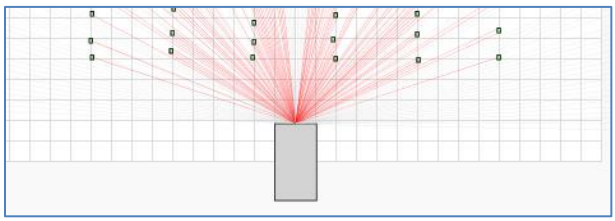
La fenêtre affiche en mode automatique, les données du LIDAR avant chaque prise de décision de l'IA. Ces informations sont très utiles pour vérifier si le robot réagit correctement.

En mode manuel, il est nécessaire de faire une requête d'obtention d'information LIDAR en appuyant simplement sur le bouton « LIDAR ».

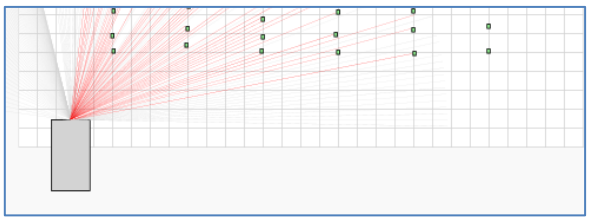


3.2.4.2 *Positionnement*

Par défaut le robot se trouve au centre, en bas des rangs. Méthodologiquement, le robot doit être placé au début des rangs externes pour réaliser un cycle de binage structuré.

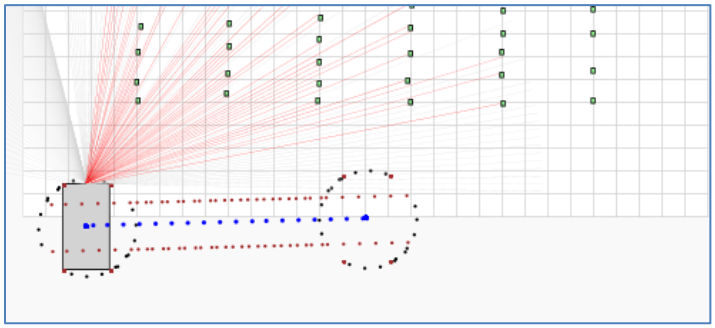


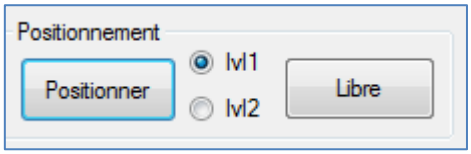
Position par defaut



Position de départ

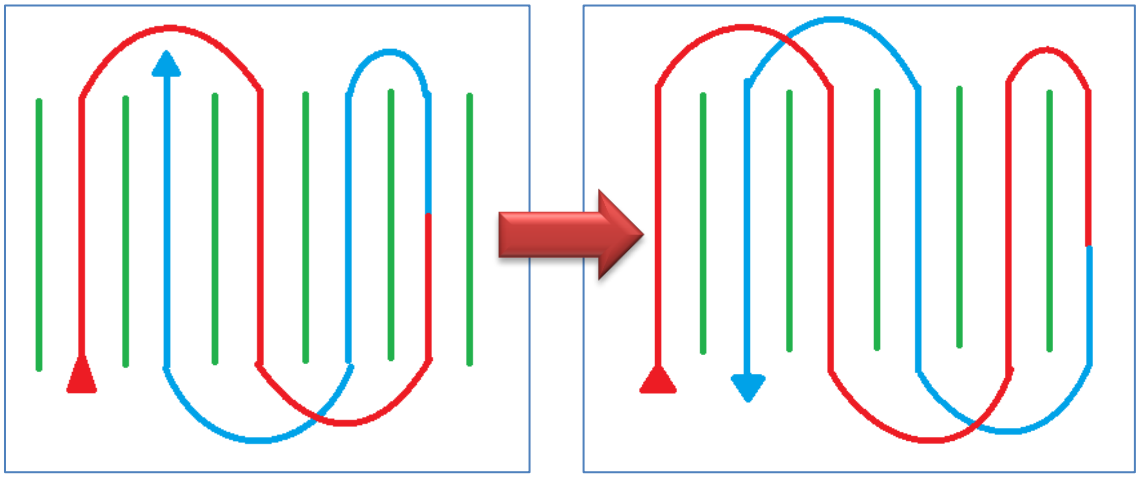
Pour faciliter la mise en positon au début du cycle pour le level1 et le level2, nous avons décidé d'implémenter une mise en position automatique au programme. Dès lors que l'on clique sur le bouton « Position », le programme va exécuter la séquence de mise en position du level choisi.





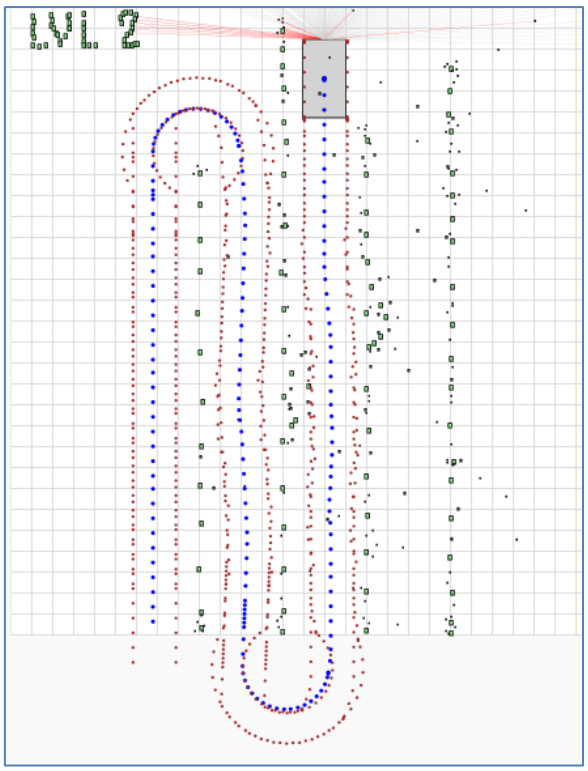
3.2.4.3 *Rang extérieur*

Afin d'apporté un plus au binage du simulateur demandé, nous avons décidé que notre programme allait également biner l'extérieurs des rangs externe.



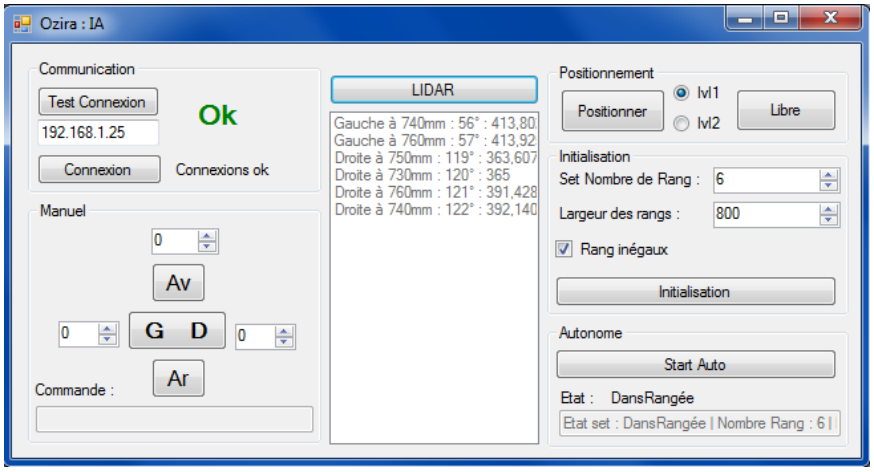
Option rangs difficiles

Le level 2 est un cas représentant une configuration de rangées vraiment mauvaise. Des parasites à l'intérieur des rangées et des rangs inégaux, cette configuration nécessite un traitement plus précis que la normale et nous avons donc décidé d'implanter l'option « Rangs inégaux » à notre application. Dès que cette option est choisie, l'IA change quelque peu en ne parcourant plus un rang sur deux mais un rang à la fois.

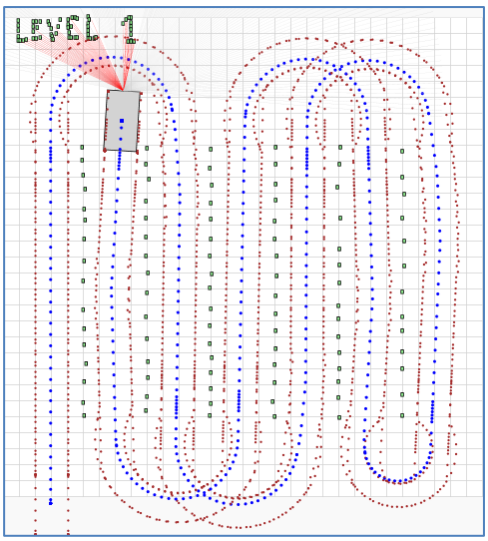


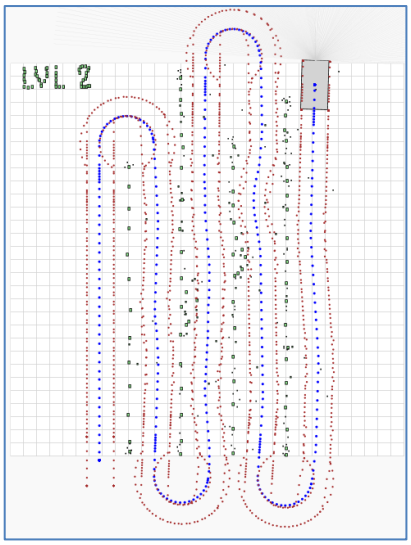
3.2.5 Bilan du travail réalisé

La programmation de l'application aura été une grande partie du projet ayant nécessité beaucoup de temps pour l'analyse et l'implémentation des différentes parties constituant le programme. L'application finale est opérationnelle et stable répondant au cahier des charges de l'épreuve simulation.



En effet, l'IA programmé dans l'application permet de réaliser les deux niveaux imposés par NAIO Technologies. Le « level3 » n'étant pas encore disponible nous avons réalisé cet IA au mieux pour tenter de garantir un bon fonctionnement dans un maximum de configuration possible.





Arrivé à terme de notre projet, nous avons réalisé une application capable de se connecter au serveur du simulateur et de commander le robot simulé avec deux modes de fonctionnement (manuel et automatique). Nous avons programmé chaque partie du programme étape par étape pour constituer une architecture solide et structurée.

3.2.6 Améliorations possibles

Même si l'application est opérationnelle pour le concours, elle reste améliorable sur certains points. Le robot réel est doté d'une multitude de composant en plus des moteurs et du LIDAR malheureusement faute de temps, le concours ne se limitant pas à l'épreuve de simulation, nous n'avons pas exploré l'utilisation des autres composants. De ce fait, la première amélioration de l'application consisterait à utiliser ces autres composants afin de rendre l'IA plus performantes.

Une autre piste d'amélioration serait la prise en compte de l'écart par rapport au centre de deux rangées pour la correction de position. En effet, le programme actuel corrige la position du robot par une commande moteur fixe et non fonction de l'écart.

Exemple :

```
if (PosD.Min() > LargeurRobot + 75) // le robot est trop loin de la rangée de droite
          {
             // alors se rapprocher du bord droit
            envoi_commande_moteur(byte.Parse("127"), byte.Parse("72"));
          }
```

Ici nous constatons que la commande moteur de correction est réalisée par des valeurs fixes déterminées empiriquement. Prendre en compte l'erreur de position pour déterminer ces valeurs améliorerait significativement la précision de la régulation lorsque le robot se trouve dans le mode « Dans Rangée ».

La dernière principale amélioration de l'application se trouve dans le traitement des données LIDAR. Du fait que les plantes sont alignées pour créer des rangées, il est possible de réaliser des extractions de lignes. Cette méthode permettrait de ne prendre en compte que les lignes extraites et faire abstractions des parasites potentiellement présents et détectés par le LIDAR, ce qui permettrait de gagner en rapidité et diminuerait la probabilité d'écraser les plantes.

3.3 Difficultés rencontrés

3.3.1 Client TCP

La première étape de la conception de l'application était la réalisation de la connexion TCP avec le simulateur. Pour cette réalisation, nous nous sommes basés sur un TP d'automatisme réalisé à l'ISTIA où nous manipulions un Client TCP codé en C++. Plusieurs essais et recherches nous ont menés à l'utilisation de la classe System.Net.Sockets.TcpClient nous permettant de réaliser une connexion simplement et d'utiliser son flux pour communiquer avec le simulateur.

Avant d'arriver à la version finale de notre partie connexion TCP, nous en avons réalisé plusieurs en utilisant diverses classes et programmes afin de mieux saisir son fonctionnement. De ce fait, l'étude des communications TCP et sa mise en place nous ont causées quelques difficultés au lancement du projet car nous n'en avions jamais réalisé auparavant, de plus ne connaissions pas non plus le fonctionnement du simulateur aussi bien que maintenant.

3.3.2 Trajectoire

L'élaboration des trajectoires, que nous réalisons dans cette application, nous a pris le plus de temps dans cette partie du concours. Les documentations techniques ne communiquaient pas les formules de calcules nécessaires aux commandes des moteurs, comme par exemple les rotations en ° fonction des nombres de pas commandés. De ce fait, nous avons dû réaliser de nombreux tests empiriques afin d'obtenir les nombres de pas nécessaires aux trajectoires souhaitées et obtenir des fonctions rudimentaires.

3.3.3 LIDAR

Nous avons dû revoir à plusieurs reprises le programme de la partie LIDAR, notamment pour la récupération des données LIDAR. Ce sont les problèmes de trames non lues les unes après les autres, expliqués dans « Récupération des données LIDAR par Thread » de la partie développement, qui nous ont poussés à utiliser un thread de lecture de trame en tâche de fond. Notre première méthode était de se connecter et se déconnecter au LIDAR à chaque fois qu'il était nécessaire d'obtenir les données pour l'IA. Malheureusement le simulateur n'étant pas assez robuste pour supporter des connexions/déconnexions toutes les 500 ms environs, nous avons dû changer de méthode et apprendre à utiliser le thread.

5 Innovation

5.1 Recherche d'innovation

5.1.1 Recherche d'idée :

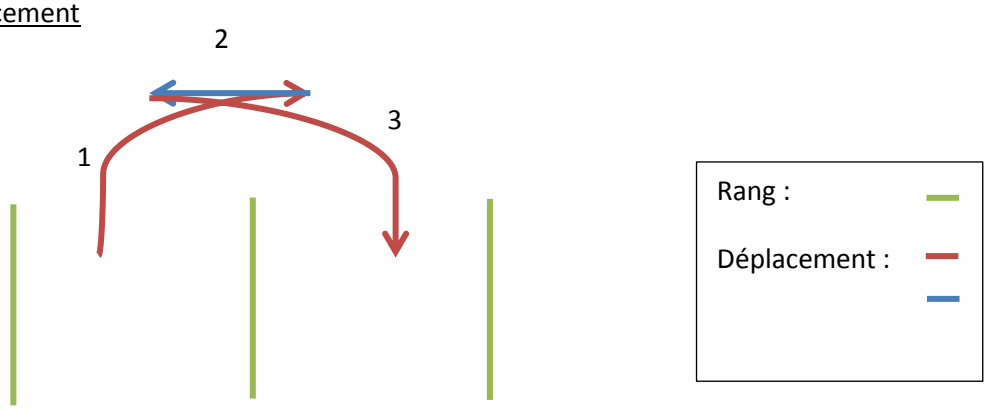
Dans le but de remplir les objectifs donnés par la troisième partie du concours, nous nous sommes attelés à la recherche d'innovations possibles. Nous avons commencé par faire des recherches sur les besoins possibles des agriculteurs. Une fois les besoins possibles définis, nous avons effectué un brainstorming chargé d'établir les solutions possibles. Lors de cet exercice, nous avons commencé par écrire toutes les idées qui nous venaient instantanément. Après avoir un nombre d'idées acceptables, nous avons commencé à approfondir ces idées en les précisant. Puis nous avons effectué un tri dans ces solutions afin d'en ressortir uniquement celles qui nous semblaient plausibles et réalisables. Dans un dernier temps, nous avons dues malheureusement abandonner des idées qui nous paraissaient faisables, mais qui était trop gourmande en temps et qui nous aurait été à coup sûr impossible à réaliser.

5.1.2 Idées sélectionnées

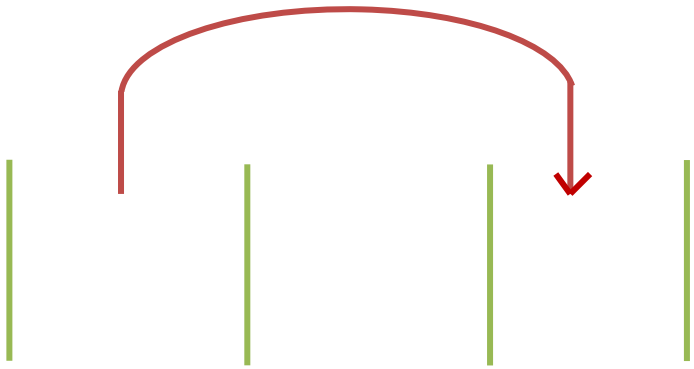
Finalement, nous n'avons retenu que deux idées. La première sur le déplacement du robot, la seconde sur l'envoi de sms. Une troisième idée a été suggérée par l'équipe enseignante et porte sur le LIDAR.

Le déplacement : Le déplacement de base du robot pendant sa phase de demi-tour était lente et complexe. Le demi-tour se faisait en trois phases. Lorsque le robot sortait du rang il devait avancer tout en décrivant un cercle jusqu'à atteindre 90°. Une fois la position atteinte, il devait faire une marche arrière sur une certaine distance. Une fois cette distance atteinte le robot faisait une marche avant en effectuant encore une fois un angle de 90° afin de rentrer dans le rang juste à côté. Nous avons modifié ceci en demandant au robot de faire un angle de 180° afin qu'il puisse slalomer dans le champ.

<u>Premier déplacement</u>



Déplacement amélioré



Le sms: Pour aider les agriculteurs et éviter une perte de temps inutile, nous avons décidé de donner l'ordre au robot de commencer son travail lorsqu'il reçoit un sms. Cette idée nous est venue lors du brainstorming .Au départ nous voulions que le robot aille tout seul dans la parcelle que l'utilisateur lui aurait envoyée par sms. Cependant, cette idée avait été écartée au vu de sa complexité notamment au niveau du code concernant le GPS. Nous avons donc décidé que l'utilisateur placerait le robot manuellement devant un rang du champ et qu'il enverrait la commande de démarrage plus tard via sms. Cela permet de faire des travaux différents sans perdre de temps puisque l'utilisateur n'a pas besoin de revenir dans le champ de départ.

Le LIDAR: le LIDAR est le capteur infrarouge du robot, l'équipe enseignante a trouvé utile, dans un but de recherche de voir les informations que le LIDAR a renvoyéES. Nous avons donc décidé d'établir une application Client/Serveur où le robot serait le serveur et enverrait en permanence les données du LIDAR qu'un client, un ordinateur réceptionnerait via le Wi-Fi ce qui nous permettrait de voir les données en temps réel.

5.2 Innovations entreprises

5.2.1 Commander le robot grâce au SMS

Pour une de nos innovations, nous avons décidé de rendre plus présente la gestion du robot à distance. Nous souhaitons donc lancer des actions en envoyant un SMS au robot.

Pour cela, nous avons modifié le code de la carte ArduinoIHM. Lorsque le robot est en mode attente de SMS, l'odroïd envoie un message à la carte ArduinoIHM lui demandant de vérifier qu'un SMS a été reçue. Si c'est le cas, le programme va lire le contenu du SMS. C'est à ce moment-là que nous modifions le code, nous allons vérifier si le mot « START » est présent dans le contenu du SMS. Le mot « START » est le mot à envoyer que nous avons choisi pour lancer une action :

```
if (checkSMS && millis() > lastRecSMS + 5000UL){
        char content[100];
        char number[20];
        int nbSMS;
        nbSMS = SMS.receiveSMS(content, number); //lit les SMS
        lastRecSMS = millis();
        {
                char log[100];
                sprintf(log,"%d nouv sms, num %s msg %s",nbSMS,number,content);
                sendLog(log,1);//passer cette priorité à 2
        }
        if (nbSMS > 0 && strlen(number) > 8){
                for (unsigned int i = 0;i<strlen(content);i++){ // met le message en majuscule
                        if (isalpha(content[i])){
                                content[i] = toupper(content[i]);
                        }
                }
                if (GSM.findSubstring(content,"START")){ // Vérifie qu'il y a le mot START
                        char msg[100];
                        sprintf(msg,"LAUNCH");
                        sendMSG(msg,true,commanditaire); //renvoie un message à l'odroïd
                        checkSMS = 0;
                }
        }
}
```

Une fois que nous avons reçu le bon message, nous envoyons un message à l'odroïd pour lui dire qu'il peut lancer l'action. Ici, nous allons lui demander de lire un fichier de configuration d'une parcelle afin de la biner. Nous allons donc nous intéressé à la fonction newInfoIHM(message msg) de la classe GestionIHM qui permet de décider des actions à suivre suite à la réception d'un message par la carte odroïd :

```
if (strncmp("LAUNCH",msg.msg,6)==0){
        message mymsg;
        parcelle p;
        int i=LoadConfFile::loadMaConf(p); // Lit le fichier de configuration
        // Initialise les paramètres pour le binage
        If (i != -1) { //Vérifie que nous avons récupéré des informations
        p_dec->modeAutoAct=MA_Binage;
        for(int j=1;j<=p.nbRangees;j++)
        {
                p_param->longAllee[j]=p.longAllee[j];
                p_param->largAllee[j]=p.largRangee[j];
                if(j<p.nbRangees) p_param->distDemiTour[j]=0.80;
        }
        p_param->distDemiTour[0]=0.80;
        p_param->nbRangees=p.nbRangees;
        p_param->largMax=p.largRangee[1];
        p_param->nbPassages=p.nbPass;
        p_param->sensDemiTour=p.sensDT;
        p_dec->alleeCur=1;
        p_dec->chkFin = 1;
        // Lance le binage
        p_dec->startBinage();
        p_dec->modeAct=M_Auto;
        }
}
```

Nous avons demandé la lecture du fichier par la fonction loadMaConf(parcelle p) de la classe LoadConfFile. Cette fonction nous permet de récupérer les informations d'une parcelle enregistrée dans un fichier : (code complet en annexe)

```
ifstream fichier("/home/odroid/OzCore/parcelle.txt", ios::in); //Nous ouvrons le fichier en lecture
//----------------------Récupération des infos du fichier--------------------------------
if (fichier) // Nous testons qu'on a réussi à ouvrir le fichier
{
        //Déclarations des variables nécessaire pour une parcelle
        string nom, slongueur, slargeur, sdistDT, coteDT, snbPassages;
        int nbRangs, nbPassages;
        int *longueur;
        double *largeur, *distDT;
        string ligne;
        while (getline(fichier, ligne))  // Nous récupérons les lignes du fichier
        {
                // Nous traitons les lignes pour récupérer les informations
        }
        fichier.close(); // Ferme le fichier
        return 3;
}
else
        return -1;
```

5.2.2 La supervision

Définition Client/serveur : L'environnement client-serveur désigne un mode de communication à travers un réseau entre plusieurs programmes ou logiciels : l'un, qualifié de client, envoie des requêtes, l'autre ou les autres, qualifiés de serveurs, attendent les requêtes des clients et y répondent. Par extension, le client désigne également l'ordinateur sur lequel est exécuté le logiciel client, et le serveur, l'ordinateur sur lequel est exécuté le logiciel serveur.

Notre supervision permet de voir les données renvoyées par le lidar. Notamment la distance à laquelle le lidar détecte les obstacles. Le serveur est codé en C++ afin de rester cohérent avec le code du robot qui est en C++ lui aussi. Ce code doit être exécutable sous linux car c'est sous cet environnement que nous compilons les différents codes du robot. Ce code se trouve dans la classe IA_Lidar du robot et est appelée dans le mode suivi du robot.

```
// ******************************************************
// Envoie des données au clients
// ******************************************************
buffer[0] = 0x49;
buffer[1] = 0x59;
buffer[2] = 0x69;

printf("***********Données lidar*************");
printf("Distance points:        Intensité: ");


for (int i = 3; i < 274; i++)
  {
        buffer[i] = p_lscan->mesures[i].dist;// On met dans le buffer les différentes distances enregistrées par le robot
  }


envoie = send(id_de_la_nouvelle_socket, buffer2, strlen(buffer2), 0);
```

Figure 15 :Code envoyé les données du serveur

La partie client permettra de lire les données du serveur sur une sortie console d'un ordinateur.

```
// ******************************************************
// Reception des données du seveur
// ******************************************************
   int nbrOctet = recv(id_de_la_socket, buffer, 828, 0);//Recpetion des données du serveur
   printf("\nBuffer reception :");
   printf("\nNombre d'octet recu : %d | -> %s <-", nbrOctet, buffer);
   printf("\n-------------------------\n");
```

Figure 16: Réception des données par le client

Afin de ne pas alourdir le rapport, le code du client et du serveur seront mis en annexes avec des commentaires explicatifs.

5.2.3 Bilan du travail réalisé

Lors de ce projet, nous avons été amenés à réfléchir sur l'ajout de nouvelles fonctions au robot. Ces fonctions sont toujours en cours de développement mais bien avancés, par exemple la commande

par SMS lance une parcelle mais de temps en temps cet appel entraine un redémarrage des cartes dû à une exception déclenchée. Pour la supervision, le système client -serveur fonctionne entre deux machines Windows mais le code du serveur a des problèmes de librairies sous Ubuntu.

Il nous reste donc à finaliser les derniers détails sur les codes et ainsi nos innovations seraient opérationnelles pour le concours.

5.3 Difficultés rencontrées

En développant la commande par SMS, nous avons rencontré le problème suivant : lorsque nous attendions la réception du SMS, et que la fonction plantée, la carte ArduinoIHM restée dans la boucle d'attente du SMS. Il était alors impossible de communiquer avec la carte odroïd, la seule solution était de couper l'alimentation des cartes via le disjoncteur présent sous la coque du robot. Nous avons également rencontré un problème avec la réception des SMS, parfois les SMS ne semblaient ne pas arriver jusqu'à la carte SIM du robot

Les problèmes rencontrés pendant la conception de la supervision ont été dans un premier temps l'élaboration du socket entre le client et le serveur. Le second problème a été le transfert entre linux et Windows. En effet ce problème persistant nous a empêché de finir ce projet. En effet même en changeant les librairies, linux nous en demandé toujours de nouvelles pour fonctionner. Nous avons eu un problème au niveau du robot en effet lors de l'implantation d'un programme. Nous ne pouvions plus accéder à aucunes fonctionnalités de celui-ci. Nous voulions au départ installé le serveur dans la fonction Core() de little Oz qui est la fonction qui s'exécute en permanence dans le robot. Cependant au vu du dernier problème nous avons décidé de le mettre dans une autre fonction afin de ne pas prendre le risque de provoqué un bug et être dans l'impossibilité de remodifier le programme par la suite et ainsi entrainer une erreur irrécupérable empêchant le fonctionnement du robot.

6 Communication

6.1 Site Web

L'un des objectifs des projets tuteurés est la réalisation d'un site internet permettant de mettre en valeur notre projet, notre expérience et nous-mêmes pour des futurs employeurs souhaitant en savoir davantage sur les projets que nous avons réalisés durant notre cursus scolaire. Le site a été réalisé à partir d'un bootstrap Template disponible en Open source sur le site *www.bootstrapzero.com* permettant ainsi de le modifier directement sans avoir de problème avec les droits d'auteur.

6.1.1 Structure

Notre site est équipé de 5 pages permettant de présenter le projet ainsi que l'équipe. La structure de notre site Web est fondée à partir de notre page d'accueil d'où découlent les autres pages.

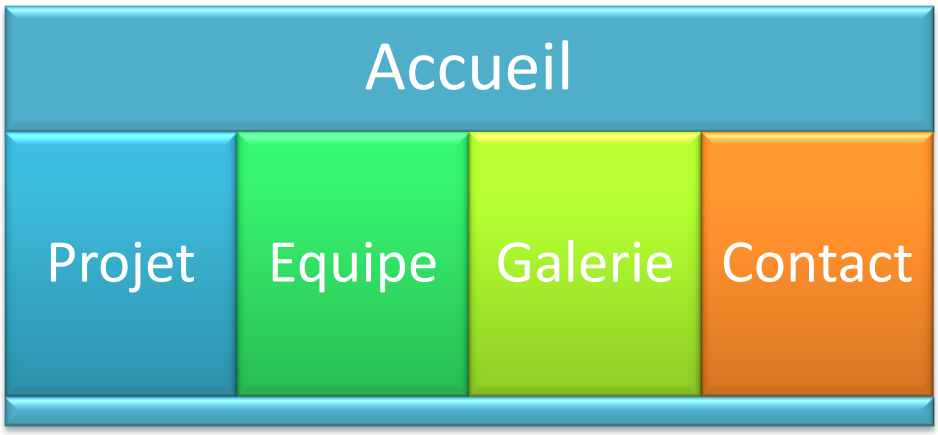


Figure 17 : Structure du site Web

Page d'accueil :

Il s'agit de la page principale du site, elle permet de faire une courte présentation du projet et de notre équipe. Cette page est découpée en 3 grandes parties :

Le projet : cette première partie permet au visiteur de se renseigner sur le projet, de regarder son avancement avec la galerie ou encore de prendre contact avec l'ISTIA ou NAIO Technologies avec leurs coordonnées en cliquant sur les boutons « *Détail* » situé sous chaque descriptif.

Figure 18 : Partie projet de la page d'accueil

L'équipe : la seconde partie est celle de l'équipe, il s'agit d'une courte présentation de chaque personne composant OZIRA, lorsque le visiteur souhaitera en savoir davantage sur une personne spécifique, il lui suffira de cliquer sur sa photo ou bien sur le bouton « *Détail* ».

Figure 19 : Partie Equipe de la page accueil.

La dernière partie n'est pas seulement visible sur la page d'accueil mais aussi sur les autres pages. Cette partie correspond à la bande publicitaire du site. Dans l'ordre, nous retrouvons un court résumé sur le projet, les logos des organismes qui ont permis la réalisation de ce projet et enfin la localisation de notre école, son histoire ainsi qu'un lien vers la page contact

Le projet :

Cette page permet d'expliquer le projet ainsi que les différentes parties du concours aux visiteurs du site, mais aussi d'accéder au rapport et à la soutenance, par des liens de téléchargement.

Figure 20 : Liens de téléchargement du compte-rendu et de la soutenance

L'équipe :

Cette page permet aux visiteurs d'en savoir plus sur les personnes composant notre équipe, leurs compétences, leurs formations et leurs expériences. Le CV de chaque étudiant est disponible en cliquant sur le bouton « *Télécharger* ».

Figure 21 : Contenu type de la présentation d'un participant

La galerie :

Les visiteurs peuvent voir le déroulement du projet à partir des photos et des vidéos situées dans la galerie.

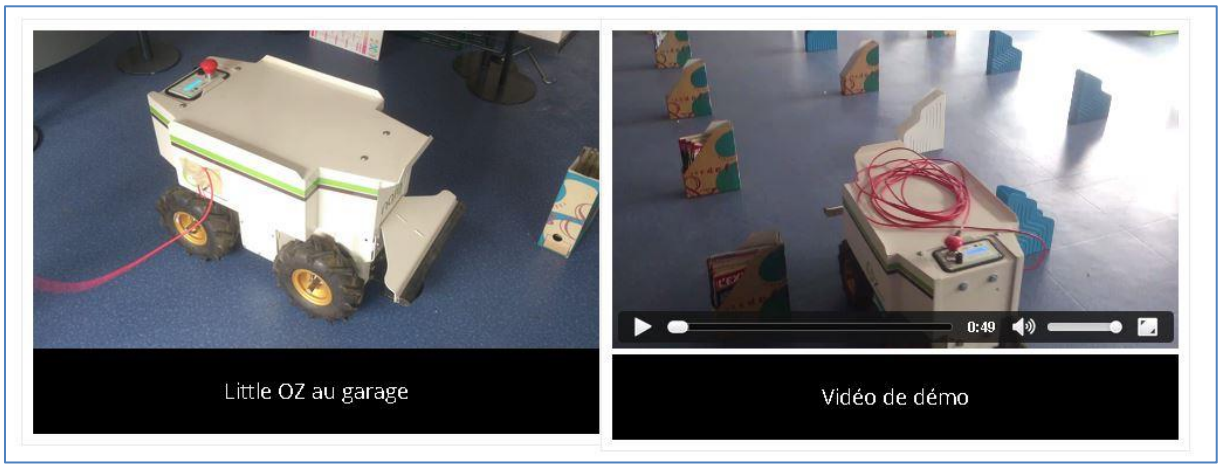


Figure 22 : Image et vidéo de la galerie

Contact :

Enfin, si les professionnels veulent en savoir plus, les coordonnées de l'ISTIA et de l'entreprise NAIO Technologies sont disponibles dans la page Contact. Un plan, provenant de l'application Google Map, permet aussi de visualiser l'emplacement de chaque établissement.

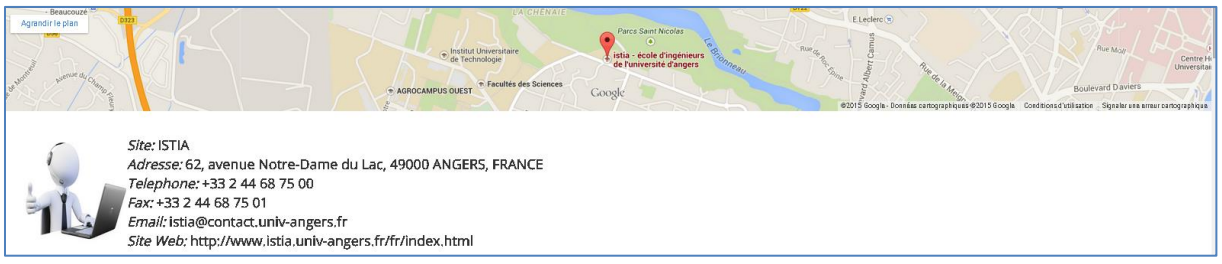


Figure 23 : Présentation des coordonnées de l'ISTIA situé dans le site Web

6.1.2 Technologies employées

6.1.2.1 *Langage*

Pour la réalisation du site, il était imposé de coder ce dernier en langage HTML, le Template disposait de base de quelques scripts CSS et JavaScript permettant la mise en forme. Les modifications que nous avons apportées au code source ont toutes été réalisées en HTML.

Figure 24 : Logo html

6.1.2.2 *Logiciel*

Pour pouvoir visualiser les modifications apportées aux codes sources du site internet, nous avons utilisé le logiciel WampServer qui est une plateforme de développement Web à l'aide du serveur Apache2.

Figure 25 : Logo WampServer

6.1.3 Autres fonctionnalités

6.1.3.1 *Téléchargement*

Notre site possède cette fonction qui permet aux visiteurs de télécharger le compte-rendu et la soutenance du projet, situé en milieu de la page « *Projet* », ou encore les CV des étudiants à partir de la page « *Équipe* ». Ses documents sont disponibles en cliquant sur les boutons « *Télécharger* ».

6.1.3.2 *Multilingue*

Avec la formation d'ingénieur et le programme d'échange à l'international, les étudiants de notre école doivent être capables de s'exprimer sans problème dans sa langue maternelle, mais aussi en anglais. Ainsi pour pouvoir montrer notre site à l'international, une option multilingue a été développée pour que notre site Web soit aussi disponible en anglais. Pour pouvoir modifier la langue de notre site, deux drapeaux, le drapeau français et le drapeau du Royaume-Uni, ont été placés dans notre barre de navigation. Cliquer sur un des deux drapeaux permet de transférer le visiteur vers le site dans la langue qu'il souhaite.

6.2 Echange autour du projet

Pendant le déroulement du projet, nous avons eu l'occasion d'avoir certains échanges sur notre travail, avec une entreprise spécialisée dans les capteurs infrarouges et des étudiants en première année de l'ISTIA.

6.2.1 Entretien avec SICK

SICK est une entreprise internationale, spécialisée dans la fabrication de capteurs intelligents pour des domaines comme l'automatisation d'une production, l'analyse industrielle ou encore l'automatisation de logistique. Cette entreprise, leader technologique et numéro un dans son domaine, a équipé le robot Little Oz avec son capteur

Figure 26 : Logo société SICK

LIDAR à infrarouge, capteur que nous utilisons principalement pour nos programmes. L'assistante de communication de la branche française a pris contact avec nous pour pouvoir nous interviewer afin de publier un article sur notre équipe et nos motivations, dans le journal de l'entreprise. Les questions concernaient principalement nos parcours ainsi que notre stratégie et si nous connaissions, avant que l'assistante de communication ne prenne contact avec nous, l'entreprise SICK.

6.2.2 Entretien avec les EI1

Durant le second semestre, les étudiants en première année doivent réaliser une expression écrite sur un projet proposé par leur professeur. Ainsi deux étudiants ont pris contact avec nous, pour réaliser leur présentation sur notre équipe, notre projet et le travail réalisé.

7 Gestion du projet

Aujourd'hui dans le monde professionnel, la moitié des projets aboutissent à des surcoûts, des dépassements des délais ou encore tout simplement, l'abandon du projet en lui-même. Pour assurer la réussite de notre projet, nous l'avons planifié de façon organisé et réfléchi. De plus, pour qu'un projet se réalise de façon cohérente, que chaque membre d'une équipe aille dans une même direction et pour prendre certaines décisions, un projet se doit d'avoir un « chef de projet ». Dans notre cas, nous avons élu Mathieu COLAS qui a accepté d'endosser ce rôle.

7.1.1 Cycle de vie

Afin de gérer notre projet le plus simplement possible mais de façon efficace, nous avons choisi un cycle de vie en spirale qui est un cycle de vie simple à construire mais robuste. Étant donné que nous sommes des étudiants ayant déjà réalisé quelques projets qui avaient pour sujet la programmation de robot, nous avions déjà quelques idées sur la façon de conduire notre projet. Ce cycle de vie, composée au total de 7 phases, nous permettait de réaliser à chaque fin de phase, une réunion pour parler de son déroulement et si les objectifs, que nous nous sommes donné, étaient atteints.

Notre cycle de vie s'est déroulé de la manière suivante :

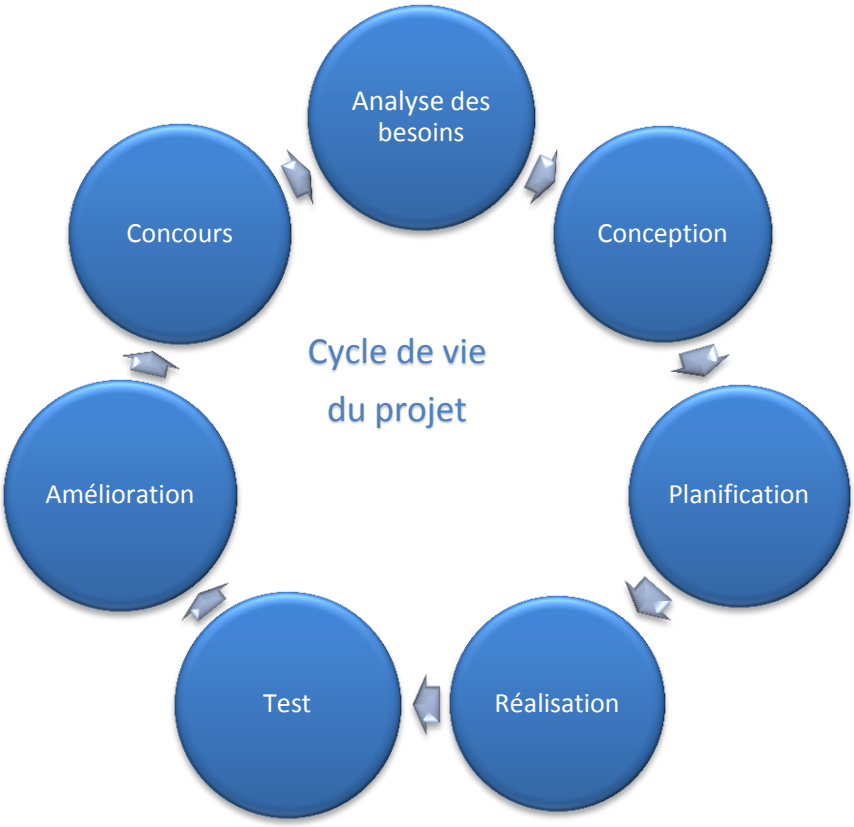


Figure 27 : Cycle de vie du projet

Analyse

Cette phase sert à l'analyse des besoins pour le concours et à l'analyse du robot en général.

Conception

Il s'agit de la phase essentielle d'un projet, elle nous a permis de concevoir notre stratégie pour pouvoir réaliser un binage rapide, précis et propre avant de se lancer directement dans la programmation. Il s'agit de la phase durant laquelle, nous avons réalisé notre cahier des charges et ainsi pouvoir définir les différents objectifs, de chaque phase, à atteindre. Et ainsi de s'organiser en conséquence.

Planification

Comme son nom l'indique, il s'agit de réaliser un planning prévisionnel pour pouvoir visualiser l'état d'avancement du projet.

Réalisation

Phase la plus longue du projet, elle est constituée de trois étapes : Apprentissage, recherche et programmation.

Test

La phase de test possède deux objectifs : le premier objectif est de vérifier que toutes les fonctionnalités du cahier des charges ont été implémentées, puis de repérer tous les bugs éventuels qui empêchent le programme de fonctionner correctement.

Amélioration

Cette phase, qui précède la phase de test, permet de corriger tous les bugs repérés auparavant et d'optimiser, si besoin le programme.

Concours

Il s'agit de la dernière phase, celle où nous allons participer au concours « *Move Your Robot !* » à Toulouse.

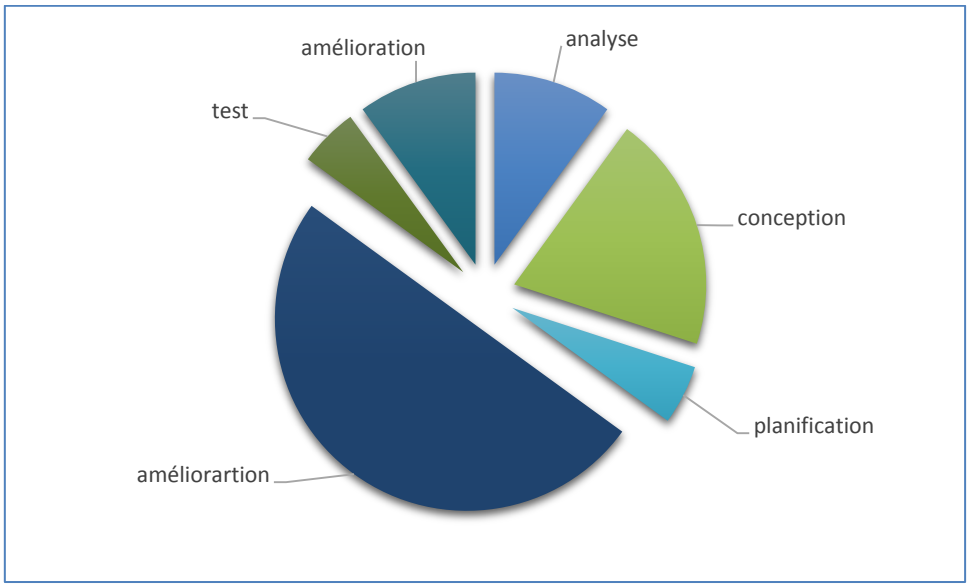
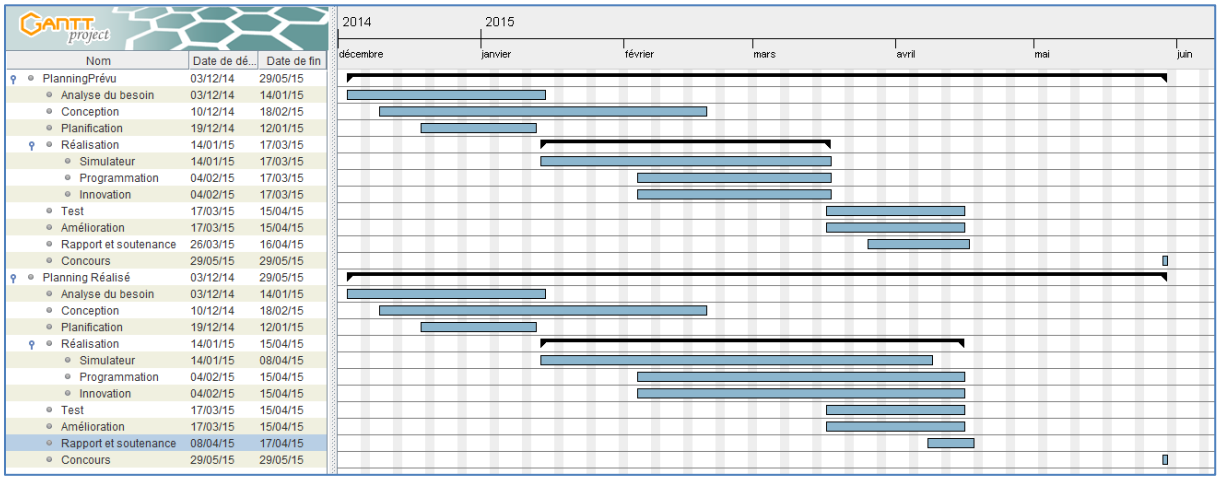


Figure 28 : Répartition temps de travail entre les différentes phases

7.1.2 Macro-planning



Nom	Date de dé...	Date de fin
PlanningPrévu	03/12/14	29/05/15
Analyse du besoin	03/12/14	14/01/15
Conception	10/12/14	18/02/15
Planification	19/12/14	12/01/15
Réalisation	14/01/15	17/03/15
Simulateur	14/01/15	17/03/15
Programmation	04/02/15	17/03/15
Innovation	04/02/15	17/03/15
Test	17/03/15	15/04/15
Amélioration	17/03/15	15/04/15
Rapport et soutenance	26/03/15	16/04/15
Concours	29/05/15	29/05/15
Planning Réalisé	03/12/14	29/05/15
Analyse du besoin	03/12/14	14/01/15
Conception	10/12/14	18/02/15
Planification	19/12/14	12/01/15
Réalisation	14/01/15	15/04/15
Simulateur	14/01/15	08/04/15
Programmation	04/02/15	15/04/15
Innovation	04/02/15	15/04/15
Test	17/03/15	15/04/15
Amélioration	17/03/15	15/04/15
Rapport et soutenance	08/04/15	17/04/15
Concours	29/05/15	29/05/15

Figure 29 : Plannings prévisionnel et réalisé

Nous pouvons constater, grâce au planning prévisionnel, que la phase de développement à durer plus de temps que prévu. Cela est dû au fait que nous avons étudié les codes déjà existants du robot et ainsi de savoir lequel pouvait être réutilisé. Mais en général, le planning réalisé suit fidèlement notre planning prévisionnel.

Concernant la répartition des tâches, chaque personne avait une tâche précise dans le projet. Cependant, il n'était pas impossible qu'un étudiant laisse sa tâche en suspend pour aller aider un autre étudiant permettant ainsi d'accélérer le développement, d'apporter de nouvelles idées ou encore de résoudre un problème.

7.1.3 Gestion des documents

La gestion de ce projet à générer de nombreux documents car, chaque semaine, un rapport a été réalisé dans le but de communiquer, à nos enseignants tuteurs, l'état d'avancement du projet. Au début du projet, une trame de gestion de projet nous a été remise afin de nous aider à organiser, avec soin et de façon précise, notre projet. Cette trame, nous a permis par la suite de générer un planning prévisionnel, mais aussi de créer un cahier des charges en employant la méthode du « QQOQCP » (Quoi, Qui, Où, Quand, Comment et Pourquoi).

Bilan personnel et technique

Alexis Teixeira :

Pour ma part, ce projet a été une expérience assez enrichissante, aussi bien dans le domaine du développement quant à la gestion du projet. Le travail d'équipe nous apprend l'humilité, à être à l'écoute, à échanger nos connaissances, mais aussi à s'adapter en fonction de chaque coéquipier.

Le projet « *Move Your Robot !* » m'a permis de me conforter dans mon choix concernant mon projet professionnel ainsi que mon choix d'option en EI5.

En terme de technique de développement, le projet « *Move Your Robot !* » m'a permis de perfectionner mes compétences en développement, mais aussi en gestion de projet. La réalisation d'un cahier des charges, la conception d'un projet et anticiper les choses sont des compétences que nous n'avons très rarement la chance de faire en tant qu'étudiant. Cela peut nous faire défaut lorsque nous rentrons dans le monde professionnel.

Mathieu Colas :

Grâce à ce projet, j'ai eu l'occasion de travailler sur un sujet correspondant parfaitement à mon objectif professionnel, celui de devenir ingénieur en recherche et développement en robotique et informatique. Ayant eu la charge, du développement complet de l'application « Simulation », j'ai significativement renforcé mes compétences en programmation, notamment en C#.

De plus, en tant que responsable de projet, je suis très satisfait du travail accompli jusqu'ici par notre équipe, nous avons réalisé le cahier des charges que nous nous étions fixé et avons été même au-delà. Ce projet m'a permis de m'autoévaluer sur ma capacité à travailler en équipe et à gérer un projet dans son intégralité.

Thomas Dorléans :

Voulant travailler dans le domaine de la robotique, ce projet était en adéquation avec mes attentes. Pour la première fois, j'ai pu réaliser un projet en école qui se rapproche du monde professionnel. Le travail en équipe a été bénéfique grâce à la cohésion du groupe, notre entente et notre bonne humeur nous ont permis de trouver des solutions rapidement lorsqu'un de nous était bloqué. Pour ma part ayant travaillé à la fois sur le simulateur et sur le robot, j'ai pu améliorer mes connaissances dans les langages C++ et C#. Dans l'ensemble, le bilan est très positif mon seul regret a été de ne pas avoir pu faire de mécanique et d'électronique sur le robot.

Yves Fierville :

Pour ma part, ce projet est une formidable expérience qui m'a permis mettre en pratique mes connaissances, il m'a également donné l'occasion de mettre en évidence mes principales lacunes en programmation. Ce projet m'a appris beaucoup de chose concernant la programmation d'un robot, comment structurer ces codes pour une meilleure efficacité dans le travail et la lecture des codes. Du point de vue de la gestion du projet, il est difficile de gérer le temps, en tant que débutant et cela m'a porté défaut pour déterminer le temps nécessaire pour réaliser tel ou tel action.

Sur un plan plus personnel, ce projet ma conforté dans mon projet professionnel de travaillé sur des systèmes automatisés, en robotique. Cela m'a également montré que même s'il nous pensions que nous avions assez de temps pour terminer le projet, il y a très souvent un imprévus qui va ralentir le projet.

Résumé

Notre projet EI4 était de préparer le concours "Move Your Robot", organisé par la société NAIO Technologies, un innovateur dans le robotique agricole avec leur robot Little Oz. La première des trois épreuves de la compétition, appelée "Simulation", était de réaliser un programme de binage autonome en C #, qui vise à contrôler une image du Little Oz sur un simulateur. Le programme a été évalué avec différents niveaux correspondant à des configurations typiques de rangées de plantes. Puis la deuxième épreuve, appelée "Plein champ " était également la programmation d'un programme autonome de binage, mais cette fois en C ++, sur Little Oz lui-même, et dans un environnement réel, les rangées de tests de NAIO Technologies. Enfin, la dernière étape du concours, nommée "Champ Libre" était une épreuve d'innovation dans lequel nous avons implémenté le lancement d'un cycle de binage par SMS, avec la technologie GSM. Ce projet correspond parfaitement avec notre objectif professionnel qui est de travailler dans le domaine de la robotique. De plus, nous avons eu l'occasion d'améliorer nos compétences en programmation, découvrir des technologies innovantes et acquérir une expérience réelle dans ce domaine.

Mots clés : Move your robot, programmation, technologie GSM, C++, C#

Abstract

Our EI4 project was to prepare the contest "Move Your Robot", organized by the company NAIO Technologies, an innovator in agricultural robotic with their Little Oz robot. The first of three events of the competition, called "Simulation", was to realize an autonomous hoeing program in C#, which aimed to control a simulated robot of Little Oz on a simulator. The program was rated with different levels matching with the typical configurations of vegetable rows. Then the second event, called "Plein champ" was also to programming an autonomous hoeing program, but this time in C++, on Little Oz itself, and in a real environment, the NAIO Technologies test rows. Finally the last stage of the contest, named "Champ Libre" was an innovation event in which we have implemented the launch of hoeing by SMS, with the GSM technology. This project matches perfectly with our professional goal which is to work in the robotic field. Moreover, we had the opportunity to improve our programming skills, discover innovative technologies and gain real experience in this field.

Keywords: Move your robot, programming, GSM technology, C++, C#

Table des illustrations

Annexe

Annexe 0 : ID des composants

Accelero Payload

id : 0x09

- 2 bytes : X, integer16 value, mG.

- 2 bytes : Y, integer16 value, mG.

- 2 bytes : Z, integer16 value, mG.

In simulator only Y axis is used, Z is fixed to 1000mG (1 G / 9.81m/s22).

Actuator Payload

id : 0x0C

- 1 byte : Value, Byte value :

// raise tool : 00000001

// lower tool : 00000010

// freeze tool : 00000011

Simulator listen to this socket, but value are not used.

Gps Payload

id : 0x04

- 8 bytes : time, Double value, gps time. ms since epoch.

- 8 bytes : Lat, Double value, gps latitude.

- 8 bytes : Lon, Double value, gps longitude.

- 8 bytes : Alt, Double value, gps altitude.

- 1 byte : Unit, byte value, gps unit used (meters there).

- 1 byte : NumberOfSat, byte value, number of satelite used by gps.

- 1 byte : Quality, byte value, 0 : no fix, 1 fixing, 2 : fix 2D, 3 : fix 3D, 6 super fix 3D.

- 8 bytes : GroundSpeed, double value, gps speed in km/h.

Gyro Payload

id : 0x0A

- 2 bytes : X, integer16 value, mDegree / s.

- 2 bytes : Y, integer16 value, mDegree / s.

- 2 bytes : Z, integer16 value, mDegree / s.

In simulator only Z.

The gain is 30.5 :

Simul :

Double degreeByMs = (degreeDiff * 1000.0 * secFactor) / 30.5;

Core :

gyr_raw_[0] = gyrData->x() * (30.5 / 1000);

gyr_raw_[1] = gyrData->y() * (30.5 / 1000);

gyr_raw_[2] = gyrData->z() * (30.5 / 1000);

Lidar Payload

id : 0x07

- 2 * 271 bytes : distance[271], integer16[271] value, lenght in mm of lidar ray with

degree n[0-270];

- 1 * 271 bytes : Albedo, byte value[271], albedo of obstacle.

You could ignore albedo.

Magneto Payload

id : 0x0B

- 2 bytes : X, integer16 value, Degree.

- 2 bytes : Y, integer16 value, Degree.

- 2 bytes : Z, integer16 value, Degree.

In simulator :

Double rad = Math.PI * this.Bearing / 180.0;

Double x = Math.Cos(rad) * 100;

Double y = Math.Sin(rad) * 100;

Double z = -23;

Motors Payload

id : 0x01

- 1 byte : LCM, sbyte value, left power [-127;127]

- 1 byte : RCM, sbyte value, right power [-127;127]

powers used are :

- 0 : stop

- 127 : full forward speed

- -127 : full backward speed

Odo Payload

id : 0x05

- 1 byte : FrontRight, byte value, 0 or 1

- 1 byte : RearRight, byte value, 0 or 1

- 1 byte : RearLeft, byte value, 0 or 1

- 1 byte : FrontLeft, byte value, 0 or 1

Each 6.465 cm run by a wheel, the tick status changes, 0 to 1, or 1 to 0, you cannot detect direction, you had to handle this with last motors command send, or noisy accelero/gyro.

Annexe 1 : guidage()

```
case MA_Binage: // Mode binage séléctionné
	switch (phaseAct){
		case P_OutilUp:
		{
		//Monter l'outil jusqu'en haut.
		err = OutilUp();
		// On ignore car on a pas d'outil
		//if (err == ERR_FIN){
		//Lancer la phase suivante (fin de travail ou demi-tour)
		dateDebutPhase.now();
		decFinRangee();
		//}
		}
		break;
		case P_OutilDown:
		{
		//Descendre l'outil jusqu'à la position cible.
		err = OutilDown();
		if (err == ERR_FIN){
			dateDebutPhase.now();
			if (!reprise){
			resetRangeeBinage();
			reprise = 0;
			}
			//Lancer la phase suivante (lancer la rangée)
			phaseAct = P_Rangee;
		}
		phaseAct = P_Rangee;
		}
		break;
		case P_Rangee:
			err = guidageRangee();
			//Ajuster la hauteur de l'outil si nécessaire
			if (p_ArdCMD->LastResynchro.elapsedMillis() < 5000 ||
(lastDegagMoustache.elapsedMillis() > 500 && lastDegagMoustache.elapsedMillis() < 5000)){
				ajustOutil();
			}
			//Test s'il y a un trou
			if (err == ERR_TROU_CULTURE){
			//Test si c'est la fin de la rangée
				if (isFinRangee()){
					err = ERR_FIN;
					sprintf(newmsg.msg,"INFO fin rangee~ ~");
					newmsg.sender = DEC;
					newmsg.reciever = IHM;
					p_ArdIHM->SendMSG(newmsg);
				}
			}
			if (err == ERR_FIN){
				dateDebutPhase.now();
				Move(OM_STOP);
				ajustementOutilEnCours = 0;
				phaseAct = P_OutilUp;
			}
			if (err == ERR_TROU_CULTURE){
				modeAct = M_Pause;
				Move(OM_STOP);
				p_gestIHM->IHMaskInfo(PP_Erreur);
			}
```

```
                        if (err == ERR_OBSTACLE || err == ERR_MOUSTACHE){
                                modeAct = M_Pause;
                                Move(OM_STOP);
                                p_gestIHM->IHMaskInfo(PP_Erreur);
                        }

                        if (err == ERR_ROUE_BLOQUEE || err == ERR_PATINAGE){
                                modeAct = M_Pause;
                                Move(OM_STOP);
                                p_gestIHM->IHMaskInfo(PP_Erreur);
                        }
                        //TODO:gérer les autres cas d'erreur
                break;
                case P_DemiTour:
                        err = guidageDemiTour();
                        newmsg.sender = DEC;
                        newmsg.reciever = IHM;
                        if (modeAct == M_Auto){
                                if (err == ERR_FIN){
                                        dateDebutPhase.now();
                                        Move(OM_STOP);
                                        ajustementOutilEnCours = 0;
                                        phaseAct = P_OutilDown;
                                        resetRangeeBinage();
                                }
                                if (err == ERR_DEMI_TOUR_RATE){
                                        modeAct = M_Pause;
                                        Move(OM_STOP);
                                        p_gestIHM->IHMaskInfo(PP_Erreur);
                                }
                        } else {
                                if (err != ERR_NONE){
                                        Move(OM_STOP);
                                        modeAct = M_Pause;
                                }
                        }
                break;
                default:
        }
break;
```

Annexe 2 :guidageDemiTour()

```
ERREUR Decision::guidageDemiTour(){
	ERREUR err;
	Consigne cons;
	err = p_ia_demitour->DemiTourAuto(&cons);
	if (p_ia_demitour->ResetNeeded){
		p_ia_demitour->ResetNeeded = 0;
		p_autom->Reset();
	}
	if (p_ia_demitour->MergeNeeded){
		p_ia_demitour->MergeNeeded = 0;
		sendMSG("MERGECI",ORDER);
	}

	sendMSG("KL 1 0",NONE);
	if(alleeCur==1 && passNumber==1)
	{
		if ((p_param->nbRangees)%2==0)// nb rangee paire
		{
			cons.side=sensPre;
		}
		else if(sensPre==C_DROITE)
		{
			cons.side=C_GAUCHE;
		}
		else cons.side=C_GAUCHE;
		Move(p_autom->turnPID(cons,(double)p_param->largAllee[alleeCur]));
	}
	else if (((p_param->nbRangees)%2==0 && alleeCur== ((p_param->nbRangees)/2)+1))
//Petit Virage test si paire
	{
		Move(p_autom->turnPID(cons,(double)p_param->largAllee[alleeCur]));
	}
	else if ((p_param->nbRangees)%2==1 && alleeCur== ((p_param->nbRangees)/2)+2)//Petit
Virage test si impaire
	{
		if(cons.side==C_DROITE)
		{
			cons.side=C_GAUCHE;
		}
		else cons.side=C_DROITE;
		Move(p_autom->turnPID(cons,(double)p_param->largAllee[alleeCur]));
	}
	else
	{
		//Grand virage
		Move(p_autom->turnPID(cons,2.*(double)p_param->largAllee[alleeCur]));
	}
	//estimation de l'erreur angulaire
	double estimAngle = p_ia_demitour->calcReestimAngle();
	if (estimAngle < 90){
		char ordreAngle[100];
		sprintf(ordreAngle,"CORR_ANGLE %d",(int)estimAngle);
		sendMSG(ordreAngle,ORDER);
	}
	sensPre=cons.side;
	return err;
}
```

Annexe 3 : loadMaConf(parcelle &p)

```
int LoadConfFile::loadMaConf(parcelle &p)
{
ifstream fichier("/home/odroid/OzCore/parcelle.txt", ios::in);
//----------------------Récupération des infos du fichier----------------------------
if (fichier)
{
	string nom, slongueur, slargeur, sdistDT, coteDT, snbPassages;
	int nbRangs, nbPassages;
	int *longueur;
	double *largeur, *distDT;
	string ligne;
	while (getline(fichier, ligne))  // tant que l'on peut mettre la ligne dans
"contenu"
	{
		int posPoint1 = ligne.find(',', 0);
		int posPoint2 = ligne.find(',', posPoint1 + 1);
		int posInter = 0;
		//On récupère le nom
		nom = ligne.substr(0, posPoint1);
		posInter = nom.find('=', 0);
		nom = nom.substr(posInter + 1, nom.size() - (posInter + 1));
		if (nom == "parcelle01")
		{
			strcpy(p.nom,nom.c_str());
			//Récupération du nombre de Rangs
			string SnbRangs = (ligne.substr(posPoint1 + 1, posPoint2 - (posPoint1
+ 1)));
			posInter = SnbRangs.find('=', 0);
			SnbRangs = SnbRangs.substr(posInter + 1, SnbRangs.size() - (posInter +
1));
			nbRangs = atoi(SnbRangs.c_str());
			p.nbRangees=nbRangs;
			//------ déclaration des tableaux de taille=nbRangs
			longueur = new int[nbRangs];
			largeur = new double[nbRangs];
			distDT = new double[nbRangs-1];
			//-----------On remplit les tableaux
			//--------Récupération des longueurs
			for (int i = 1; i <= nbRangs; i++)
			{
				posPoint1 = posPoint2;
				posPoint2 = ligne.find(',', posPoint1+1);
				slongueur = ligne.substr(posPoint1 + 1, posPoint2 - (posPoint1
+ 1));
				posInter = slongueur.find('=', 0);
				longueur[i] = atoi(slongueur.substr(posInter + 1,
slongueur.size() - (posInter + 1)).c_str());
				p.longAllee[i]=longueur[i];
			}
			//----------Récupération des largeurs
			for (int i = 1; i <= nbRangs; i++)
			{
				posPoint1 = posPoint2;
				posPoint2 = ligne.find(',', posPoint1 + 1);
				slargeur = ligne.substr(posPoint1 + 1, posPoint2 - (posPoint1 +
1));
				posInter = slargeur.find('=', 0);
				largeur[i] = stod(slargeur.substr(posInter + 1, slargeur.size()
- (posInter + 1)).c_str());
				p.largRangee[i]=largeur[i];
			}
```

```
                        //------------Récupération des distances demi-tour
                        for (int i = 1; i < nbRangs; i++)
                        {
                                posPoint1 = posPoint2;
                                posPoint2 = ligne.find(',', posPoint1 + 1);
                                sdistDT = ligne.substr(posPoint1 + 1, posPoint2 - (posPoint1 +
1));
                                posInter = sdistDT.find('=', 0);
                                //Bug sur cette ligne
                                //distDT[i] = atof(sdistDT.substr(posInter + 1, sdistDT.size()
- (posInter + 1)).c_str());
                                p.distDemiTour[i]=distDT[i];
                        }
                        posPoint1 = posPoint2;
                        posPoint2 = ligne.find(',', posPoint1 + 1);
                        coteDT = ligne.substr(posPoint1 + 1, posPoint2 - (posPoint1 + 1));
                        posInter = coteDT.find('=', 0);
                        coteDT = coteDT.substr(posInter + 1, coteDT.size() - (posInter + 1));
                        if (coteDT=="C_DROITE") p.sensDT= C_DROITE;
                        else p.sensDT= C_GAUCHE;
                        snbPassages = ligne.find(',', posPoint2 + 1);
                        snbPassages = ligne.substr(posPoint2 + 1, ligne.size() - (posPoint2 +
1));
                        posInter = snbPassages.find('=', 0);
                        nbPassages = atoi(snbPassages.substr(posInter + 1, snbPassages.size()
- (posInter + 1)).c_str());
                        p.nbPass=nbPassages;
                }

        }
        fichier.close(); // on ferme le fichier
        return 3;
}
else
        return -1;
}
```

Annexe3 : trameNAIO

```
class TrameNAIO
{
  private byte _id;
  private byte[] _matrame;// A changer en private en l'associant avec Retourtrame()
  private int _taille;

  public TrameNAIO( byte ID ){
    _id = ID;

    switch(ID){
      case 0x01 :_matrame = new byte[17];break;
        // Accelero Payload
      case 0x09:_matrame = new byte[21];break;
        // Bearing Payload
      case 0x0B:_matrame = new byte[21];break;
      // Gps Payload
      case 0x04:_matrame = new byte[58];break;
      // Gyro Payload
      case 0x0A:_matrame = new byte[21];break;
      // Lidar Payload
      case 0x07:_matrame = new byte[828];break;
    }
    _taille = _matrame.Length;
    InitTrame(_matrame);
  }

  public TrameNAIO(byte[] TRAME){
      _id = 0x07;
      _matrame = TRAME;
      _taille = _matrame.Length;
  }

  private void InitTrame(byte[] trame){
        byte[] Bytes = new byte[4];//converti les la taille qui est en int en byte
        for (int j = 0; j < 4;j++ )
        {
            Bytes[j] = (byte)(TailleData >> (j * 8)); //converti la taille base(10) en
base(16) byte
        }
        // On remplie la trame avec les lettres NAIO01
        trame[0] = 0x4E; // N
        trame[1] = 0x41; // A
        trame[2] = 0x49; // I
        trame[3] = 0x4F; // O
        trame[4] = 0x30; // 0
        trame[5] = 0x31; // 1
        trame[6] = _id; // ID
        trame[7] = Bytes[3]; // Taille de la trame
        trame[8] = Bytes[2]; // avec la taille
        trame[9] = Bytes[1]; // convertie
        trame[10] = Bytes[0];//
        trame[_taille - 4] = 0x00; // CRC
        trame[_taille - 3] = 0x00; // CRC
        trame[_taille - 2] = 0x00; // CRC
        trame[_taille - 1] = 0x00; // CRC
  }

  public int TailleTrame{
      get { return _taille; }
  }
```

```
    public int TailleData{
        get { return _taille - 15; }
    }

    public byte ValeurOctet(int index){
        return _matrame[index];
    }

    public byte ValeurData(int index){
        return _matrame[index+11];
    }

    public byte[] GetTrame(){
        return _matrame;
    }

    public void SetTrame(int index, byte value){
        _matrame[index] = value;
    }

    public string Affiche(){
      string TRAME="";
      for (int i =0; i<_taille;i++)
      {
        TRAME += _matrame[i];
      }
      return TRAME;
    }
}
```

Annexe 4 : connexion TCP

```
//Test une connexion sur les moteurs, si réussi "OK" sinon "Erreur"
      labelTest.ForeColor = Color.Black;
      labelTest.Text = "...";
      try // essai de ...
      {
        TcpClient tcpclnt = new TcpClient(); // instancie un socket TCP
        tcpclnt.Connect(textBoxIP.Text, 3331); // connexion : IP simulateur + port
moteurs
        labelTest.ForeColor = Color.Green;
        labelTest.Text = "Ok"; // Afficher Ok à l'utilisateur
        tcpclnt.Close(); // ferme le socket
        buttonConnexion.Enabled = true; // déverrouille le bouton de connexion

      }
      catch (Exception ex) // Si échoué
      {
        labelTest.ForeColor = Color.Red;
        labelTest.Text = "Erreur"; // Afficher Erreur à l'utilisateur
        MessageBox.Show(ex.ToString()); // Indiquer le message des raisons de l'échec
      }



try // essayer
      {
        tcpclntMOTORS.Connect(textBoxIP.Text, 3331); // connexion aux moteurs
        stmMOTORS = tcpclntMOTORS.GetStream(); // acquisition du flux de communication
        tcpclntLIDAR.Connect(textBoxIP.Text, 3337); // connexion au LIDAR
        stmLIDAR = tcpclntLIDAR.GetStream(); // acquisition du flux de communication
        label2.Text = "Connexions ok";
        if (!_etatThread) // si le thread de récupération des données LIDAR n'est pas
déjà lancé
        {
          bwProgress.RunWorkerAsync(); // lancer le thread
          _etatThread = true; // mémorisation que le thread est lancé
        }
        buttonEtat.Enabled = true; // autorisation du mode manuel
        buttonAV.Enabled = true; //
        buttonGD.Enabled = true; //
        buttonAR.Enabled = true; //
        buttonPos.Enabled = true;
        buttonLibre.Enabled = true;
      }
      catch (Exception ex)
      {
        MessageBox.Show(ex.ToString()); // si échoué, afficher la raison
      }
```

Annexe 5 : boutons manuels

```
private void buttonAV_Click(object sender, EventArgs e)
    {
      envoi_commande_moteur(byte.Parse(((255-(NumAV.Value*(-1)- 1)) %
256).ToString()),
                            byte.Parse(((255 - (NumAV.Value * (-1) - 1)) %
256).ToString()));
    }

private void buttonAR_Click(object sender, EventArgs e)
{
  envoi_commande_moteur(byte.Parse(((255 - (NumAV.Value - 1)) % 256).ToString()),
                        byte.Parse(((255 - (NumAV.Value - 1)) % 256).ToString()));
}
private void buttonGD_Click(object sender, EventArgs e)
{
  envoi_commande_moteur(byte.Parse(((255 - (NumGauche.Value * (-1) - 1)) %
256).ToString()),
                        byte.Parse(((255 - (NumDroite.Value * (-1) - 1)) %
256).ToString()));
}
```

Annexe 6 : Commande moteur

```
try
      {
        // instancie une trame au format moteurs
        TrameNAIO trameMoteurs = new TrameNAIO(0x01);
        // Assigne le nombre de pas demandé pour les roues de gauche
        trameMoteurs.SetTrame(11, MLEFT);
        // Assigne le nombre de pas demandé pour les roues de droite
        trameMoteurs.SetTrame(12, MRIGHT);
        // affiche la commande envoyée à l'utilisateur
        textBoxCommande.Text = "Moteur Gauche = " + (MLEFT).ToString() +" ; "+
                               "Moteur Droit = " + (MRIGHT).ToString();
        //envoi la commande des moteurs via le flux de communication
        stmMOTORS.Write(trameMoteurs.GetTrame(), 0, trameMoteurs.TailleTrame);
      }
      catch (Exception ex)
      {
        // affiche le cas échéant
        textBoxCommande.Text = "Erreur... " + ex.StackTrace;
      }
```

Annexe 7 : Récupération des informations LIDAR

```
TrameNAIO TramePreThread = new TrameNAIO(0x07); // instancie une trame NAIO Lidar
while (!bwProgress.CancellationPending) // Qu'on ne demande pas l'arret du thread
{
  int k = 0;
  do
  {
    // Lit le flux de données du LIDAR et le copie dans la trame NAIO Lidar
    k = stmLIDAR.Read(TramePreThread.GetTrame(), k, TramePreThread.TailleTrame - k);
  } while (stmLIDAR.DataAvailable); // Tant que des données sont disponibles en
lecture
  if (DemandeLIDAR == true) // Si on demande les dernières données LIDAR
  {
    //Charge la trame NAIO Lidar avec les données les plus récentes
    TramePostThread = TramePreThread;
    DemandeLIDAR = false; // Accuser la demande
  }
  Thread.Sleep(20); // temporisation
}
```

Annexe 8 : Mise en forme des données LIDAR

```
TrameNAIO trameLIDARBrute = RecupTrameLIDAR(); // récupère la trame des valeurs du
LIDAR
int[] LIDAR = new int[271]; // tableau des valeurs numériques du LIDAR 0°-270°
//Parcours des octets du LIDAR sur deux octets, moins la qualité du signal (inutile
sur simulateur)
for (int i = 0; i < (trameLIDARBrute.TailleData - 271); i++)
{
  if (i % 2 == 1) // Si impaire, octect de poids faible
  {
    LIDAR[(i - 1) / 2] += trameLIDARBrute.ValeurData(i); // converstion octet vers int
  }
  else // sinon paire, otect de poinds fort
  {
    // décalage de 8 bits vers la droite pour obtenir la valeur réelle et conversion
octet vers int
    LIDAR[i / 2] += trameLIDARBrute.ValeurData(i) << 8;
  }
}
return (LIDAR); // renvoi le LIDAR en 0°-271°, avec distance en mm
```

Annexe 9 : Traitement des données LIDAR

```
bool _Devant = false;
int[] LIDAR = GetLIDAR(); // récupère le LIDAR 0°-271°, distance en mm
List<double> PosG = new List<double>(); //Liste des détections à gauche
List<double> PosD = new List<double>(); // Liste des détections à droite
listBoxObj.Items.Clear(); // Nettoyage de la liste des objets détectés

for (int i = 0; i < LIDAR.Length - 2; i++) //parcours du LIDAR 0°-271°, 44°-225° pour
obtenir 0°-180°
{
if ((44 < i) && (i < 115)) // détection de 70° à gauche
{
  if (LIDAR[i] < LargeurRang && ((LargeurRang - LargeurRobot) / 2) < LIDAR[i])
// si l'obstacle est dans le champs de vision définit (largeur de rang)
  {
    if (LIDAR[i+1] < LargeurRang && ((LargeurRang - LargeurRobot) / 2) < LIDAR[i+1])
// si l'obstacle est dans le champs de vision définit (largeur de rang)
    {
      if (LIDAR[i+2] < LargeurRang && ((LargeurRang - LargeurRobot) / 2) < LIDAR[i+2])
// si l'obstacle est dans le champs de vision définit (largeur de rang)
      {
        //Ajoute l'obstacle détectés à gauche dans la liste de gauche
        PosG.Add((Math.Cos(((i - 45) * Math.PI) / 180)) * LIDAR[i]);
      }
    }
  }
}
else if ((115 < i) && (i < 155) && !checkBoxRang.Checked) // détection de 30° devant
{
  if (LIDAR[i] < LargeurRobot && 0 < LIDAR[i]) // si l'obstacle est dans le champs de
vision définit (largeur de rang / 2)
  {
    if (LIDAR[i + 1] < LargeurRobot && 0 < LIDAR[i + 1]) // si l'obstacle est dans le
champs de vision définit (largeur de rang / 2)
    {
      if (LIDAR[i + 2] < LargeurRobot && 0 < LIDAR[i + 2]) // si l'obstacle est dans
le champs de vision définit (largeur de rang / 2)
      {
        _Devant = true; textBoxInfo.Text = "Obstacle détecter devant.";
      }
    }
  }
}
else if ((155 < i) && (i < 225)) // détection de 75° à droite
{
  if (LIDAR[i] < LargeurRang && ((LargeurRang - LargeurRobot) / 2) < LIDAR[i]) // si
l'obstacle est dans le champs de vision définit (largeur de rang)
  {
    if (LIDAR[i + 1] < LargeurRang && ((LargeurRang - LargeurRobot) / 2) < LIDAR[i+1])
// si l'obstacle est dans le champs de vision définit (largeur de rang)
    {
      if (LIDAR[i+2] < LargeurRang && ((LargeurRang - LargeurRobot) / 2) < LIDAR[i+2])
// si l'obstacle est dans le champs de vision définit (largeur de rang)
      {
        //Ajoute l'obstacle détectés à droite dans la liste de droite
        PosD.Add((Math.Cos(((225 - i) * Math.PI) / 180)) * LIDAR[i]);
      }
    }
  }
}
}
```

Annexe 10 :Déplacement dans une rangée

```
if (PosD.Count == 0) { PosD.Add(400); textBoxInfo.Text = "Ajout PosD"; }
if (PosG.Count == 0) { PosG.Add(400); textBoxInfo.Text = "Ajout PosG"; }
if (PosD[0] == 400 && PosG[0] == 400){
  textBoxInfo.Text = "PosD = 0 et PosG = 0";
  envoi_commande_moteur(byte.Parse("38"), byte.Parse("38"));
  if (Sortie > 3){
    Etat = "SortieDeRangée1"; // changement d'état en "SortieDeRangée1"
    Sortie = 0;
  }
  else{
    Sortie++;
  }
}
else if (_Devant == true){
  envoi_commande_moteur(byte.Parse("0"), byte.Parse("0"));
  textBoxInfo.Text = "Obstacle devant !";
}
else if ((PosD.Min() > LargeurRobot + 75) || (LargeurRobot + 75 < PosG.Min())) // si
le centre du robot se trouve à plus de 475 mm du bord coté droit
{
  if (Direction == "Droite"){
    if (PosD.Min() > LargeurRobot + 75){
      textBoxInfo.Text = "Trop loins de Droite";
      envoi_commande_moteur(byte.Parse("127"), byte.Parse("72")); // alors se
rapprocher du bord droit
    }
    else if (LargeurRobot + 75 < PosG.Min()){
      textBoxInfo.Text = "Trop loins de Gauche";
      envoi_commande_moteur(byte.Parse("72"), byte.Parse("127")); // tourner se
rapprocher du bord gauche
    }
  }
  else{
    if (LargeurRobot + 75 < PosG.Min()){
      textBoxInfo.Text = "Trop loins de Gauche";
      envoi_commande_moteur(byte.Parse("72"), byte.Parse("127")); // tourner se
rapprocher du bord gauche
    }
    else if (PosD.Min() > LargeurRobot + 75){
      textBoxInfo.Text = "Trop loins de Droite";
      envoi_commande_moteur(byte.Parse("127"), byte.Parse("72")); // alors se
rapprocher du bord droit
    }
  }
}
else if ((PosD.Min() < LargeurRobot - 100) || (LargeurRobot - 100 > PosG.Min())) // si
le centre du robot se trouve à moins de 300 mm coté droit
{
  if (Direction == "Droite"){
    if ((PosD.Min() < LargeurRobot - 100)){
      textBoxInfo.Text = "Trop près de Droite";
      envoi_commande_moteur(byte.Parse("72"), byte.Parse("127")); // alors s'éloigner
du coté droit
    }
    else if ((LargeurRobot - 100 > PosG.Min())){
      textBoxInfo.Text = "Trop près de Gauche";
      envoi_commande_moteur(byte.Parse("127"), byte.Parse("72")); // alors s'éloigner
du coté gauche
    }
  }
```

```
  else{
    if ((LargeurRobot - 100 > PosG.Min())){
      textBoxInfo.Text = "Trop près de Gauche";
      envoi_commande_moteur(byte.Parse("127"), byte.Parse("72")); // alors s'éloigner
du coté gauche
    }
    else if ((PosD.Min() < LargeurRobot - 100)){
      textBoxInfo.Text = "Trop près de Droite";
      envoi_commande_moteur(byte.Parse("72"), byte.Parse("127")); // alors s'éloigner
du coté droit
    }
  }
}
else{
  textBoxInfo.Text = "Au milieu";
  envoi_commande_moteur(byte.Parse("127"), byte.Parse("127")); // sinon c'est que le
robot est bien positionné dans la rangée donc avancer vitesse max
}
```

Annexe 11 : Changement de rangée

```
if (CurrentRang == _NbrPetGra){
  type = "Petit";
}
else{
  type = "Grand";
}
if (checkBoxRang.Checked){
  type = "Petit";
}
if (EstPaire){
  if (type == "Petit")  {
    if (EstDansRangeeSimple())    {
      Etat = "DansRangée";
      if (type == "Grand") { CurrentRang += 2; }
      else { CurrentRang += 1; }
      if (CurrentRang != _NbrPetGra)      {
        if (Direction == "Droite") { Direction = "Gauche"; }
        else { Direction = "Droite"; }
      }
      return;
    }
  }
  else{
    if (EstDansRangeeSimple1()){
      Etat = "DansRangée";
      if (type == "Grand") { CurrentRang += 2; }
      else { CurrentRang += 1; }
      if (CurrentRang != _NbrPetGra)      {
        if (Direction == "Droite") { Direction = "Gauche"; }
        else { Direction = "Droite"; }
      }
      return;}
  }
}
else{
  if (type == "Petit")  {
    if (EstDansRangeeSimple())    {
      if (CurrentRang != _NbrPetGra)      {
        if (Direction == "Droite") { Direction = "Gauche"; }
        else { Direction = "Droite"; }
      }
      Etat = "DansRangée";
      if (type == "Grand") { CurrentRang += 2; }
      else { CurrentRang += 1; }
      return;
    }
  }
  else  {
    if (EstDansRangeeSimple1())    {
      if (CurrentRang != _NbrPetGra)      {
        if (Direction == "Droite") { Direction = "Gauche"; }
        else { Direction = "Droite"; }
      }
      Etat = "DansRangée";
      if (type == "Grand") { CurrentRang += 2; }
      else { CurrentRang += 1; }
      return;
    }
  }
}
```

Annexe 12 : EstDansRangée / EstDansRangée1

```
if ((PosD.Count >= 5 || PosG.Count >= 5) && ProcDetec > 13) // si le robot est dans la
nouvelle rangée
{
  ProcDetec = 0;
  return (true); // return true "oui"
}
else
{
  ProcDetec++;
  return (false); // return false "non"
}

//*****************

if (ProcDetec > 22)
{
  envoi_commande_moteur(byte.Parse("127"), byte.Parse("127"));
  return (true);
}
else if ((PosD.Count >= 5 || PosG.Count >= 5) && ProcDetec == 20) // si le robot est
dans la nouvelle rangée
{
  ProcDetec = 0;
  return (true); // return true "oui"
}
else
{
  ProcDetec++;
  return (false); // return false "non"

}
```

Annexe : Partie serveur

```
/*****************************************************************************/
#include <winsock2.h>
#include <cstdio>
#pragma comment(lib,"ws2_32.lib")
/*****************************************************************************/
void serveur::EnvoieTrame()
{
  WSADATA initialisation_win32; // Variable permettant de récupérer la structure d'information sur l'initialisation
  int erreur; // Variable permettant de récupérer la valeur de retour des fonctions utilisées
  int tempo; // Variable temporaire de type int
  int nombre_de_caractere; // Indique le nombre de caractères qui a été reçu ou envoyé
  int envoie;
  char buffer[65535]; // Tampon contenant les données reçues ou envoyées
  char buffer2[65535];
  SOCKET id_de_la_socket; // Identifiant de la socket
  SOCKET id_de_la_nouvelle_socket; // Identifiant de la nouvelle socket
  SOCKADDR_IN information_sur_la_source; // Déclaration de la structure des informations lié à l'écoute


  //int main(int argc, char* argv[])
  //{
  printf("\nBonjour, vous etes du cote serveur\n");

  // ********************************************************
  // Initialisation de Winsock
  // ********************************************************
  erreur = WSAStartup(MAKEWORD(2, 2), &initialisation_win32);
  if (erreur != 0)
    printf("\nDesole, je ne peux pas initialiser Winsock du a l'erreur : %d %d", erreur, WSAGetLastError());
  else
    printf("\nWSAStartup  : OK");

  // ********************************************************
  // Ouverture d'une Socket
  // ********************************************************
  id_de_la_socket = socket(AF_INET, SOCK_STREAM, 0);
  if (id_de_la_socket == INVALID_SOCKET)
    printf("\nDesole, je ne peux pas creer la socket du a l'erreur : %d", WSAGetLastError());
  else
    printf("\nsocket      : OK");

    // ********************************************************
    // Activation de l'option permettant d'activer l'algorithme de Nagle
    // ********************************************************
    tempo = 1;
    erreur = setsockopt(id_de_la_socket, IPPROTO_TCP, TCP_NODELAY, (char *)&tempo, sizeof(tempo));
    if (erreur != 0)
      printf("\nDesole, je ne peux pas configurer cette options du à l'erreur : %d %d", erreur, WSAGetLastError());
    else
      printf("\nsetsockopt  : OK");

    // ********************************************************
    // Lie la socket à une ip et un port d'écoute
    // ********************************************************
    information_sur_la_source.sin_family = AF_INET;
    information_sur_la_source.sin_addr.s_addr = INADDR_ANY; // Ecoute sur le routeur
    information_sur_la_source.sin_port = htons(7575); // Ecoute sur le port 7575
    erreur = bind(id_de_la_socket, (struct sockaddr*)&information_sur_la_source, sizeof(information_sur_la_source));
    if (erreur != 0)
      printf("\nDesole, je ne peux pas ecouter ce port : %d %d", erreur, WSAGetLastError());
    else
      printf("\nbind        : OK");

    // ********************************************************
    // Attente d'ouverture de session
    // ********************************************************
    erreur = 99; // Initiation de erreur pour être sur que l'on va rentrer dans la boucle
    while (erreur != 0) // Boucle tant qu'une demande de session (SYN) tcp n'a pas été reçu
      erreur = listen(id_de_la_socket, 1);
    printf("\nlisten      : OK");
```

```
// *******************************************************
// Acceptation de la demande d'ouverture de session
// *******************************************************
printf("\nAttente de la reception de demande d'ouverture de session tcp (SYN)");
tempo = sizeof(information_sur_la_source); // Passe par une variable afin d'utiliser un pointeur
id_de_la_nouvelle_socket = accept(id_de_la_socket, (struct sockaddr*)&information_sur_la_source, &tempo);
if (id_de_la_nouvelle_socket == INVALID_SOCKET)
  printf("\nDesole, je ne peux pas accepter la session TCP du a l'erreur : %d", WSAGetLastError());
else
  printf("\naccept      : OK");

// *******************************************************
// Envoie des données au clients
// *******************************************************
buffer[0] = 0x49;
buffer[1] = 0x59;
buffer[2] = 0x69;

printf("***********Données lidar*************");
printf("Distance points:         Intensité: ");


for (int i = 3; i < 274; i++)
  {
        buffer[i] = p_lscan->mesures[i].dist;// On met dans le buffer les différentes distances enregistrées par le robot
  }


envoie = send(id_de_la_nouvelle_socket, buffer2, strlen(buffer2), 0);


// *******************************************************
// Fermeture de la session TCP Correspondant à la commande connect()
// *******************************************************
erreur = shutdown(id_de_la_nouvelle_socket, 2); // 2 signifie socket d'émission et d'écoute
if (erreur != 0)
  printf("\nDesole, je ne peux pas fermer la session TCP du a l'erreur : %d %d", erreur, WSAGetLastError());
else
  printf("\nshutdown    : OK");
// *******************************************************
// Fermeture de la socket correspondant à la commande socket()
// *******************************************************
erreur=closesocket(id_de_la_socket);
if (erreur!=0)
  printf("\nDesole, je ne peux pas liberer la socket du a l'erreur : %d %d",erreur,WSAGetLastError());
else
  printf("\nclosesocket : OK");

// *******************************************************
// Quitte proprement le winsock ouvert avec la commande WSAStartup
// *******************************************************
erreur=WSACleanup(); // A appeler autant de fois qu'il a été ouvert.
if (erreur!=0)
  printf("\nDesole, je ne peux pas liberer winsock du a l'erreur : %d %d",erreur,WSAGetLastError());
else
  printf("\nWSACleanup  : OK");

getchar();
system("PAUSE");
}
```

Annexe : Client

```
// ******************************************************
// Les includes
// ******************************************************
#include <winsock2.h> // pour les fonctions socket
#include <cstdio> // Pour les Sprintf

// ******************************************************
// Les librairies
// ******************************************************
#pragma comment(lib,"ws2_32.lib")

// ******************************************************
// Définition des variables
// ******************************************************


WSADATA initialisation_win32; // Variable permettant de récupérer la structure d'information sur l'initialisation
int erreur; // Variable permettant de récupérer la valeur de retour des fonctions utilisées
int tempo; // Variable temporaire de type int
int nombre_de_caractere; // Indique le nombre de caractères qui a été reçu ou envoyé
char buffer[65535]; // Tampon contennant les données reçues ou envoyées
char buffer2[65535];
SOCKET id_de_la_socket; // Identifiant de la socket
SOCKET id_de_la_nouvelle_socket;
SOCKADDR_IN information_sur_la_destination; // Déclaration de la structure des informations lié au serveur


int main (int argc, char* argv[])
  {
  printf("\nBonjour, vous etes du cote client\n");

  // ******************************************************
  // Initialisation de Winsock
  // ******************************************************
  erreur=WSAStartup(MAKEWORD(2,2),&initialisation_win32);
  if (erreur!=0)
    printf("\nDesole, je ne peux pas initialiser Winsock du a l'erreur : %d %d",erreur,WSAGetLastError());
  else
    printf("\nWSAStartup  : OK");

  // ******************************************************
  // Ouverture d'une Socket
  // ******************************************************
  id_de_la_socket=socket(AF_INET,SOCK_STREAM,0);
  if (id_de_la_socket==INVALID_SOCKET)
    printf("\nDesole, je ne peux pas creer la socket du a l'erreur : %d",WSAGetLastError());
  else
    printf("\nsocket      : OK");

  // ******************************************************
  // Activation de l'option permettant d'activer l'algorithme de Nagle
  // ******************************************************
  tempo=1;
  erreur=setsockopt(id_de_la_socket,IPPROTO_TCP,TCP_NODELAY,(char *)&tempo,sizeof(tempo));
  if (erreur!=0)
    printf("\nDesole, je ne peux pas configurer cette options du à l'erreur : %d %d",erreur,WSAGetLastError());
  else
    printf("\nsetsockopt  : OK");
```

```
// ********************************************************
// Etablissement de l'ouverture de session
// ********************************************************
information_sur_la_destination.sin_family=AF_INET;
information_sur_la_destination.sin_addr.s_addr=inet_addr("10.0.0.1"); // Indiquez l'adresse IP de votre serveu
information_sur_la_destination.sin_port=htons(7575); // Port écouté du serveur (7575)
erreur=connect(id_de_la_socket,(struct sockaddr*)&information_sur_la_destination,sizeof(information_sur_la_des
if (erreur!=0)
  printf("\nDesole, je n'ai pas pu ouvrir la session TCP : %d %d",erreur,WSAGetLastError());
else
  printf("\nsetsockopt  : OK");

// ********************************************************
// Reception des données du seveur
// ********************************************************
   int nbrOctet = recv(id_de_la_socket, buffer, 828, 0);//Recpetion des données du serveur
   printf("\nBuffer reception :");
   printf("\nNombre d'octet recu : %d | -> %s <-", nbrOctet, buffer);
   printf("\n-------------------------\n");

// ********************************************************
// Fermeture de la session TCP Correspondant à la commande connect()
// ********************************************************
erreur=shutdown(id_de_la_socket,2); // 2 signifie socket d'émission et d'écoute
if (erreur!=0)
  printf("\nDesole, je ne peux pas fermer la session TCP du a l'erreur : %d %d",erreur,WSAGetLastError());
else
  printf("\nshutdown    : OK");

// ********************************************************
// Fermeture de la socket correspondant à la commande socket()
// ********************************************************
erreur=closesocket(id_de_la_socket);
if (erreur!=0)
  printf("\nDesole, je ne peux pas liberer la socket du a l'erreur : %d %d",erreur,WSAGetLastError());
else
  printf("\nclosesocket : OK");
```