Mathieu LENORMAND 2013/2014
I.M.A. (Informatique Microélectronique Automatique)
POLYTECH' Lille
Ecole Polytechnique Universitaire de Lille

Mémoire de P.F.E
Projet de Fin d'Etude

Association de capteurs pour La navigation autonome d'un robot mobile Robotino A l'aide d'un module Northstar

Tuteur enseignant :
M Rochdi MERZOUKI

Remerciements

Je remercie M Rochdi MERZOUKI, de m'avoir encadré pendant mon projet de fin d'étude.

Je remercie M Olivier SCRIVE, de m'avoir mis à disposition la salle Robotino, d'avoir répondu à mess questions et de m'avoir aidé pour des problèmes de connexions.

Je remercie les doctorants Achille MELINGUI et Othman LAKHAL, pour m'avoir donné des idées de testes à faire sur la précision et d'avoir discuté avec moi de son projet et du mien.

Introduction

But du projet :

L'optimisation la navigation autonome d'un robot mobile à l'aide de différents capteurs.

Contexte du projet :

Pour mon projet de l'année dernière, j'ai utilisé les fonctions de traitement d'image de RobotinoView2, le logiciel de Festo pour programmer le Robotino. La même année, j'ai utilisé lors des Travaux Pratiques de Robotique, la fonction d'odométrie du Robotino. L'odométrie est une technique permettant d'estimer la position d'un véhicule en mouvement. Le terme vient du grec hodos (voyage) et metron (mesure). Cette mesure est présente sur quasiment tous les robots mobiles, grâce à des capteurs embarqués permettant de mesurer le déplacement du robot (de ses roues). Le principe de l'odométrie repose sur la mesure individuelle des déplacements des roues, en comptant les incrémentations du moteur, pour reconstituer le mouvement global du robot. En partant d'une position initiale connue et en intégrant les déplacements mesurés, on peut ainsi calculer à chaque instant la position courante d'un robot mobile.

C'est pourquoi cette année, j'ai eu l'opportunité d'utiliser le module Northstar qui est un système de localisation indoor acquis par Polytech'Lille, comparable à un système de géo-localisation dans un plan. Ce système de navigation associe un projecteur et un capteur infrarouge.

Il est intéressant d'associer différents capteurs (Odomètre, gyroscope et module Northstar) pour répondre à l'optimisation la navigation autonome d'un robot mobile.

Développement

Remerciements 2
Introduction 3
Développement 4
I) Description du contexte du projet 5
1) Cahier des charges 5
2) Présentation du robot mobile 5
2-a) Présentation de Festo 5
2-b) Description du Robotino 6
3) Description du module Northstar 7
3-a) Kit Northstar 7
3-b) Projecteur Northstar 7
3-c) Capteur Northstar 8
3-d) Bloc sur RobotinoView2 9
3-e) Interface du bloc 9
II) Première partie : Phases du Projet sur RobotinoView 10
1) Recherches des paramètres d'initialisation du module Northstar 10
1-a) Programme de paramétrage 10
1-b) Programme d'enregistrements de parcours 11
2) Etude des enregistrements avec Matlab 11
3) Association des capteurs pour une meilleur précision du positionnement 12
III) Seconde partie : Phases du Projet sur Matlab 13
1) Transcription du programme sur RobotinoView sur Matlab 13
1-a) Correspondance entre les blocs sur RobotinoView et les fonctions sur Matlab 13
1-b) Construction d'un programme pour commander un Robotino sur Matlab 15
1-c) Programme principal 16
1-d) Programme d'Interface Homme-Machine (IHM) 20
2) Amélioration de la recherche du paramétrage d'initialisation 22
2-a) Programme de paramétrage par dichotomie 22
2-b) Programme de paramétrage pas à pas 25
3) Résultats, robustesse et améliorations 25
3-a) Résultats 25
3-b) Robustesse : prise en compte globale par un coefficient ou un correcteur 26
3-c) Amélioration par l'utilisation d'autres capteurs 27
IV) Gestion de projet 28
1) Planning prévisionnel 28
2) Planning réel 28
Conclusion 30
Annexe 31

I) Description du contexte du projet

1) Cahier des charges

Objectif principal :

Tester l'association de capteurs sur une trajectoire quelconque.

Appliquer cela à des cas concrets, d'abord des trajectoires assez simples, sur des problèmes tels que le patinage ou le glissement des roues qui créent des erreurs avec l'odométrie. Puis en associant mes résultats au projet d'un doctorant qui travail sur l'évitement d'obstacle avec le Robotino par la logique floue.

Objectifs liés au module Northstar :

- Comprendre le fonctionnement du module Northstar.
- Savoir paramétrer et initialiser le projecteur et le capteur.
- Prendre en mains le bloc associé à Northstar sur RobotinoView2.
- Etudier la précision du module Northstar (selon les consignes, la vitesse, la lumière...)
- Rendre le positionnement dans l'espace le plus précis possible, après ces étapes, c'était l'objectif en associant les valeurs de x, y et phi calculées par l'odométrie et par le module Northstar le plus précis possible.

Contraintes :

- Le terrain
- La sensibilité des capteurs
- La vitesse
- La fiabilité

Moyens d'actions :

Les capteurs déjà utilisés :
- Un odomètre
- Un gyroscope

Un nouveau capteur :
- Un module Northstar

2) Présentation du robot mobile

2-a) Présentation de Festo

Festo est un fournisseur mondial (car présent dans 176 pays) de technologies d'automatisation et un leader en matière de formation. Leur but est d'apporter à leurs clients une productivité et une compétitivité maximales.

La branche Festo France a été créée en 1958 et implantée en région parisienne (Bry-sur-Marne, 94), elle propose différentes gammes de services (tels que des bureaux d'études, une division didactique et un centre de relation clientèle) à but professionnel ou éducatif.

Festo développe des robots en s'inspirant de la nature, appelé « Bionic Learning Network », ce système d'étude utilise des principes naturels pour concevoir des applications technologiques et industrielles.

Par exemple : the Bionic Handling Assistant qui est un bras bionique qui s'inspire d'une trompe d'éléphant dont un prototype a été fourni à Polytech.

Dans un but pédagogique, Festo a aussi développé un robot mobile, le Robotino, avec lequel c'est fait le projet.

2-b) Description du Robotino

Robot mobile : Châssis rond en inox et trois unités de motorisation omnidirectionnelles
- Diamètre : 370 mm
- Hauteur (habillage compris) : 210 mm
- Poids total : environ 11 kg

Capteurs :
- 1 Structure de protection en caoutchouc intégrant un détecteur anticollision
- 9 capteurs de distance analogiques à infrarouge
- 1 capteur inductif analogique
- 2 capteurs optiques numériques
- 1 caméra Web couleur à interface USB et compression jpeg

Commande :
- 1 PC embarqué avec OS Linux Ubuntu temps réel et plusieurs interfaces de communication :
- Ethernet
- 2 ports USB
- 2 liaisons RS232
- 1 port parallèle
- 1 connecteur VGA
- 1 Point d'accès Wi-Fi avec antenne, que l'on peut faire basculer en mode client et avoir un Cryptage WPA2 en option.

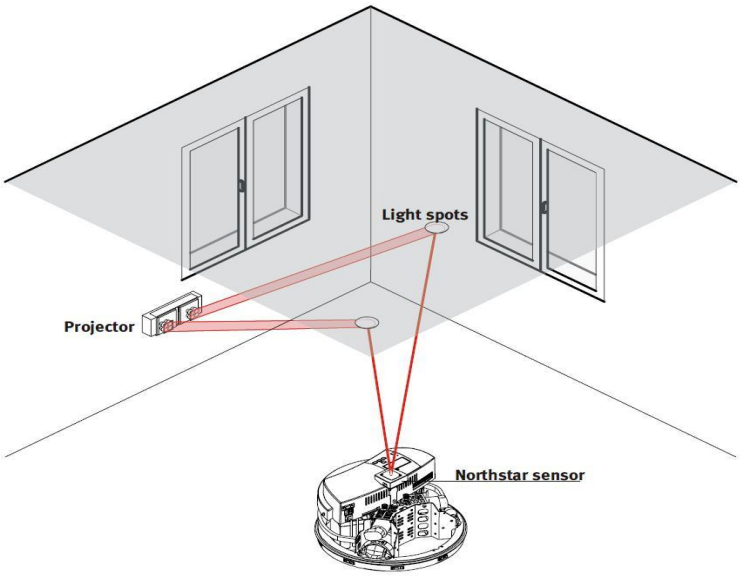
Platine de commande EA09 qui permet :
- Le pilotage de quatre moteurs à courant continu
- L'interface Ethernet pour accès externe direct à la régulation des moteurs
- L'intégration de composants électriques additionnels à l'aide de deux connecteurs d'E/S à 20 broches

RobotinoView2, le logiciel de programmation du Robotino conçu par Festo, dont la configuration requise est un PC avec Win 2000/XP SP2/VISTA/Windows 7. Le Robotino peut être programmé avec d'autres logiciels (tels que Matlab) et en utilisant d'autres langages de programmation comme le C++ et Java.

Ce robot peut être utilisé pour des Travaux Pratiques, des projets ou des concours comme le Festo Robotino Hockey Challenge Cup.

3) Description du module Northstar

Schéma de fonctionnement :



Le module Northstar est un Kit additionnel au Robotino disponible chez Festo.

L'accessoire système de navigation NorthStar Robotino est un système de localisation infrarouge qui utilise des points lumineux infrarouges comme points de repère. Un capteur infrarouge détermine sa position et son orientation à partir de la position de deux points lumineux. Les données d'évaluation et le contrôle des points lumineux infrarouges détectés sont inclus en tant que module fonctionnel dans le logiciel RobotinoView2. Le système de navigation NorthStar est constitué d'un projecteur et d'un capteur NorthStar. Grâce à cela, il peut être localisé dans un espace de travail d'un maximum de 4 x 4 m. Des Robotinos supplémentaires peuvent être ajoutés au système de navigation grâce à des capteurs supplémentaires NorthStar.

3-a) Kit Northstar

- 1 capteur Northstar
- 1 câble USB pour le relier le capteur au Robotino
- 1 projecteur avec un trépied
- 1 bloc d'alimentation électrique adaptable aux prises internationales
- 1 câble pour relier le projecteur au bloc d'alimentation
- 1 manuel d'utilisation (allemand et anglais)

3-b) Projecteur Northstar

Il s'agit d'une source lumineuse à infrarouge qui produit un motif lumineux spécial sur une surface réfléchissante

Le projecteur NorthStar est une source lumineuse infrarouge qui produit un modèle lumineux spécial sur une surface réfléchissante, telle qu'un plafond. Ce modèle lumineux peut être détecté et évalué par le capteur NorthStar.
Il existe 2 types :

Type NS1	Type NS2
 ProjectorKit NS1	 Projector NS2

Le Kit que j'ai utilisé pour mon projet comprenait le projecteur de type NS2.

Selon le type de projecteur, on a différents réglages possibles qui sont données dans la documentation technique ou l'aide en ligne qui nous fournit ce tableau:

Room ID	Projector type	Switch-Setting	Frequency [Hz]
0	ProjectorKit NS1	Spot 1 SW2 = 0 Spot 2 SW2 = 8	1040 2350
1	ProjectorKit NS1	Spot 1 SW2 = 1 Spot 2 SW2 = 9	1150 2450
2	ProjectorKit NS1	Spot 1 SW2 = 2 Spot 2 SW2 = A	1250 2560
3	**Projector NS2-50Hz**	1	**Spot A 3025** **Spot B 3925**
4	**Projector NS2-50Hz**	2	**Spot A 3125** **Spot B 4025**
5	**Projector NS2-50Hz**	3	**Spot A 3225** **Spot B 4125**

Chacun des projecteurs envoient 2 spots lumineux de couleurs rouges que l'on voit pendant 2 secondes pour régler leur position. Ensuite ces points passent dans le domaine des infra rouge et donc invisible à l'œil nu mais ils sont détectable par le capteur du module Northstar. Pour que les points lumineux redeviennent visibles il suffit de toucher une nouvelle fois le cercle en métal situé sur le projecteur.

Les informations du tableau donnent les valeurs en Hertz du signal selon la configuration choisie.

Pour chaque configuration, il faut régler le calibrage du projecteur, nommé switch-setting dans le tableau et que l'on peut voir sur cette image :

L'autre paramètre à régler est le numéro du local, nommé Room ID dans le tableau et qui se définit dans le logiciel avec le bloc Northstar.

Bilan des configurations possibles :
- calibrage du projecteur (switch-setting)=1 et numéro du local (Room ID sur le bloc Northstar)=3
- calibrage du projecteur (switch-setting)=2 et numéro du local (Room ID sur le bloc Northstar)**=**4
- calibrage du projecteur (switch-setting)=3 et numéro du local (Room ID sur le bloc Northstar)**=**5

3-c) Capteur Northstar

Le capteur NorthStar est fixé au Robotino au moyen d'une fixation de montage fournie. Pour calibrer, il suffit de saisir la distance entre le capteur et le plafond pour atteindre une précision de positionnement d'environ +/- 50 mm dans l'espace de travail.

3-d) Bloc sur RobotinoView2

Le bloc permettant de paramétrer et d'obtenir des résultats du module Northstar se présente ainsi :

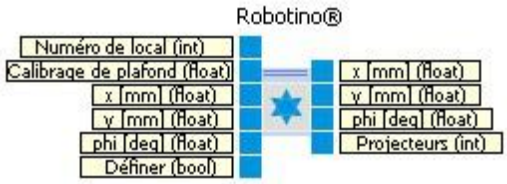
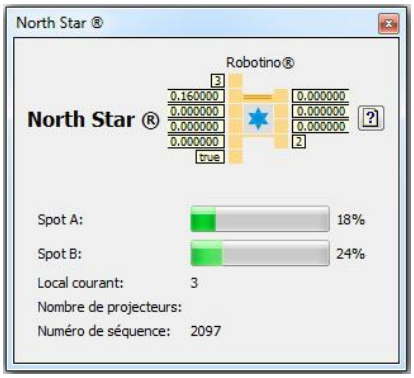


Tableau de définition des Entrées/Sorties:

Entrées	**Type**	**Unité**	**Standard**	**Description**
Numéro de local	int		3	Le numéro du local dans lequel Robotino se trouve
Calibrage du plafond	float	mm	2800	Distance entre le détecteur et le plafond
X	float	mm	0	Position x de l'origine définie par l'entrée "Définir"
Y	float	mm	0	Position y de l'origine définie par l'entrée "Définir"
Phi	float	Degré	0	Orientation de l'origine définie par l'entrée "Définir"
Définir	float		false	Si vrai (true) la pose (x,y,phi) est prise pour origine
Sorties	**Type**	**Unité**	**Standard**	**Description**
X	float	mm		La position x courante
Y	float	mm		La position y courante
Phi	float	Degré		L'orientation courante
Projecteurs	int			Nombre de projecteurs visibles

3-e) Interface du bloc



Le bloc permet est cliquant dessus d'obtenir une interface qui se présente sur cette image.

Cette interface reprend les valeurs d'Entrées/Sorties du bloc Northstar et d'autres informations :

- Le pourcentage de détection des 2 spots : Spot A et Spot B
- Le numéro du local courant
- Le nombre de projecteurs détecté par le capteur
- Le numéro de séquence qui correspond au nombre de fois où le capteur à tenter de capter les spots depuis le lancement du programme.

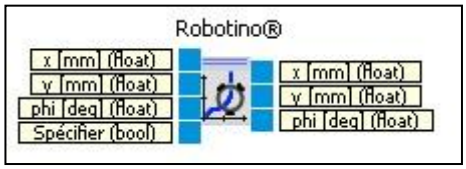
II) Première partie : Phases du Projet sur RobotinoView

1) Recherches des paramètres d'initialisation du module Northstar

1-a) Programme de paramétrage

Pour définir le paramétrage, j'ai écris un sous programme sous RobotinoView2. Le principe est qu'une ou plusieurs consignes sont stockés dans 3 tableaux, un pour x, un pour y et un pour phi. Ces valeurs sont associées à des valeurs réelles obtenues par le capteur d'Odométrie sont envoyées à un bloc parcoureur de position. Ce bloc à pour but de calculer l'écart entre ces valeurs et donner une valeur de vitesse Vx, Vy et Omega qui seront ensuite transmis aux 3 blocs moteur.

En parallèle dès que le bloc d'Odométrie est lancé, le bloc Northstar l'est aussi. En cochant l'affichage des valeurs de connexion, on peut voir l'évolution en temps réelle des valeurs d'Odométrie et de Northstar pour les comparer en particulier au moment de l'atteinte de la consigne.



La configuration choisie pour le projecteur est :
Calibrage du projecteur (switch-setting)=1 et Numéro du local (Room ID sur le bloc Northstar)=3
Cette configuration vient du tableau de la partie I) 3-b) qui définit à quelle fréquence le projecteur et le capteur fonctionnent.

Le but de ce programme est d'obtenir la valeur de calibrage du plafond, c'est cette valeur qui influence le plus la précision des valeurs du capteur Northstar.

Dans le manuel, la valeur de calibrage du plafond correspond à la différence entre la hauteur du plafond moins la hauteur d'un Robotino, exprimé en millimètre.

Cependant la contrainte est que cette valeur n'est pas toujours celle correspondant à la meilleure précision.

Pour ce calibrage, j'ai utilisé 2 configurations de hauteur de plafond :

- La première, où la table substituait le plafond

Les avantages sont vu que la hauteur est plus basse, on obtient un meilleur pourcentage de détection des spots (entre 80 et 100 %).
Les inconvénients sont que le champ d'action du Robotino est réduit.

- La deuxième, où le programme utilisait réellement le plafond

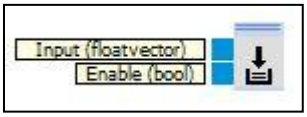
Les avantages sont d'avoir un champ d'action du Robotino plus large qui n'est plus limité par les dimensions de la table, mais par la surface d'utilisation du capteur Northstar.
Les inconvénients sont que les valeurs de pourcentage de détection des spots sont plus faibles (entre 5 et 25 %), mais cela n'empêche pas l'utilisation de ce capteur.

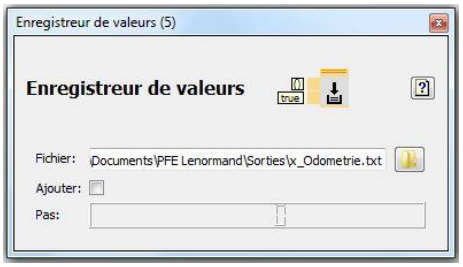
L'autre contrainte de ce calibrage est que la valeur de calibrage pour une même hauteur varie entre 2 Robotino, j'ai testé 2 Robotinos (172.26.94.17 et 172.26.94.16) dû à la logistique et à l'utilisation pour des TPs, ce qui m'a permis de constaté cela.

1-b) Programme d'enregistrements de parcours

Comme le calibrage est important pour avoir une bonne précision. J'ai modifié ce programme qui avait toujours pour but d'obtenir la valeur de calibrage du plafond, en rajoutant 6 blocs Enregistreur de valeurs. Le principe de bloc est quand il est actif de noté toutes les valeurs qu'il obtient dans un document texte. Cela permet d'obtenir 6 fichiers texte, dont on peut choisir le chemin de stockage, dans un dossier Résultats :

- x_Northsyar
- y_Northstar
- phi_Northstar
- x_Odometrie
- y_Odometrie
- phi_Odometrie





Ces 6 fichiers servent à définir le calibrage, mais après des discussions, ils peuvent aussi servir à étudier Etudier la précision du module Northstar (selon les consignes, la vitesse...)

2) Etude des enregistrements avec Matlab

Pour faire cette étude, j'ai écris un programme sous Matlab, nommé Analyse_Sorties.m et décri en annexe. Il a pour but d'obtenir des graphiques, des valeurs, des écarts selon les parcours.

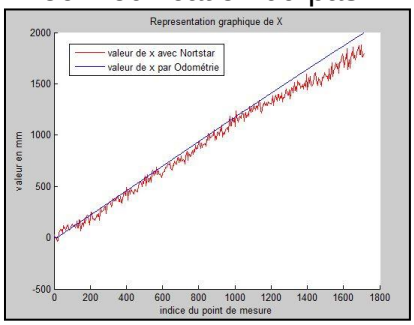
Ce programme utilise la fonction textread pour récupérer les 6 fichiers textes et les définit dans 6 vecteurs. Il utilise aussi les fonctions figure, plot, title, xlabel et ylabel pour définir les graphiques. Et il utilise les fonctions mean, min et max pour calculer les écarts entre les courbes.

Les contraintes sont dues la vitesse de fonctionnement des capteurs, car en moyenne le capteur par Odométrie est 3 fois plus rapide le capteur Northstar.

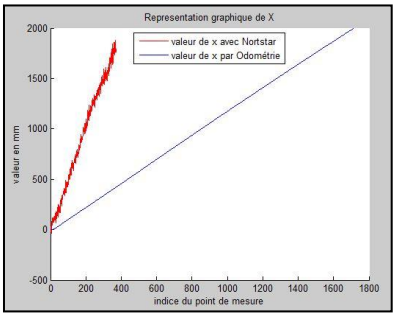
Il a fallu modifier le pas de calcul sur les valeurs de Northstar pour qu'une valeur corresponde à 3 valeurs sur le capteur d'Odométrie.

Exemple avec une consigne de 2000 mm selon x :

Avec modification du pas :

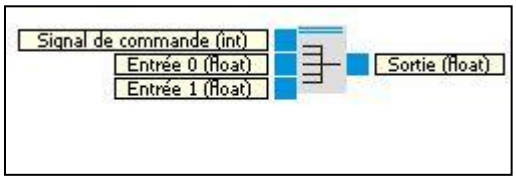


Sans modification du pas :



Il a fallu modifier le nombre de valeurs pour les calculs d'écart, car ces fonctions s'appliquent que sur des vecteurs de même taille. Donc il a fallu les retirées de façon logique ; ne pas garder que les première mais 1 sur 3, grâce à l'utilisation d'une boucle tant que.

3) Association des capteurs pour une meilleur précision du positionnement



Jusqu'à présent le capteur Northstar était passif, donc il y a eu modifications du programme sous RobotinoView2 pour que la position soit commandée initialement les valeurs d'Odométrie, puis si on constate un décalage de Delta, un bloc Multiplexeur effectue le changement pour que la commande soit faite à partir des valeurs de Northstar. Ce Delta pourra être choisi par l'utilisateur, il est initialement définit à 100 mm car la précision de Northstar est +/- 5cm, pour avoir une marge.

Le but est d'associer des capteurs pour une meilleure précision du positionnement.

Ce même programme codé sous RobotinoView2 est codé sous Matlab, pour pouvoir être plus facilement modifiable et surtout pour communiquer avec d'autres programmes Matlab.

III) Seconde partie : Phases du Projet sur Matlab

1) Transcription du programme sur RobotinoView sur Matlab

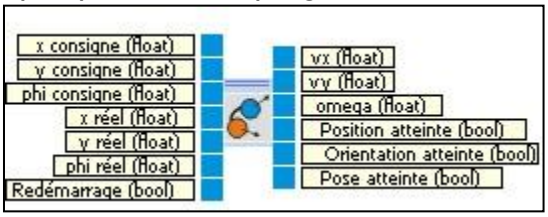
Objectifs :

Le but de ce programme est de transcrire le programme écrit sous RobotinoView en Matlab qui est moins intuitif au niveau de la programmation mais qui permet de modifier plus facilement un programme et d'interagir avec d'autres.

Il a fallu définir un Delta puis d'autres pour la précision comme pour les programmes sur RobotinoView2 et Matlab, cette valeur pourra être modifiée par la suite.

Le but est de ne plus avoir de défauts de transition entre les positions.

Pour faire le changement de consignes entre chaque position, le programme utilisait comme condition la sortie Position atteinte du bloc Parcoureur de position.



Ce bloc a 3 sorties de ce type :
Position atteinte : quand vx = 0 et vy = 0
Orientation atteinte : quand omega = 0
Pose atteinte : quand vx = 0, vy = 0 et omega = 0

Comme le programme utilisait la sortie Position atteinte quand on lui soumettait une consigne selon phi, il y avait une divergence, car cette sortie ne prend pas en compte omega, la vitesse angulaire. C'est pourquoi, le programme a été modifié pour prendre comme condition de transition la sortie Pose atteinte.

Les contraintes sont que la précision souhaitée en trop précise par rapport aux vitesses de déplacement. On peut définir la vitesse en fonction de la distance sur l'interface du bloc Parcoureur de position, c'est-à-dire plus on approche de la consigne plus on diminue la vitesse. Cependant avec une vitesse trop faible, le Robotino ne se déplace plus. Et si ce n'est pas le cas, le robot oscille autour de la position de consigne. Car cela exige une trop grande précision avec la précision des capteurs en particulier Northstar, pour pallier à cette contrainte que la définition d'un Delta serait utile sous Matlab.

Remarques : il y a eu 2 Delta, un pour x et y qui s'exprime en mm et un autre pour phi qui s'exprime en degrés décrit dans la suite du programme.

1-a) Correspondance entre les blocs sur RobotinoView et les fonctions sur Matlab

Le Robotino possède une Toolbox qui fournit un ensemble complet de fonctions qui permettent pour contrôler presque tous les aspects de Robotino dans Matlab. Chaque objet a un groupe de fonctions qui contient au moins la fonction *construct* (qui retourne un NomFonctionId qui est l'ID de l'objet qui vient d'être construit) et *destroy* (qui détruit l'objet NomFonction_destroy).

Voici les fonctions du Robotino sur Matlab contenu à l'accueil de l'aide de la Toolbox RobotinoMatlab :

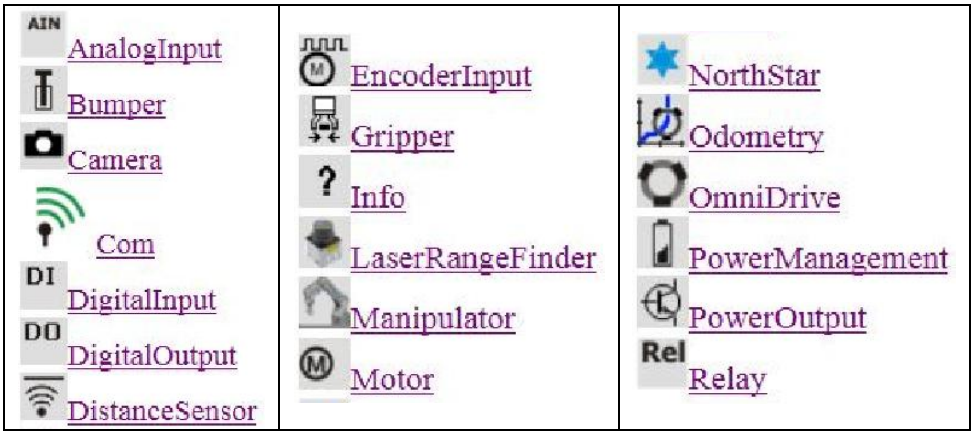


Tableau de correspondances entre les Entrées/Sorties des blocs sur RobotinoView et les fonctions Matlab pour le capteur Northstar :

Entrées	**Unité**	**Description**	**Fonction Matlab**
Numéro de local		Le numéro du local dans lequel Robotino se trouve	NorthStar_setCeilingCal
Calibrage du plafond	mm	Distance entre le détecteur et le plafond	NorthStar_setRoomId
X	mm	Position x de l'origine définie par l'entrée "Définir"	Ces entrées n'ont pas de fonction équivalente, comme la fonction Odometry_set pour l'Odométrie
Y	mm	Position y de l'origine définie par l'entrée "Définir"	
Phi	Degré	Orientation de l'origine définie par l'entrée "Définir"	
Définir		Si vrai (true) la pose (x,y,phi) est prise pour origine	
Sorties	**Unité**	**Description**	**Fonction Matlab**
X	mm	La position x courante	NorthStar_posX
Y	mm	La position y courante	NorthStar_posY
Phi	Degré	L'orientation courante	NorthStar_posTheta
Projecteurs		Nombre de projecteurs visibles	NorthStar_numSpotsVisible

Il y a aussi NorthStar_construct pour construire l'objet Northstar, NorthStar_destroy pour le détruire à la fin et NorthStar_setComId pour associer l'objet avec une interface de communication, c'est-à-dire lier l'objet Northstar à un Robotino spécifique par son adresse IP.

Les fonctions liées à l'interface du bloc Matlab sont NorthStar_magSpot0, NorthStar_magSpot1 pour le pourcentage de détection des 2 spots : Spot A et Spot B et NorthStar_sequenceNo pour le numéro de séquence qui correspond au nombre de fois où le capteur à tenter de capter les spots depuis le lancement du programme.

Tableau de correspondances entre les Entrées/Sorties des blocs sur RobotinoView et les fonctions Matlab pour le capteur Odométrie :

Entrées	Unité	Description	Fonction Matlab
X	mm	La nouvelle position x La valeur est reprise si l'entrée "Définir" est à l'état vrai (true).	Odometry_set (OdometryId, x, y, phi) = [success]
Y	mm	La nouvelle position y La valeur est reprise si l'entrée "Définir" est à l'état vrai (true).	
Phi	Degré	La nouvelle orientation. La valeur est reprise si l'entrée "Définir" est à l'état vrai (true).	
Définir		Si l'entrée est à l'état vrai (true), l'odométrie de Robotino recopie les valeurs des entrées x, y, phi. Pou mettre l'odométrie à (0,0,0), il suffit donc de mettre l'entrée à l'état vrai (true) pendant un cycle. Si l'entrée est à l'état faux (false), l'odométrie ne change que si les roues de Robotino bougent.	
Sorties	**Unité**	**Description**	**Fonction Matlab**
X	mm	La position x courante	Odometry_get (OdometryId) = [x, y, phi]
Y	mm	La position y courante	
Phi	Degré	L'orientation courante	

Comme pour le capteur Northstar les fonctions : Odometry_construct, Odometry_setComId et Odometry_destroy

1-b) Construction d'un programme pour commander un Robotino sur Matlab

Avant de lancer un programme qui utilise un Robotino sur Matlab, il faut lancer le programme nommé starup.m qui se trouve dans le dossier RobotinoMatlab et qui contient :

```
addpath( strcat( getenv('ROBOTINOMATLAB_DIR'), '/blockset' ) );
addpath( strcat( getenv('ROBOTINOMATLAB_DIR'), '/toolbox' ) );
```

Cela permet de charger les librairies associées.

Explication du code du programme :

```
close all
clear
clc
```

Close all : ferme toutes les figures et graphiques ouverts.
Clear : supprime toutes les variables à partir de l'espace de travail, les libérant de la mémoire système.
Clc : Clear Command Window : efface toutes les entrées et sorties de l'écran la fenêtre de commande.

```
ComId = Com_construct;
OmniDriveId = OmniDrive_construct;
OdometryId = Odometry_construct;
NorthStarId = Northstar_construct;
```

```
Motor0Id = Motor_construct(0);
Motor1Id = Motor_construct(1);
Motor2Id = Motor_construct(2);
BumperId = Bumper_construct;
```

Le programme Matlab nécessite d'abord de construire des objets utilisés dans le programme correspondant à peu près au bloc que l'on peut trouver sur RobotinoView. Le programme utilise comme objets de Com, OmniDrive, Bumper, Odometry et NorthStar. Dans le cas des Motor, il faut spécifier le numéro du moteur.

```
Com_setAddress(ComId,'172.26.94.17');
Com_connect(ComId);
```

Une fois les objets construits, un Id est retourné pour chaque objet. Cet ID est utilisé plus tard lors de la communication avec Robotino. Avant il faut définir l'adresse du Robotino utilisé puis s'y connecter.

Attention l'adresse IP et le numéro de port peuvent être différents

```
OmniDrive_setComId(OmniDriveId, ComId);
Odometry_setComId(OdometryId, ComId);
Northstar_setComId( NorthStarId, ComId );
Motor_setComId(Motor0Id, ComId);
Motor_setComId(Motor1Id, ComId);
Motor_setComId(Motor2Id, ComId);
Bumper_setComId(BumperId, ComId);
```

Une fois connecté au Robotino, chacun des objets créés auparavant sont liés à l'aide de la Robotino ComId.

```
{
        CORPS DU PROGRAMME
}
```

```
Com_disconnect(ComId);
```

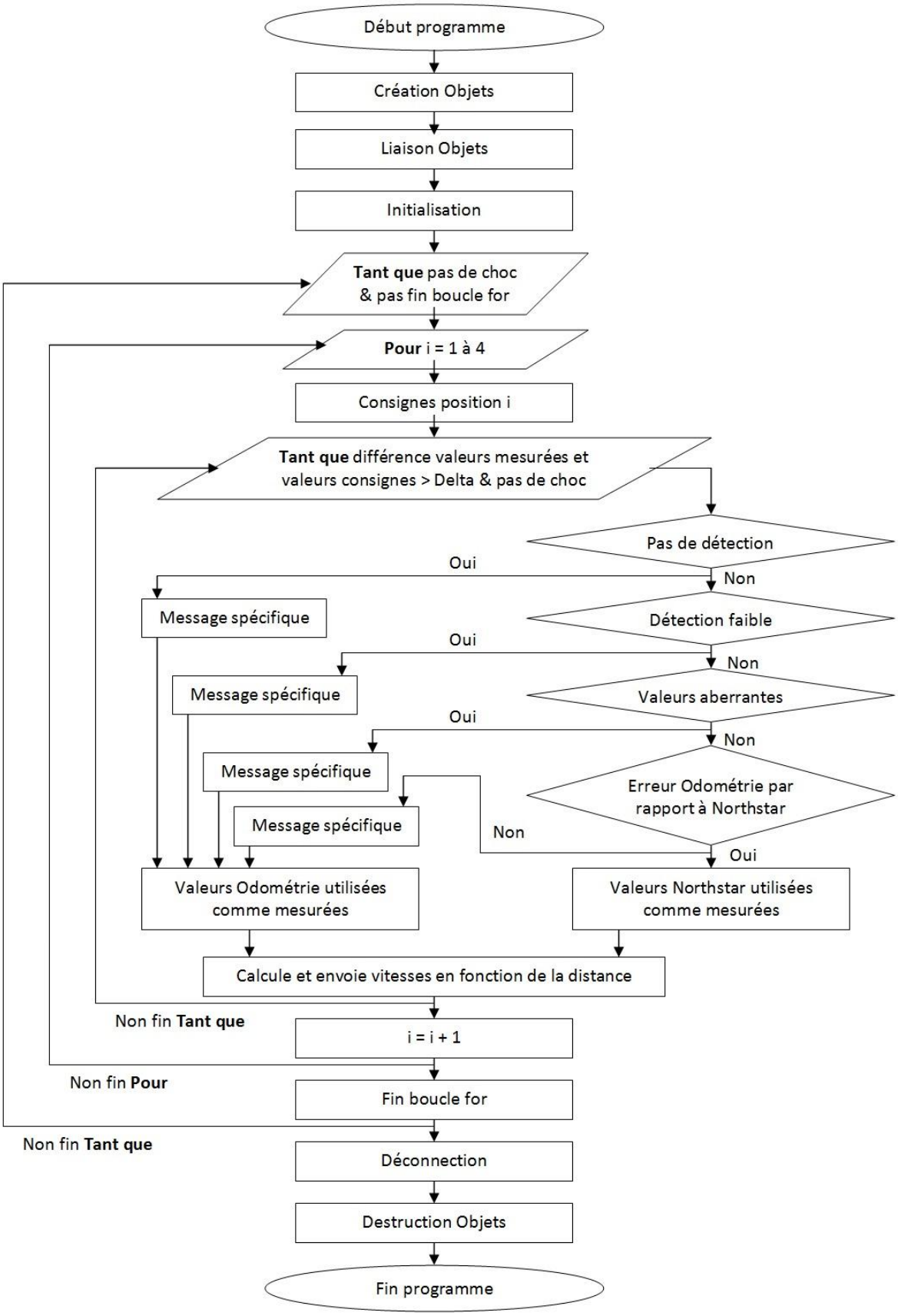
Une fois le programme terminé, il faut se déconnecter du Robotino.

```
Bumper_destroy(BumperId);
Motor_destroy( Motor0Id );
Motor_destroy( Motor1Id );
Motor_destroy( Motor2Id );
NorthStar_destroy(NorthStarId);
Odometry_destroy(OdometryId );
OmniDrive_destroy(OmniDriveId);
Com_destroy(ComId);
```

Il est recommandé de détruire tous les objets qui ont été créés pour le programme comme on a pu voir en cours d'informatique et de temps réel.

1-c) Programme principal

Ordinogramme du programme principal :


Début programme
Création Objets
Liaison Objets
Initialisation
Tant que pas de choc & pas fin boucle for
Pour i = 1 à 4
Consignes position i
Tant que différence valeurs mesurées et valeurs consignes > Delta & pas de choc
Pas de détection
Oui
Non
Message spécifique
Détection faible
Oui
Non
Message spécifique
Valeurs aberrantes
Oui
Non
Message spécifique
Erreur Odométrie par rapport à Northstar
Message spécifique
Non
Oui
Valeurs Odométrie utilisées comme mesurées
Valeurs Northstar utilisées comme mesurées
Calcule et envoie vitesses en fonction de la distance
Non fin Tant que
i = i + 1
Non fin Pour
Fin boucle for
Non fin Tant que
Déconnection
Destruction Objets
Fin programme

Explication du code du programme principal :

Une boucle *while* est lancée après l'initialisation des variables, elle est basée sur la condition que le pare-chocs (Bumper) ne soit pas en contact et qu'une variable qui passe à 1 quand la boucle fort qui suit se termine.

```
while Bumper_value(BumperId) ~= 1 && fin_position_for ~= 0
```

Une boucle *for* permet de l'exécution des différentes positions de consigne

```
for i=1:4

Cx = posC(i,1);
Cy = posC(i,2);
Cphi = posC(i,3);
```

Cx, Cy et Cphi sont les valeurs de consignes selon ces 3 variables et x, y et phi correspond aux valeurs actuelles mesurés soit Ox, Oy et Ophi si les valeurs de l'odométrie sont plus cohérentes ou Nx, Ny et Nphi si ceux sont celles de Northstar.

Une boucle *while* qui a pour condition que tant que la valeur absolue de l'écart entre la valeur de consigne et celle mesurée selon x, y et phi soit supérieur à une valeur Delta. Qui correspond à la précision exigée autour d'une position (x, y, phi). Ce delta varie est soit égale à DeltaN ou DeltaO car le capteur d'Odométrie est plus précis que Northstar.

```
while abs(Cx - x) > Delta && Bumper_value(BumperId) ~= 1 ...
      || abs(Cy - y) > Delta && Bumper_value(BumperId) ~= 1 ...
      || abs(Cphi - phi) > Delta && Bumper_value(BumperId) ~= 1
```

Ensuite, avec un *if* et plusieurs *elseif*, plusieurs conditions sont vérifiées :

(1) S'il n'y a aucune détection du capteur
C'est-à-dire s'il ne capte aucun spot lumineux et que leurs intensités sont nuls

```
if ( NorthStar_magSpot0(NorthStarId) ~= 0 && NorthStar_magSpot1(NorthStarId) ~= 0 ...
     && NorthStar_numSpotsVisible(NorthStarId) ~= 0)
```

(2) Si la détection est trop faible
C'est-à-dire si le capteur capte les 2 spots lumineux et que chacun d'eux une intensité supérieur à un pourcentage, ici 3 %

```
if (NorthStar_magSpot0(NorthStarId)/1000 > 3 && NorthStar_magSpot1(NorthStarId)/1000
> 3 ...
    && NorthStar_numSpotsVisible(NorthStarId) == 2)
```

(3) Si les valeurs du capteur Northstar semblent incohérentes
C'est-à-dire si pour une vitesse non nulle selon x, y ou phi, la valeur de la position ne doit pas être égale à 0

Cette condition n'est pas la plus efficace, mais à part en ayant un troisième capteur référent, elle parait possible car avec la précision à 4 chiffres après la virgule, cela semble cohérent.

```
if (Vx~=0 || Vy~=0 || Vphi~=0) && ...
        (Nx~=0 && Ny~=0 && Nphi~=0)
```

(4) Si l'écart entre les valeurs de l'Odométrie et celles de Northstar sont assez importantes

C'est-à-dire si l'erreur entre la valeur du capteur Northstar et celle par Odométrie pour chaque variable x, y et phi est supérieur à la variable erreur_O_N qui est la valeur d'erreur acceptable vis-à-vis de la précision du capteur Northstar

```
if (abs(Ox - Nx) > erreur_O_N && abs(Oy - Ny) > erreur_O_N ...
    || abs(Ox - Nx) > erreur_O_N && abs(Ophi - Nphi) > erreur_O_N ...
    || abs(Oy - Ny) > erreur_O_N && abs(Ophi - Nphi) > erreur_O_N)
```

Si ces 4 conditions sont remplies les valeurs utilisées sont celles du capteur Northstar

```
x = Nx;
y = Ny;
phi = Nphi;
Delta = DeltaN;
```

Sinon ceux sont celles obtenues par le capteur par Odométrie

```
x = Ox;
y = Oy;
phi = Ophi;
Delta = DeltaO; % en mm % precision capteur Odometrie
```

Dans ce cas s'affiche dans l'écran de commande Window pour quelle raison n'est pas retenu le capteur Northstar selon les 4 critères précédents :

```
Pas_erreur_Odometrie = 1;
Valeurs_NorthStar_fausses = 1;
Captation_NorthStar_trop_faible = 1;
Pas_Captation_NorthStar = 1;
```

Après selon l'écart entre la consigne et la valeur réelle mesurée, le programme détermine la vitesse, plus il est loin, plus il va vite. Exemple avec la vitesse selon x :

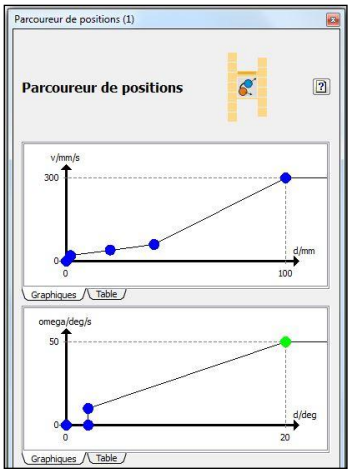
```
% vitesse selon x

distance_x = Cx - x;
if (0 <= abs(distance_x) && abs(distance_x) < 2)
        Vx = 0 * sign(distance_x);
elseif (2 <= abs(distance_x) && abs(distance_x) < 20)
        Vx = 20 * sign(distance_x);
elseif (20 <= abs(distance_x) && abs(distance_x) < 40)
        Vx = 40 * sign(distance_x);
elseif (40 <= abs(distance_x) && abs(distance_x) < 100)
```

```
        Vx = 60 * sign(distance_x);
else
        Vx = 300 * sign(distance_x);
end
```

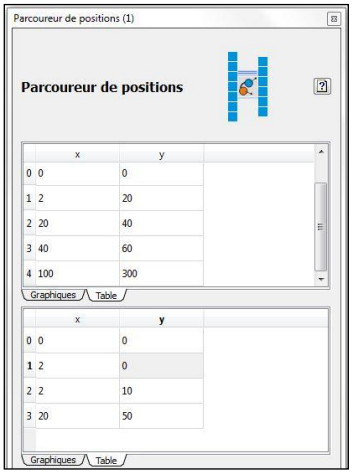
La vitesse Vy selon y est exactement la même. Pour celle Vphi selon phi, seul un nombre de pallier plus petit et la valeur maximum diffère.

Ce code correspond à la transcription en version palier des courbes du bloc *Parcoureur de positions* sur RobotinoView :

Affichage graphiques :



Affichage tableaux :



Une fois, les valeurs des vitesses déterminées, elles sont envoyées au moteur par le biais de la fonction *OmniDrive_setVelocity* qui commande les moteurs :

```
OmniDrive_setVelocity(OmniDriveId, Vx, Vy, Vphi);
```

A ne pas confondre avec la fonction *OmniDrive_getVelocities* qui elle ne renvoie que les valeurs de vitesse converties pour chacun des 3 moteurs en fonction des valeurs de vitesse selon la direction x, y et phi. Et qui n'a aucune action réelle sur les moteurs.

```
[ Vm1, Vm2, Vm3 ] = OmniDrive_getVelocities(OmniDriveId, Vx, Vy, Vphi);
```

À partir de là, la deuxième boucle *while* continue tant que le Robotino n'est pas proche à la précision voulue (Delta) de la position de consigne. Quand cette condition est atteinte la boucle *for* est incrémentée pour passer à la position de consigne suivante. Enfin la première boucle *while* se termine quand toutes les positions de consigne ont été parcourues. Ensuite une commande de vitesse nulle pour Vx, Vy et Vphi est envoyée, la communication est déconnectée et les objets détruits pour libérer la mémoire.

1-d) Programme d'Interface Homme-Machine (IHM)

Pour pouvoir modifier les valeurs de consignes RobotinoView, il y a un tableau à droite dans le programme avec le nom des variables du programme, leur type et leur valeur. C'est à cet endroit que l'on modifie les valeurs des consignes qui sont les vecteurs nommés *x*, *y* et *phi* (surligné en bleu sur l'image à droite).

Dans les programmes Matlab, les consignes sont en « dur » mais en retirant le pourcentage de commentaire devant et en utilisant le programme d'interface associée :

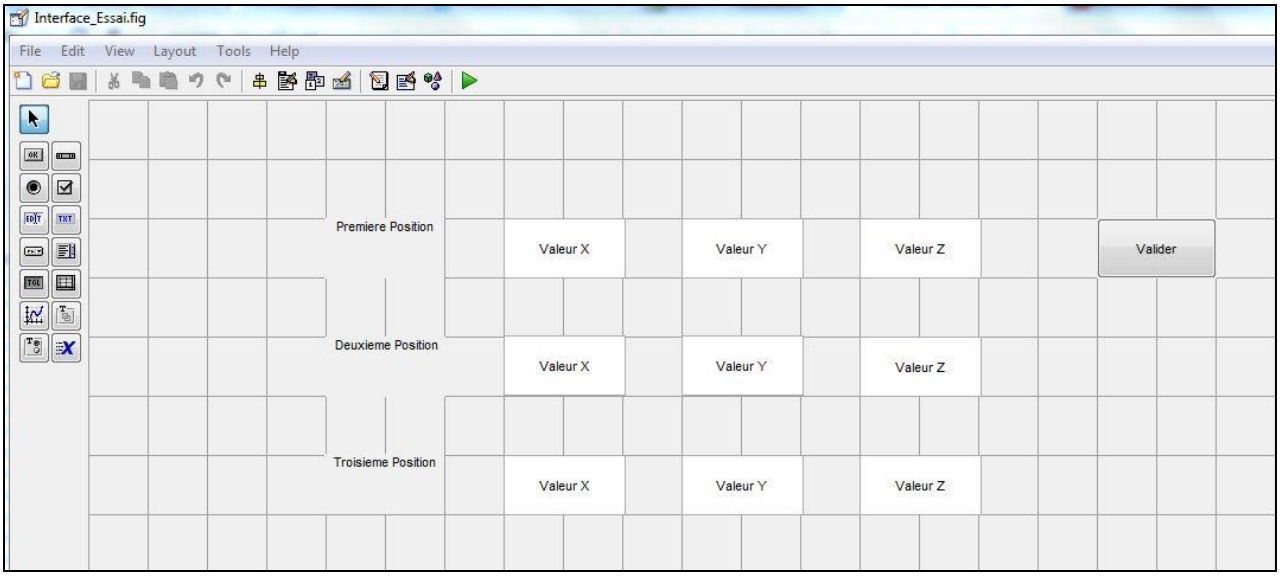
Nom	Type	Valeur
Boolean	float	0.000000
Calibrage_plafon	float	0.000000
Nbr_projecteurs	float	0.000000
Numero_local	float	3.000000
Odo_ou_North	float	0.000000
arret	float	0.000000
incrementation	float	0.000000
incrementation_n1	float	0.000000
incrementation_ok	float	0.000000
nbr_positions	float	4.000000
phi	floatvector	(0 0 0 0)
phi_error	float	0.000000
phi_origine	float	0.000000
position_atteinte	float	0.000000
position_consig_x_y_phi	float	0.000000
x	floatvector	(0 1000 0 -1000 0)
x_error	float	0.000000
x_origine	float	0.000000
y	floatvector	(0 0 500 0 -500)
y_error	float	0.000000
y_origine	float	0.000000

```
% pos1 = [x1 y1 phi1];
% pos2 = [x2 y2 phi2];
% pos3 = [x3 y3 phi3];
% posC = [pos1; pos2; pos3];
```

On n'a plus besoin de rentrer dans le programme pour modifier ces valeurs de consignes

On peut programmer une interface graphique sur Matlab appelé GUI (Graphical User Interfaces) avec un fichier de type .fig sur lequel on dispose les boutons, interfaces, éditeurs, listes ou autres. Quand on exécute le fichier .fig, on obtient un fichier Matlab .m sur lequel, on peut rajouter du code. On peut aussi programmer directement l'interface en avec le fichier .m sans passer par le fichier .fig.

Pour la programmation, je suis passé par un fichier .fig, de cette forme



Pour pouvoir les distinguer les différents éditeurs de texte (en blanc), on utilise le *tag* qui est l'identifiant de l'objet. Ensuite en exportant, en fichier .m, j'ai ajouté les lignes de code, exemple sur l'éditeur de texte avec le tag *x1* :

```
g_x1 = str2num(get(handles.x1,'string'));
```

Cette instruction récupère le contenu du tag *x1* qui est un string (chaînes de caractères) en valeur numérique dans la variable *g_x1*.

```
assignin('base','x1',g_x1);
```

Cette instruction crée dans le Workspace une valeur *x1* à laquelle est attribuée la valeur de *g_x1*.
Remarque la valeur crée dans le Workspace est nommé *x1* et le tag de l'éditeur de texte est *x1*, cependant ils n'ont pas de lien direct.

2) Amélioration de la recherche du paramétrage d'initialisation

Lors de l'initialisation, le défaut du paramétrage du plafond en particulier, c'est qu'il était fait part essais successifs par l'utilisateur.

Objectifs :

Le but des programmes écrits dans cette partie est de définir la valeur du calibrage du plafond de façon automatique. Le programme est inspiré du programme de recherches de paramètres écrits sous RobotinoView, c'est-à-dire de déplacer ou non le robot mobile vers une position de consigne en prenant comme valeurs réelles pour la commande celles obtenues par Odométrie et au lieu de tester différentes valeurs de calibration à la main, il faudrait le tester de façon rapide et simple.

Ce qui permet de ne plus faire cette recherche de paramétrage de façon empirique mais de façon automatique.

Les contraintes sont de choisir une méthode qui pourrait répondre à cette problématique. La méthode qui parait la plus appropriée à première vue est la méthode par dichotomie étudiée en cours d'analyse numérique.

2-a) Programme de paramétrage par dichotomie

La construction du programme est la même que pour le programme en ce qui concerne la création, liaison et destruction des objets, c'est le corps du programme qui change.

La dichotomie est un algorithmique itératif ou récursif de recherche où, à chaque étape, on coupe en deux parties un espace de recherche qui devient restreint à l'une de ces deux parties à l'itération suivante.

Le but de cet algorithme était donc de prendre une plage de valeur de calibration du plafond assez grande et de se rapprocher peut d'une valeur pertinente pour le programme principale.

A l'initialisation, on définit la plage de valeur en définissant le calibrage minimum et maximum, en suite appelé *a* et *b* avec *c* le milieu de l'intervalle. On définit aussi un nombre d'itération maximum au cas où soit l'algorithme ne trouve pas de valeur ou que la précision choisie est trop élevée. On définit la valeur de d'un epsilon qui correspond à la précision recherchées dans un premier temps, le programme utilisait un seul epsilon *epsi* mais dans un second temps, il en utilisait deux, un pour les valeurs en millimètre (x et y) : *epsi_mm* et un pour les valeurs en degrés (phi) : *epsi_deg*.

```
a = calib_min;
b = calib_max;
c = (a+b)/2;

nb_iter_max = 300;

epsi = 100;
epsi_mm = 200;
epsi_deg = 50;
```

Une boucle *while* est lancée après l'initialisation des variables, elle est basée sur la condition que le pare-chocs (Bumper) ne soit pas en contact, que le nombre d'itération est inférieur au nombre maximum et l'écart entre la valeur du capteur Nothstar avec le calibrage *c* (centre de la plage de calibration) et la valeur de l'Odométrie pour l'une des trois coordonnées (x, y et phi) soit supérieur ou égale à la précision recherchée.

```
while Bumper_value(BumperId) ~= 1 && iter<nb_iter_max && ...
        (abs(xNc-xO)>=epsi_mm || abs(yNc-yO)>=epsi_mm || abs(phiNc-phiO)>=epsi_deg)
```

Ensuite on peut choisir de mettre une de vitesse pour faire le paramétrage avec le Robotino en mouvement ou immobile en commentant cette ligne.

```
OmniDrive_setVelocity(OmniDriveId, 50, -50, 0);
```

Après on définit pour le calibrage minimum *a* la valeur du capteur Northstar selon x, y et phi.

```
NorthStar_setCeilingCal( NorthStarId, a );
xNa = NorthStar_posX( NorthStarId );
yNa = NorthStar_posY( NorthStarId );
phiNa = NorthStar_posTheta( NorthStarId );
```

De même pour la valeur de calibrage maximum *b* et du centre de l'intervalle *c*.

Ayant un doute sur la vitesse de changements des valeurs en fonction du changement de calibrage, une temporisation a été rajouté par le biais d'une boucle *for*.

```
for i=1:10

      NorthStar_setCeilingCal( NorthStarId, a );
      xNa = NorthStar_posX( NorthStarId );
      yNa = NorthStar_posY( NorthStarId );
      phiNa = NorthStar_posTheta( NorthStarId );

      i = i + 1;
end
```

Ensuite, avec un *if* et plusieurs *elseif*, plusieurs conditions sont vérifiées :

(1) Si la valeur par Odométrie de x, y et phi est dans l'intervalle [a, c] des valeurs de Northstar

```
if (xNa <= xO && xO <= xNc) && (yNa <= yO && yO <= yNc) ...
                && (phiNa <= phiO && phiO <= phiNc)
```

Alors cette intervalle sera choisit pour la prochaine itération :

```
b=c;
c=(a+c)/2;
```

(2) Si la valeur par Odométrie de x, y et phi est dans l'intervalle [c, b] des valeurs de Northstar

```
elseif (xNc <= xO && xO <= xNb) && (yNc <= yO && yO <= yNb) ...
                && (phiNc <= phiO && phiO <= phiNb)
```

Alors cette intervalle sera choisit pour la prochaine itération :

```
a=c;
c=(b+c)/2;
```

(3) Si la valeur par Odométrie de x et y (en négligeant une erreur sur phi) est dans l'intervalle [a, c] des valeurs de Northstar

```
elseif (xNa <= xO && xO <= xNc) && (yNa <= yO && yO <= yNc)
```

Alors cette intervalle sera choisit pour la prochaine itération :

```
b=c;
c=(a+c)/2;
```

(4) Si la valeur par Odométrie de x et y (en négligeant une erreur sur phi) est dans l'intervalle [c, b] des valeurs de Northstar

```
elseif (xNc <= xO && xO <= xNb) && (yNc <= yO && yO <= yNb)
```

Alors cette intervalle sera choisit pour la prochaine itération :

```
a=c;
c=(b+c)/2;
```

Ainsi de suite jusqu'à ce que la valeur de calibrage de plafond réponde au attente de précision.

Cependant comme l'algorithme ne fonctionnait pas correctement, il y a eu deux solutions envisagées et mises en place. La première est de juste de regardé seulement si les valeurs de l'Odométrie sont soit inférieur ou supérieur aux valeurs de Northstar pour le calibrage *c* (le milieu de l'intervalle) sans prendre en compte les extrémités de l'intervalle. Cette solution a été envisagée car les valeurs maximum et minimum de calibrage fixé dans le programme de dichotomie pouvaient induire en erreur l'algorithme en donnant des valeurs aberrantes aux extrêmes, ce qui empêche de considéré les valeurs dans l'un des deux intervalles [a, c] ou [c, b]. Donc la deuxième solution était de mieux définir *a* et *b* pour l'intervalle, c'est le but de ce nouveau programme.

2-b) Programme de paramétrage pas à pas

Pour mieux définir l'intervalle de recherches de l'algorithme de dichotomie, écriture d'un autre programme dont la construction du programme est la même que pour le programme principale et par dichotomie en ce qui concerne la création, liaison et destruction des objets, c'est le corps du programme qui change.

Le corps du programme est à un but assez proche que la dichotomie, mais là quand il balaie pas à pas (pas = 1) un intervalle dans le sens croissant puis dans le sens décroissant, la valeur du calibrage à partir duquel la valeurs de x, y et phi sont différentes de zéro sont enregistrés pour servir de minimum et de maximum pour l'intervalle de recherches de l'algorithme de dichotomie.

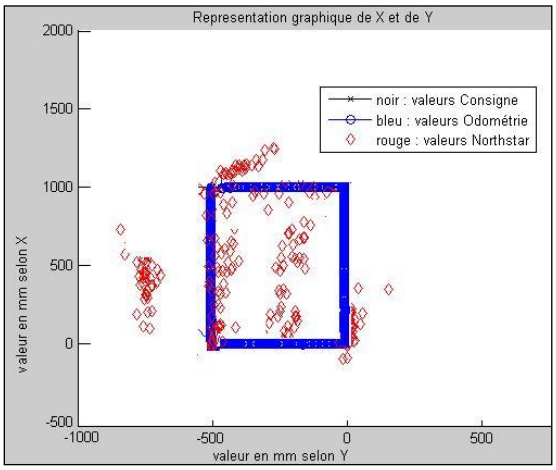
De plus, pour que ce programme ai une utilité plus large, une condition est rajouté : c'est d'enregistrer quand il parcourt pas à pas l'intervalle complet de calibration de trouver l'intervalle dans lequel, la différence entre les valeurs (x, y et phi) du capteur Northstar et celles de l'Odométrie sont inférieurs aux valeurs *epsi_mm* et *epsi_deg*, utilisées dans l'algorithme et décrites précédemment.

3) Résultats, robustesse et améliorations

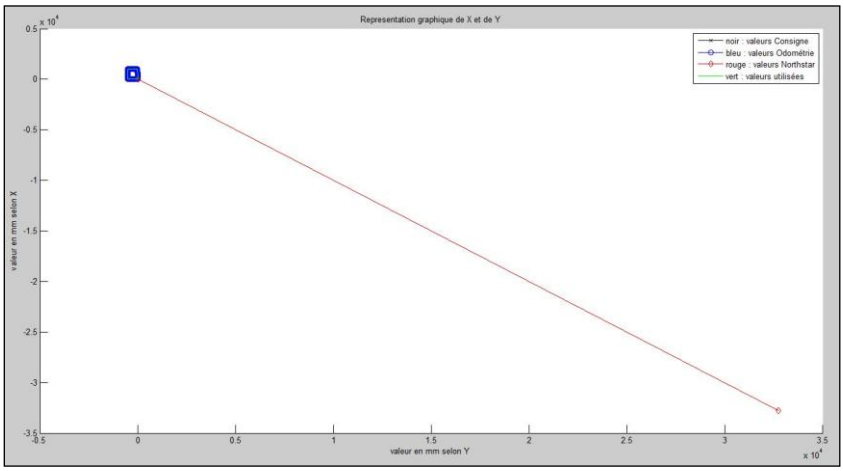
3-a) Résultats

Voici le résultat du parcours de 4 positions formant un rectangle de 1000 mm de longueur et 500 mm de largeur sur RobotinoView et sur Matlab

Sur RobotinoView :

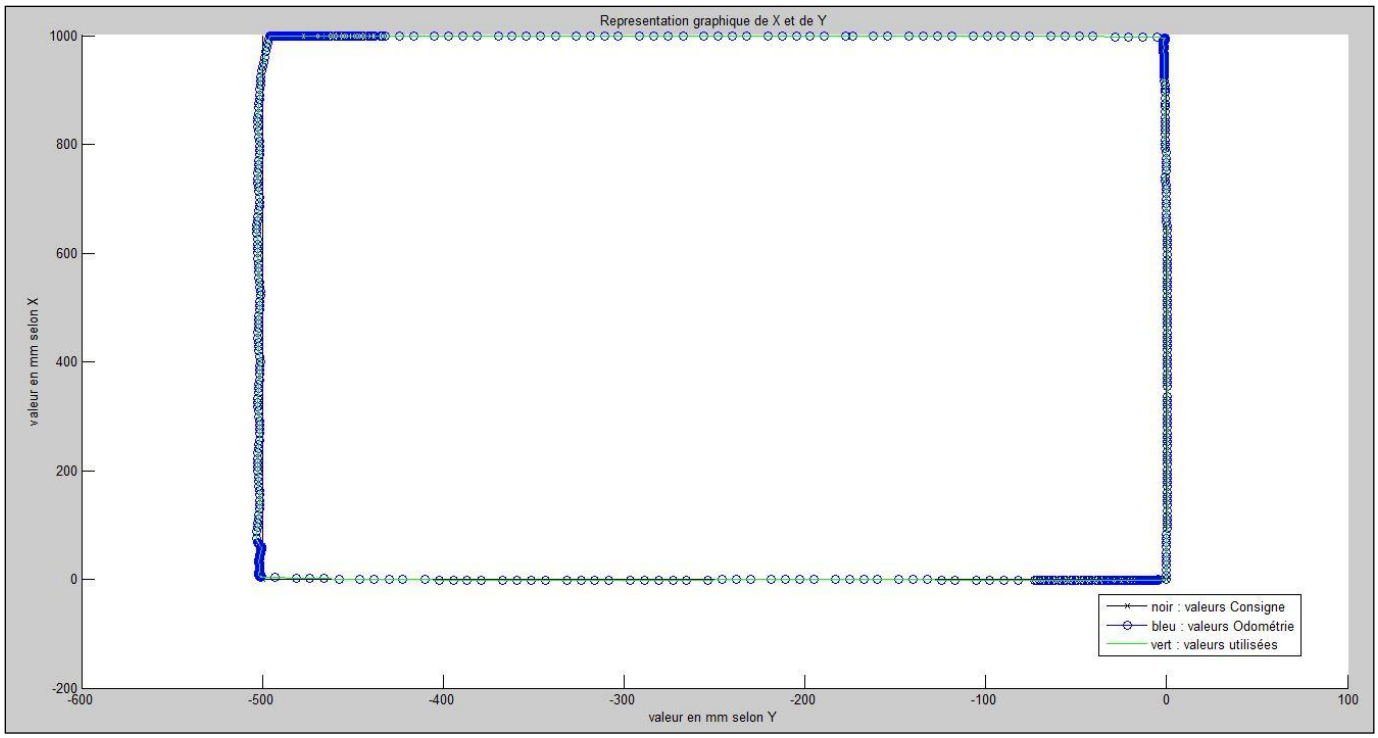


Sur Matlab :



Le trait en diagonale représente les valeurs de Northstar trop grande, alors que le même calibrage est utilisé, on fait un zoom sur la consigne

Zoom sur la consigne sur Matlab :



3-b) Robustesse : prise en compte globale par un coefficient ou un correcteur

Les programmes Matlab étant plus facilement modifiables, on pourrait calculer et mettre en place un coefficient qui serait comme un gain qui dépendrai de :

- La trajectoire du Robotino (car l'étude des trajectoires à montrer que Northstar donné de des valeurs plus précise selon la trajectoire, par exemple si la trajectoire est uniquement selon la direction x. Les valeurs de x délivrées par Northstar seront plus fiable que selon y ou phi.)
- La vitesse du Robotino (l'étude à montrer que plus, le robot va vite, moins les valeurs sont précises dû au temps de traitement du capteur)
- Le pourcentage de détection des spots par le capteur Northstar (l'étude à montrer que lorsque que le pourcentage devient trop faible, inférieur à environ 3 %, les valeurs de Northstar ne sont plus cohérentes.

Le but serait de ne pas avoir seulement un dispositif binaire, car la solution de correction est à lors actuel fait avec un multiplexeur sur RobotinoView et par une condition *if* sur Matlab c'est-à-dire soit on se réfère aux valeurs d'un capteur ou d'un autre.

Les contraintes sont que ce coefficient pondère au mieux l'influence des conditions précédentes. On aurait pu penser à un moyenneur moins cette méthode ne serait pas assez adaptative en fonction des conditions.

Cependant cela n'a pas été mis en place car le problème est de ne pas avoir des résultats aussi cohérents sur Matlab que sur RobotinoView, car si c'était le cas,

3-c) Amélioration par l'utilisation d'autres capteurs

L'utilisation des capteurs de distance (DistanceSensor) peut être une autre sécurité.

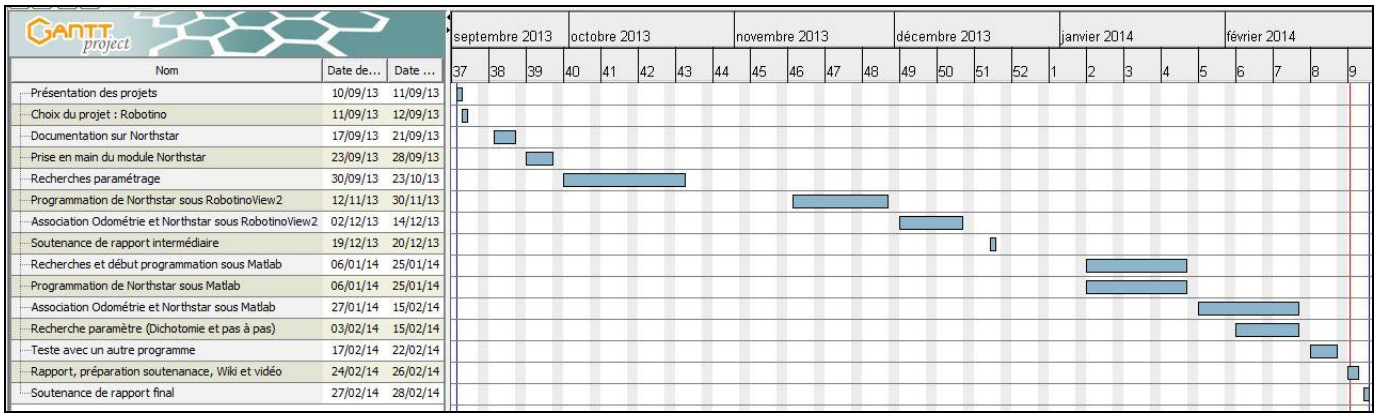
Si par exemple le capteur est très loin de la consigne dans une direction, sa vitesse sera importante, car c'est ce qui est écrit dans le programme principal. Cependant en utilisant les capteurs de contact, on pourrait modérer cette vitesse.

Cette sécurité serait utile pour protéger le Robotino des chocs, car actuellement la vitesse est coupée quand le pare-chocs (Bumper) est à « 1 », c'est-à-dire en contact. Mais cela permettrait aussi de pouvoir mettre le projecteur au centre de la piste pour avoir un domaine d'actions plus important.

IV) Gestion de projet

1) Planning prévisionnel

Voici le Gantt qui représente le planning prévisionnel fixé au début du projet et refixé à partir de la soutenance intermédiaire :

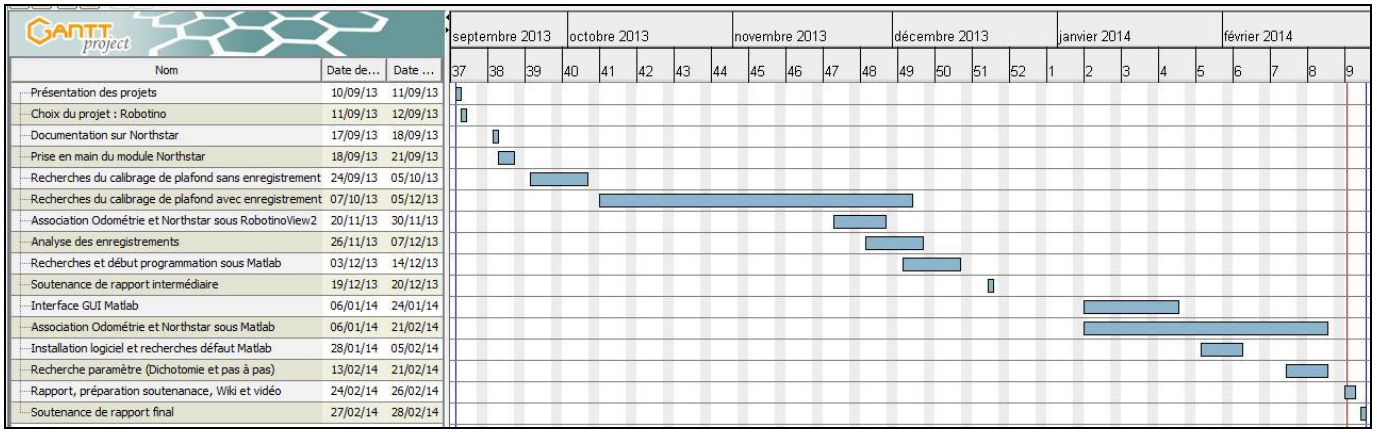


Nom	Date de...	Date ...
Présentation des projets	10/09/13	11/09/13
Choix du projet : Robotino	11/09/13	12/09/13
Documentation sur Northstar	17/09/13	21/09/13
Prise en main du module Northstar	23/09/13	28/09/13
Recherches paramétrage	30/09/13	23/10/13
Programmation de Northstar sous RobotinoView2	12/11/13	30/11/13
Association Odométrie et Northstar sous RobotinoView2	02/12/13	14/12/13
Soutenance de rapport intermédiaire	19/12/13	20/12/13
Recherches et début programmation sous Matlab	06/01/14	25/01/14
Programmation de Northstar sous Matlab	06/01/14	25/01/14
Association Odométrie et Northstar sous Matlab	27/01/14	15/02/14
Recherche paramètre (Dichotomie et pas à pas)	03/02/14	15/02/14
Teste avec un autre programme	17/02/14	22/02/14
Rapport, préparation soutenanace, Wiki et vidéo	24/02/14	26/02/14
Soutenance de rapport final	27/02/14	28/02/14

Ce Gantt fait au début des deux parties du projet devait me servir de fil conducteur pour estimer ma progression et mon retard si cela était le cas.

2) Planning réel

Voici le Gantt qui représente le planning réel suivi pendant le projet :



Nom	Date de...	Date ...
Présentation des projets	10/09/13	11/09/13
Choix du projet : Robotino	11/09/13	12/09/13
Documentation sur Northstar	17/09/13	18/09/13
Prise en main du module Northstar	18/09/13	21/09/13
Recherches du calibrage de plafond sans enregistrement	24/09/13	05/10/13
Recherches du calibrage de plafond avec enregistrement	07/10/13	05/12/13
Association Odométrie et Northstar sous RobotinoView2	20/11/13	30/11/13
Analyse des enregistrements	26/11/13	07/12/13
Recherches et début programmation sous Matlab	03/12/13	14/12/13
Soutenance de rapport intermédiaire	19/12/13	20/12/13
Interface GUI Matlab	06/01/14	24/01/14
Association Odométrie et Northstar sous Matlab	06/01/14	21/02/14
Installation logiciel et recherches défaut Matlab	28/01/14	05/02/14
Recherche paramètre (Dichotomie et pas à pas)	13/02/14	21/02/14
Rapport, préparation soutenanace, Wiki et vidéo	24/02/14	26/02/14
Soutenance de rapport final	27/02/14	28/02/14

L'écart entre le planning prévisionnel et réel est dû :

- Pour la première partie, j'avais déjà utilisé le logiciel RobotinoView, c'est pourquoi j'ai peut être eu une vision un trop optimiste. D'une part, je n'avais jamais utilisé ce capteur et d'autre n'avais jamais programmé avec certain

- Pour la deuxième partie, j'ai eu du retard dans mon planning pour deux raison principale. La première raison était dû au fait que je n'ai pas pensé que le

problème de fonctionnement des programmes écrits en Matlab n'était pas dû à une version de logiciel mais plutôt au type de PC (64 bits et non 32 bits). La seconde raison était surement de penser que comme les blocs spécifiques au Robotino existants sur RobotinoView ont une ou plusieurs fonctions équivalentes sur Matlab. Que j'obtiendrai directement le même résultat que j'avais obtenu sur l'un comme sur l'autre.

Conclusion

➔ Etat d'avancement du projet :

Les objectifs liés à la connaissance et à la prise en main du module Northstar ont été traités sur RobotinoView et Matlab.

Cependant il reste des améliorations à apporter pour les cas cités précédemment pour augmenter la robustesse des programmes, qui n'ont pas pu être fait à cause de valeurs plus imprécises sur Matlab que sur RobotinoView.

Ce défaut empêche le teste sur une trajectoire quelconque ou en l'associant avec un autre programme écrit sous Matlab.

➔ Bilan des compétences techniques acquises :

Ce projet m'a permis d'utiliser une nouvelle fois, un Robotino et le logiciel de programmation avec RobotinoView2, dont le développement se fait avec des blocs qui représente des fonctions plus ou moins évoluées. Et dont le programme principal s'écrit comme un Grafcet.

Au niveau du logiciel Matlab, j'ai pu réutiliser des connaissances acquises lors du cours d'analyse de données pour l'étude des enregistrements. Mais aussi à apprendre à commander un Robotino avec Matlab, une partie que nous avions à peine abordée l'année dernière. Et cela m'a permis d'utiliser l'interface graphique GUI sur Matlab.

➔ Bilan des compétences acquises en gestion de projet :

Ce projet m'a permis d'acquérir une meilleure gestion de projet en utilisant les Gantt, en particulier, à mieux gérer les imprévus, comme avec les compatibilités des versions des logiciel, les disponibilités de matériels et de salles … Mais aussi à voir l'importance de respecter son planning comme je l'ai expliqué dans la partie gestion de projet.

Ce sujet, m'a permis aussi d'avoir un recul sur mes connaissances sur les capteurs et leur précision. Car au début du projet, comme j'avais déjà fait des Travaux Pratiques sur Robotino, l'Odométrie me paraissait sans faille. Cependant arrivé au stade du projet, j'ai pu constater les limites de ce capteur dans des cas particuliers. Ce qui me fait voir l'intérêt de tester la faisabilité d'un projet, liée aux technologies et techniques associées. Dans le cas de problèmes comme j'ai rencontrés, dans mon avenir professionnel, je pense qu'il faut prévoir une ou plusieurs solutions de changement ou avoir de l'expérience dans le domaine vis d'autres projets. Cela correspond à la résolution de défauts ou d'erreur sur un capteur en utilisant la redondance abordée en sureté de fonctionnement et en supervision.

Ce problème a encore montré l'intérêt de ne pas partir tête baissée dans un sujet et de bien définir les objectifs.

Annexe

Document décrivant :

Programme de commande de Northstar codé sur Matlab : Northstar.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     Commande de Nothstar                              %
%                          Avec Matlab                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% fonctions NorthStar

% NorthStar_construct
% NorthStar_destroy
% NorthStar_magSpot0
% NorthStar_magSpot1
% NorthStar_numSpotsVisible
% NorthStar_posTheta
% NorthStar_posX
% NorthStar_posY
% NorthStar_roomId
% NorthStar_sequenceNo
% NorthStar_setCeilingCal
% NorthStar_setComId
% NorthStar_setRoomId

close all
clear
clc

%  objects construction

ComId = Com_construct;
OmniDriveId = OmniDrive_construct;
OdometryId = Odometry_construct;

NorthStarId = Northstar_construct;

% motors

Motor0Id = Motor_construct( 0 );
Motor1Id = Motor_construct( 1 );
Motor2Id = Motor_construct( 2 );
BumperId = Bumper_construct;

% the IP address and port number might be different

Com_setAddress(ComId,'172.26.94.17');
Com_connect(ComId);
```

```
% bind each of the object we created to Robotino using the ComId

OmniDrive_setComId(OmniDriveId, ComId);

Odometry_setComId(OdometryId, ComId);

Motor_setComId(Motor0Id, ComId);
Motor_setComId(Motor1Id, ComId);
Motor_setComId(Motor2Id, ComId);
Bumper_setComId(BumperId, ComId);

Northstar_setComId( NorthStarId, ComId );


NorthStar_setRoomId( NorthStarId, 3 );

% NorthStar_setCeilingCal( NorthStarId, 750 );

%NorthStar_setCeilingCal( NorthStarId, 0.16 );

%NorthStar_setCeilingCal( NorthStarId, 3200 );

NorthStar_setCeilingCal( NorthStarId, 3200 ); % valeur dichotomie
%NorthStar_setCeilingCal( NorthStarId, 4.3594e+003 ); % valeur dichotomie



% start our "stop watch"
tStart = tic;

% programme

% Consignes

%pos1 = [1000 0 0];
%pos2 = [0 -500 0];
%pos3 = [-1000 0 0];
%pos4 = [0 500 0];

pos1 = [1000 0 0];
pos2 = [1000 -500 0];
pos3 = [0 -500 0];
pos4 = [0 0 0];

posC = [pos1; pos2; pos3; pos4];

% pos1 = [x1 y1 phi1];
% pos2 = [x2 y2 phi2];
% pos3 = [x3 y3 phi3];

% posC = [pos1; pos2; pos3];

% Cx = 1000;
```

```
% Cy = 0;
% Cphi = 0;

% initialisations

DeltaN = 100; % en mm % precision capteur Northstar
DeltaO = 5; % en mm % precision capteur Odometrie
Delta = 5; % en mm % precision globale soit Delta_mm_N soit Delta_mm_O

%Delta_mm_N = 100; % en mm % precision capteur Northstar
%Delta_mm_O = 5; % en mm % precision capteur Odometrie
%Delta_mm = 5; % en mm % precision globale soit Delta_mm_N soit Delta_mm_O

%Delta_deg_N = 20; % en degres % precision capteur Northstar
%Delta_deg_O = 10; % en degres % precision capteur Odometrie
%Delta_deg = 10; % en degres % precision globale soit Delta_deg_N soit Delta_deg_O

erreur_O_N = 100; % en mm % valeur switch Odometrie vers Northstar

Ox = 0;
Oy = 0;
Ophi = 0;
Odometry_set( OdometryId, Ox, Oy, Ophi);
[Ox, Oy, Ophi] = Odometry_get( OdometryId );

Nx = 0;
Ny = 0;
Nphi = 0;

Vx=0;
Vy=0;
Vphi=0;

%OxTabl = [0];
%OyTabl = [0];
%OphiTabl = [0];

%NxTabl = [0];
%NyTabl = [0];
%NphiTabl = [0];

% départ valeurs courantes = valeurs Odométrie

x = Ox;
y = Oy;
phi = Ophi;

fin_position_for = 1;

% Etats

Valeurs_utilisees_NorthStar = 0;
Valeurs_utilisees_Odometrie = 0;
```

```
Pas_erreur_Odometrie = 0; % Odometrie egal precision pres Northstar
Valeurs_NorthStar_fausses = 0;
Captation_NorthStar_trop_faible = 0;
Pas_Captation_NorthStar = 0;



while Bumper_value(BumperId) ~= 1 && fin_position_for ~= 0

for i=1:4

Cx = posC(i,1);
Cy = posC(i,2);
Cphi = posC(i,3);

while abs(Cx - x) > Delta && Bumper_value(BumperId) ~= 1 ...
      || abs(Cy - y) > Delta && Bumper_value(BumperId) ~= 1 ...
      || abs(Cphi - phi) > Delta && Bumper_value(BumperId) ~= 1

    %tElapsed = toc(tStart);
    % If 60 seconds are elapsed then exit while loop
    %if(tElapsed >= 60 )
        %break;
    %end;

intesSpot1 = NorthStar_magSpot0(NorthStarId)/1000;
intesSpot2 = NorthStar_magSpot1(NorthStarId)/1000;
nbSpot = NorthStar_numSpotsVisible(NorthStarId);

[Ox, Oy, Ophi] = Odometry_get( OdometryId );

Nx = NorthStar_posX( NorthStarId );
Ny = NorthStar_posY( NorthStarId );
Nphi = NorthStar_posTheta( NorthStarId );

% Aucune detection

if ( NorthStar_magSpot0(NorthStarId) ~= 0 && NorthStar_magSpot1(NorthStarId) ~= 0 ...
     && NorthStar_numSpotsVisible(NorthStarId) ~= 0)

    Problemes_de_detection = 0;

else

    Problemes_de_detection = 1;

end


% pas erreur captation de Northstar

if (Problemes_de_detection == 0)
```

```
% Detection trop faible reference Odometrie

if (NorthStar_magSpot0(NorthStarId)/1000 > 3 && NorthStar_magSpot1(NorthStarId)/1000
> 3 ...
    && NorthStar_numSpotsVisible(NorthStarId) == 2)
    %&& NorthStar_roomId(NorthStarId) == NorthStar_setRoomId(NorthStarId, 3))

% valeurs Northstar cohérentes avec vitesse

if (Vx~=0 || Vy~=0 || Vphi~=0) && ...
        (Nx~=0 && Ny~=0 && Nphi~=0) ...
        && (abs(Ox - Nx) < 1000 && abs(Oy - Ny) < 1000 && abs(Ophi - Nphi) < 1000)

% erreur entre valeurs Odometrie et Northstar trop importante

if (abs(Ox - Nx) > erreur_O_N && abs(Oy - Ny) > erreur_O_N ...
    || abs(Ox - Nx) > erreur_O_N && abs(Ophi - Nphi) > erreur_O_N ...
    || abs(Oy - Ny) > erreur_O_N && abs(Ophi - Nphi) > erreur_O_N)

        x = Nx;
        y = Ny;
        phi = Nphi;
        Delta = DeltaN; % en mm % precision capteur Northstar

        % affichage Command Window
        if Valeurs_utilisees_NorthStar == 0
        Valeurs_utilisees_NorthStar = 1
        end
        Valeurs_utilisees_Odometrie = 0;

        Pas_erreur_Odometrie = 0;
        Valeurs_NorthStar_fausses = 0;
        Captation_NorthStar_trop_faible = 0;
        Pas_Captation_NorthStar = 0;

else
        x = Ox;
        y = Oy;
        phi = Ophi;
        Delta = DeltaO; % en mm % precision capteur Odometrie
        Valeurs_utilisees_NorthStar = 0;
        Valeurs_utilisees_Odometrie = 1;

        % affichage Command Window
        if Pas_erreur_Odometrie == 0
        Pas_erreur_Odometrie = 1 % Odometrie egal precision pres Northstar
        end
        Valeurs_NorthStar_fausses = 0;
        Captation_NorthStar_trop_faible = 0;
        Pas_Captation_NorthStar = 0;

end

else
        x = Ox;
```

```
        y = Oy;
        phi = Ophi;
        Delta = DeltaO; % en mm % precision capteur Odometrie
        Valeurs_utilisees_NorthStar = 0;
        Valeurs_utilisees_Odometrie = 1;

        Pas_erreur_Odometrie = 0; % Odometrie egal precision pres Northstar
        % affichage Command Window
        if Valeurs_NorthStar_fausses == 0
        Valeurs_NorthStar_fausses = 1
        end
        Captation_NorthStar_trop_faible = 0;
        Pas_Captation_NorthStar = 0;

end

else
        x = Ox;
        y = Oy;
        phi = Ophi;
        Delta = DeltaO; % en mm % precision capteur Odometrie
        Valeur_utilisees_NorthStar = 0;
        Valeur_utilisees_Odometrie = 1;

        Pas_erreur_Odometrie = 0; % Odometrie egal precision pres Northstar
        Valeurs_NorthStar_fausses = 0;
        % affichage Command Window
        if Captation_NorthStar_trop_faible == 0
        Captation_NorthStar_trop_faible = 1
        end
        Pas_Captation_NorthStar = 0;

end

else
        x = Ox;
        y = Oy;
        phi = Ophi;
        Delta = DeltaO; % en mm % precision capteur Odometrie
        Valeurs_utilisees_NorthStar = 0;
        Valeurs_utilisees_Odometrie = 1;

        Pas_erreur_Odometrie = 0; % Odometrie egal precision pres Northstar
        Valeurs_NorthStar_fausses = 0;
        Captation_NorthStar_trop_faible = 0;
        % affichage Command Window
        if Pas_Captation_NorthStar == 0
        Pas_Captation_NorthStar = 1
        end

end

% vitesse selon x

distance_x = Cx - x;
```

```
if (0 <= abs(distance_x) && abs(distance_x) < 2)
        Vx = 0 * sign(distance_x);
elseif (2 <= abs(distance_x) && abs(distance_x) < 20)
        Vx = 20 * sign(distance_x);
elseif (20 <= abs(distance_x) && abs(distance_x) < 40)
        Vx = 40 * sign(distance_x);
elseif (40 <= abs(distance_x) && abs(distance_x) < 100)
        Vx = 60 * sign(distance_x);
else
        Vx = 300 * sign(distance_x);
end

% vitesse selon y

distance_y = Cy - y;

if (0 <= abs(distance_y) && abs(distance_y) < 2)
        Vy = 0 * sign(distance_y);
elseif (2 <= abs(distance_y) && abs(distance_y) < 20)
        Vy = 20 * sign(distance_y);
elseif (20 <= abs(distance_y) && abs(distance_y) < 40)
        Vy = 40 * sign(distance_y);
elseif (40 <= abs(distance_y) && abs(distance_y) < 100)
        Vy = 60 * sign(distance_y);
else
        Vy = 300 * sign(distance_y);
end

% vitesse selon phi

distance_phi = Cphi - phi;

if (0 <= abs(distance_phi) && abs(distance_phi) < 2)
        Vphi = 0 * sign(distance_phi);
elseif (2 <= abs(distance_phi) && abs(distance_phi) < 20)
        Vphi = 10 * sign(distance_phi);
else
        Vphi = 50 * sign(distance_phi);
end

%[ Vm1, Vm2, Vm3 ] = OmniDrive_getVelocities(OmniDriveId, Vx, Vy, Vphi);

    OmniDrive_setVelocity(OmniDriveId, Vx, Vy, Vphi);

%Oxtabl = [OxTabl; Ox];
%Oytabl = [OyTabl; Oy];
%Ophitabl = [OphiTabl; Ophi];

%Nxtabl = [NxTabl; Nx];
%Nytabl = [NyTabl; Ny];
%Nphitabl = [NphiTabl; Nphi];

end
```

```
    OmniDrive_setVelocity(OmniDriveId, 0, 0, 0);

i = i + 1

end

fin_position_for = 0;

end

%disconnection from Robotino

Com_disconnect(ComId);

Bumper_destroy(BumperId);
Motor_destroy( Motor0Id );
Motor_destroy( Motor1Id );
Motor_destroy( Motor2Id );

NorthStar_destroy(NorthStarId);
Odometry_destroy(OdometryId );
OmniDrive_destroy(OmniDriveId);

Com_destroy(ComId);

fin_programme = 1

   % save sensors, save inputs, save outputs,
```

Document décrivant :

Programme d'analyse des enregistrements codé sur Matlab : Analyse_Sorties.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              Recuperation et analyse données                          %
%                  Odométrie et Northstar                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clear
clc

% Positions Northstar

x_Northstar = textread('x_Northstar.txt');
y_Northstar = textread('y_Northstar.txt');
phi_Northstar = textread('phi_Northstar.txt');

% Positions Odométrie

x_Odometrie = textread('x_Odometrie.txt');
y_Odometrie = textread('y_Odometrie.txt');
phi_Odometrie = textread('phi_Odometrie.txt');

% Taille des matrice

nb_ligne_x_Nor = length(x_Northstar(:,1));
nb_ligne_y_Nor = length(y_Northstar(:,1));
nb_ligne_phi_Nor = length(phi_Northstar(:,1));

nb_ligne_x_Odo = length(x_Odometrie(:,1));
nb_ligne_y_Odo = length(y_Odometrie(:,1));
nb_ligne_phi_Odo = length(phi_Odometrie(:,1));

% Définition du pas de l'échelle

pas_x = nb_ligne_x_Odo/nb_ligne_x_Nor;
pas_y = nb_ligne_y_Odo/nb_ligne_y_Nor;
pas_phi = nb_ligne_phi_Odo/nb_ligne_phi_Nor;

% Representation graphique de X

figure
title('Representation graphique de X');
xlabel('indice du point de mesure');
ylabel('valeur en mm');
hold on;
plot([1:pas_x:nb_ligne_x_Odo], x_Northstar, 'r')
hold on;
plot([1:nb_ligne_x_Odo], x_Odometrie, 'b')
legend('valeur de x avec Nortstar', 'valeur de x par Odométrie')
```

```
% Representation graphique de Y

figure
title('Representation graphique de Y');
xlabel('indice du point de mesure');
ylabel('valeur en mm');
hold on;
plot([1:pas_y:nb_ligne_y_Odo], y_Northstar, 'r')
hold on;
plot([1:nb_ligne_y_Odo], y_Odometrie, 'b')
legend('valeur de y avec Nortstar', 'valeur de y par Odométrie')

% Representation graphique de Phi

figure
title('Representation graphique de Phi');
xlabel('indice du point de mesure');
ylabel('valeur en degres');
hold on;
plot([1:pas_phi:nb_ligne_phi_Odo], phi_Northstar, 'r')
hold on;
plot([1:nb_ligne_phi_Odo], phi_Odometrie, 'b')
legend('valeur de phi avec Nortstar', 'valeur de phi par Odométrie')

% Ecart X

% Initiatilisation ecart X

new_x_Odometrie = [0];
x_error = [0];
i = 1;

while length(new_x_Odometrie(:,1)) < length(x_Northstar(:,1))
    new_x_Odometrie = [new_x_Odometrie; x_Odometrie(i)];
    i = i + fix(nb_ligne_x_Odo/nb_ligne_x_Nor);
end

length(x_Northstar(:,1));
length(x_Odometrie(:,1));
length(new_x_Odometrie(:,1));

% Calcul ecart X

for i = 1:length(new_x_Odometrie(:,1))
    if sign(x_Northstar(i,1)) == sign(new_x_Odometrie(i,1))
        dist_x_error = sqrt( (x_Northstar(i,1) - new_x_Odometrie(i,1))^2 );
    else
        if x_Northstar(i,1) > new_x_Odometrie(i,1)
            dist_x_error = sqrt( (x_Northstar(i,1) - new_x_Odometrie(i,1))^2 );
        else
            dist_x_error = sqrt( (new_x_Odometrie(i,1) - x_Northstar(i,1))^2 );
        end
    end
    x_error = [x_error; dist_x_error];
```

```
end

x_error;

x_error_max = max(x_error)
x_error_min = min(x_error)
x_error_moy = mean(x_error)

% Ecart Y

% Initiatilisation ecart Y

new_y_Odometrie = [0];
y_error = [0];
i = 1;

while length(new_y_Odometrie(:,1)) < length(y_Northstar(:,1))
    new_y_Odometrie = [new_y_Odometrie; y_Odometrie(i)];
    i = i + fix(nb_ligne_y_Odo/nb_ligne_y_Nor);
end

length(y_Northstar(:,1));
length(y_Odometrie(:,1));
length(new_y_Odometrie(:,1));

% Calcul ecart Y

for i = 1:length(new_y_Odometrie(:,1))
    if sign(y_Northstar(i,1)) == sign(new_y_Odometrie(i,1))
        dist_y_error = sqrt( (y_Northstar(i,1) - new_y_Odometrie(i,1))^2 );
    else
        if y_Northstar(i,1) > new_y_Odometrie(i,1)
            dist_y_error = sqrt( (y_Northstar(i,1) - new_y_Odometrie(i,1))^2 );
        else
            dist_y_error = sqrt( (new_y_Odometrie(i,1) - y_Northstar(i,1))^2 );
        end
    end
    y_error = [y_error; dist_y_error];
end

y_error;

y_error_max = max(y_error)
y_error_min = min(y_error)
y_error_moy = mean(y_error)

% Ecart Phi

% Initiatilisation ecart Phi

new_phi_Odometrie = [0];
phi_error = [0];
i = 1;
```

```
while length(new_phi_Odometrie(:,1)) < length(phi_Northstar(:,1))
    new_phi_Odometrie = [new_phi_Odometrie; phi_Odometrie(i)];
    i = i + fix(nb_ligne_phi_Odo/nb_ligne_phi_Nor);
end

length(phi_Northstar(:,1));
length(phi_Odometrie(:,1));
length(new_phi_Odometrie(:,1));

% Calcul ecart Phi

for i = 1:length(new_phi_Odometrie(:,1))
    if sign(phi_Northstar(i,1)) == sign(new_phi_Odometrie(i,1))
        dist_phi_error = sqrt( (phi_Northstar(i,1) - new_phi_Odometrie(i,1))^2 );
    else
        if phi_Northstar(i,1) > new_phi_Odometrie(i,1)
            dist_phi_error = sqrt( (phi_Northstar(i,1) - new_phi_Odometrie(i,1))^2 );
        else
            dist_phi_error = sqrt( (new_phi_Odometrie(i,1) - phi_Northstar(i,1))^2 );
        end
    end
    phi_error = [phi_error; dist_phi_error];
end

phi_error;

phi_error_max = max(phi_error)
phi_error_min = min(phi_error)
phi_error_moy = mean(phi_error)
```