Rapport technique

Par Lucas Bonnet,

Encadrantes de stage : Mme Colin et Mme Conteville

Stage de M1 SRD à L'Efrei Research LAB : *Automatisation d'un fingerprint à l'aide d'un robot mobile pour la localisation indoor*

2021-2022

1.Introduction :

Ce rapport technique est une aide afin de faciliter la prise en main du matériel utilisé durant mon stage et résumer le travail que j'ai pu réaliser avec le robot Pioneer 3-DX dans le cadre de mon stage nommé « Autotomatisation d'un fingerprint multi-variable à l'aide d'un robot mobile pour la localisation indoor ». Vous trouverez dans ce document, des explications sur les technologies utilisées que ce soit les logiciel ou le matériel ainsi que de l'aide pour télécharger les documents.

Comme vu dans la table des matières de la page ci-dessus. Ce rapport commencera par vous expliquez comment installer tous les logiciels nécessaires pour utiliser le robot, ensuite vous aurez des informations sur le robot en lui-même, puis vous aurez les détails sur les tests effectués. Vous trouverez après cela des informations sur la prise en main du matériel RFID. Pour finir, je vous parlerai de la synchronisation des programmes.

Un dossier existe avec une machine virtuelle nommée *Clone of UbuntuROS* qui contient tous les logiciels déjà installés et les programmes créés jusqu'à présent. Si vous avez cette machine virtuelle, vous pouvez passer directement à la partie *2.2.1Test après installation.*

2. Prise en main des logiciels :

Afin de programmer le robot, vous aurez besoin de plusieurs logiciels, cette partie est faite pour vous aider dans l'installation de ces logiciels et vous facilitez leurs prises en main.

1. Installation :

1. ROS :

Lien du site : https://www.ros.org/

ROS est un ensemble de librairies qui a été créé afin de faciliter la programmation et la communication entre les robots et l'utilisateur.

Dans un premier temps, vous aurez besoin d'installer VMware afin de faire tourner une machine virtuelle (VM) sous linux. Lien : https://www.vmware.com/fr.html

L'installation de tous les programmes nécessaires pour ROS est longue, une version est disponible sur moodle dans les documents du cours IA&ROS qui est donné chaque année ou dans le PDF nommé *ROS_Install+Intro* présent dans ce dossier.

Moodle : Dans le cours du semestre 9 de l'année 2020-2021 nommé *IA et ROS,* vous trouverez un PDF nommé *ROS commands*.

Lien : https://moodle.myefrei.fr/course/view.php?id=7137

Chemin à suivre : *Moodle -> 2020-2021 -> Semestre 9 -> Système intelligent et robotique -> IA et ROS -> ROS Commands (essayez d'autres années ou d'autres semestres si le chemin indiqué n'est plus disponible.)*

Dans les deux cas sur la slide 2 de ce PDF un lien est présent qui vous permettra de télécharger la VM.

Une fois VMware et la machine virtuelle téléchargée, lancer VMware puis ouvrez la VM que vous venez de télécharger.

Le mot de passe est : **turtlebot3**

Note : si votre ordinateur n'arrive pas à lancer la machine ou s'éteint au lancement de la machine virtuelle, il faut soit mettre à jour VMware soit passer à une version antérieure.

2. Aria, MobileSim, Mapper3 :

Il faut maintenant télécharger 3 logiciels créés par l'entreprise qui a fabriqué le robot :

Aria est une librairie en C++ qui vous facilitera la programmation du robot.

MobileSim permet de simuler les programmes que vous avez réalisés.

Mapper3 permet de créer des cartes pour la simulation.

Ces trois logiciels sont normalement présents dans le dossier si ce n'est pas le cas plusieurs possibilités vous sont disponibles.

L'entreprise créant le robot n'existe plus, la première option est donc d'utiliser un site d'archive de page web :

https://web.archive.org/web/20110826074242/http://robots.mobilerobots.com/wiki/Main_Page

Vous trouverez des informations sur le robot et les programmes, certains programmes ne sont pas téléchargeables, à cause de restriction.

La deuxième possibilité est sûrement la plus simple est ce lien dans lequel vous trouverez tous les logiciels :

http://vigir.missouri.edu/~gdesouza/Research/MobileRobotics/Software/

Si vous avez téléchargé ces logiciels depuis votre ordinateur et pas depuis la VM, vous avez juste à les copier puis à les coller sur le bureau de la VM.

Afin d'installer les logiciels, il n'y a plus qu'à les ouvrir avec Software Install, qui permet d'installer des logiciels sous Linux.

3. ROSARIA :

ROSAria est la librairie ROS créée pour les robots de l'entreprise Mobile Robots. Pour télécharger la librairie veuillez suivre ces étapes :

Lancez la machine virtuelle, rentrez le mot de passe (turtlebot3), ouvrez un terminal en cliquant sur l'icône ci-dessous ou en faisant un clic droit sur le bureau et *open in Terminal.*

Icône du terminal

Dans un premier temps, mettez le clavier en azerty en tapant la commande suivante dans le terminal : *setxkbmap fr*

Cette commande sera à effectuer à chaque fois que vous utilisez la machine virtuelle.

Note : Il y a possibilité d'enregistrer les paramètres clavier pour ne pas avoir à le faire à chaque fois.

Après avoir téléchargé et installé la librairie Aria entrez les 3 lignes ci-dessous (appuyez sur entrée entre chaque ligne). La première permet de se déplacer dans les fichiers. Les deux autres permettent de mettre en place et de rendre utilisable la librairie. Les lignes 2 et 3 mettent quelque temps à s'effectuer.

```
cd /usr/local/Aria
make clean
make -j4
```

Ensuite, rentrez ces deux autres lignes de commandes afin de télécharger au bon endroit la librairie présente sur GitHub :

```
cd ~/catkin_ws/src
git clone https://github.com/amor-ros-pkg/rosaria.git
```

Pour finir rentrer les deux lignes suivantes afin de mettre à jour les modifications apportées.

cd ~/catkin_ws/

catkin_make

Note : si une erreur est affichée durant cette installation, il vous faudra sûrement installer des fonctionnalités tierces. Recherchez l'erreur sur Internet, vous trouverez facilement une réponse.

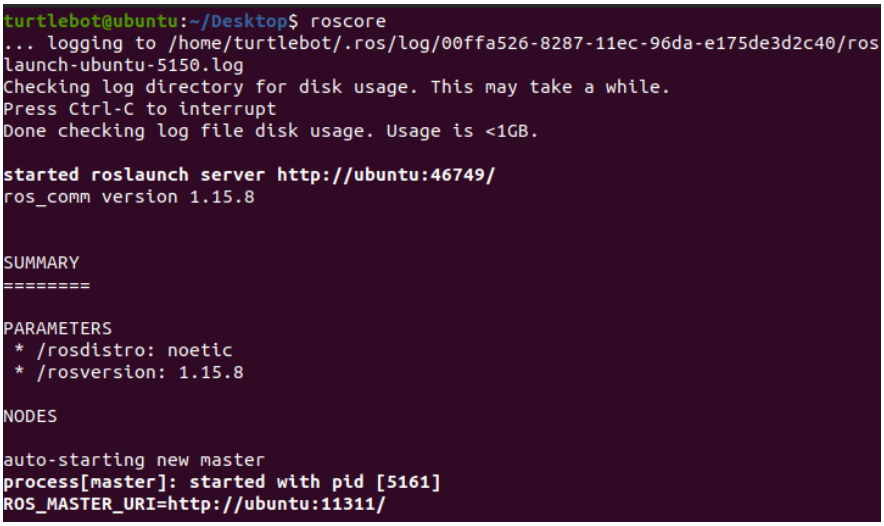
2. Premiers pas :

1. Test après installation :

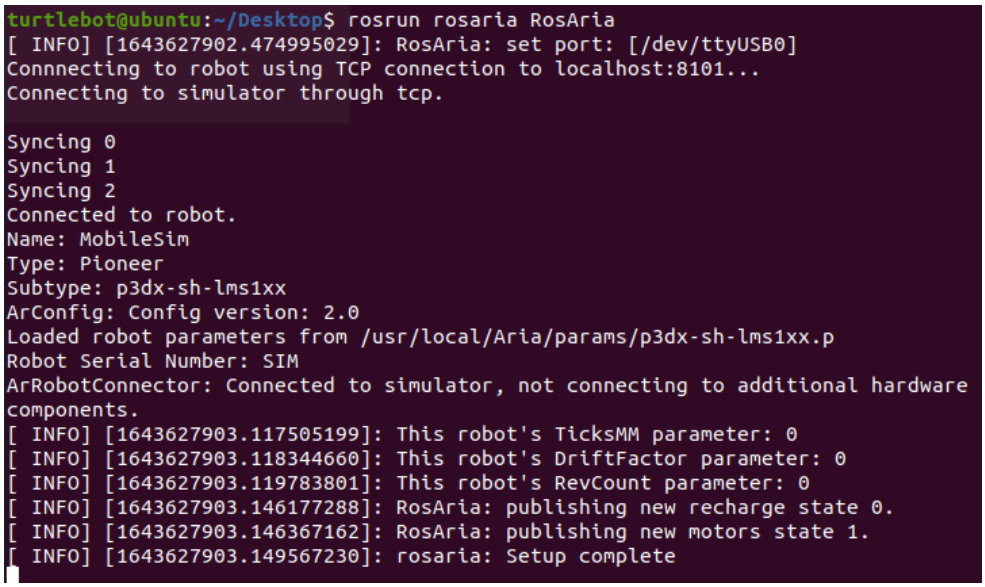
Pour tester si tout est bien installé, lancez Mobilesim en cliquant sur le menu en bas à gauche du bureau puis en cherchant avec la barre de recherche ou dans les applications affichées. Sélectionnez *No Map* en bas à droite.

Icône de MobileSim

Puis ouvrez deux fenêtres. Dans la première, écrivez roscore et appuyez sur entrée. Cette commande va permettre de lancer ROS comme ci-dessous.

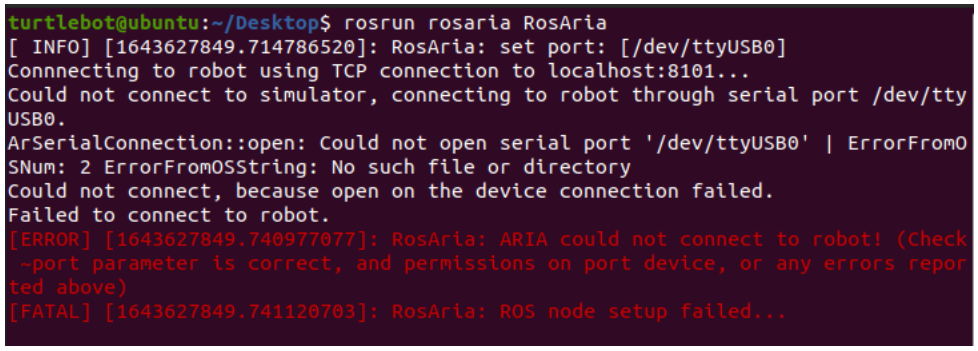


Garder la première fenêtre ouverte et dans la deuxième fenêtre écrivez rosrun rosaria RosAria comme sur la photo ci-dessous.



Quelques informations : rosrun est la commande ROS permettant de lancer un programme, rosaria est la librairie que nous voulons utiliser et RosAria est le nom de l'exécutable que nous voulons lancer. Cet exécutable est lié à un ou plusieurs programmes.

Si MobileSim n'est pas lancer ou qu'aucun robot n'est trouvé, voici le message d'erreur que vous pourrez lire :



Si aucune erreur n'est apparue alors votre installation est réussite. Si ce n'est pas le cas, une étape n'a pas du bien se faire durant l'installation.

Afin d'arrêter les deux programmes lancés depuis les terminaux, il ne vous suffit que de faire CTRL+C.

2. Prise en main d'Aria et ROSAria :

Aria et ROSAria sont des librairies créées afin de faciliter la programmation, la communication et la connexion avec le robot. Ces deux librairies fonctionnent ensemble.

Si vous allez dans la librairie Aria *Files ->Other Locations -> usr->local->Aria->exemples* vous trouverez des exemples de programme qui vous permettront de comprendre comment sont fait les codes. Le mieux est de commencer par les programmes suivant dans l'ordre *simpleConnect, simpleMotionCommands, gotoActionExemple.*

Si vous voulez tester le fonctionnement d'un programme, copier le programme que vous voulez essayer, ici, nous prendrons comme exemple le fichier *simpleConnect*. Puis déplacez-vous dans le dossier suivant *Home -> catkin_ws -> src->rosaria*. Coller le programme, dans le dossier puis ouvrez le fichier nommé *CMakeLists.txt* présent au même endroit. Ce fichier a pour but de paramétrer la librairie ROS, ici, nous allons lier le programme à la librairie ROS pour cela allez à la ligne 96 de ce fichier et ajouter les deux lignes encadrées en rouge sur l'image ci-dessous.

```
70  else()
71    if(EXISTS "${prefix}/include/Aria.h")
72      message("Found ${prefix}/include/Aria.h, assuming Adept ARIA source directory.")
73      add_definitions(-DADEPT_PKG)
74      include_directories(${prefix}/include)
75      link_directories(${prefix}/lib)
76    else()
77      if(EXISTS "${prefix}/include/Aria/Aria.h")
78        message("Found ${prefix}/include/Aria/Aria.h, assuming AriaCoda or repackaged ARIA.")
79        #add_definitions(-DARIACODA)
80        include_directories(${prefix}/include)
81        link_directories(${prefix}/lib)
82      else()
83        message("Aria.h not found in ${prefix}. Continuing with default header and library paths.")
84      endif()
85    endif()
86       endif()
87 endif()
88
89 add_executable(RosAria RosAria.cpp LaserPublisher.cpp)
90 add_dependencies(RosAria rosaria_gencfg)
91 add_dependencies(RosAria rosaria_gencpp)
92
93 target_link_libraries(RosAria ${catkin_LIBRARIES} ${Boost_LIBRARIES} Aria pthread dl rt)
94 set_target_properties(RosAria PROPERTIES COMPILE_FLAGS "-fPIC")
95 #set_target_properties(RosAria PROPERTIES LINK_FLAGS "-Wl")
96
97 #ajout par Lucas Bonnet
98 add_executable(simpleConnect simpleConnect.cpp)
99 target_link_libraries(simpleConnect ${catkin_LIBRARIES} ${Boost_LIBRARIES} Aria pthread dl rt)
```

Dans la ligne *add_executable (simpleConnect simpleConnect.cpp), simpleConnect* sera le nom de l'exécutable ROS, *simpleConnect.cpp* est le programme lié à cet exécutable. La deuxième ligne *target_link_libraries (simpleConnect ${catkin_LIBRARIES} ${Boost_LIBRARIES} Aria pthread dl rt)* permet de donner les librairies liées à cet exécutable. Après ces modifications, enregistrez le document, ouvrez un nouveau terminal et entrez ces deux lignes afin de se déplacer dans le dossier catkin_ws et d'actualiser les modifications apportées à la librairie ROSAria cela permettra de voir notre nouvel exécutable.

```
turtlebot@ubuntu:~$ cd ~/catkin_ws/
turtlebot@ubuntu:~/catkin_ws$ catkin_make
```

Une fois finit, vous pouvez vérifier l'ajout de l'exécutable en entrant la commande *rosrun rosaria* (avec un espace à la fin) et en appuyant sur tab deux fois.

```
turtlebot@ubuntu:~/catkin_ws$ rosrun rosaria
actionExample              Forward400               rosaria__launcher.launch
actionExample.cpp          Forward400.cpp           RosAriaTest
actionTest                 Forward600               RosAriaTest.cpp
actionTest.cpp             Forward600.cpp           simpleConnect
ArTimeToROSTime.h          Forward800               simpleConnect.cpp
boucletest                 Forward800.cpp           test
boucletest.cpp             gotoActionExample        test.cpp
BumperState.msg            gotoActionExample.cpp    UMBMARK1
Circle                     gotoActionTest           UMBMARK1.cpp
Circle.cpp                 gotoActionTest.cpp       UMBMARK2
CMakeLists.txt             improve                  UMBMARK2.cpp
connectPerso               improve.cpp              UMBMARK3
connectPerso.cpp           LaserPublisher.cpp       UMBMARK3.cpp
Forward                    LaserPublisher.h         UMBMARK4
Forward1000                libaria.rdmanifest       UMBMARK4.cpp
Forward1000.cpp            mainpage.dox             UMBMARK5
Forward1200                package.xml              UMBMARK5.cpp
Forward1200.cpp            README.md                wander
Forward200                 RosAria                  wander.cpp
Forward200 (copy).cpp      RosAria.cfg
Forward200.cpp             RosAria.cpp
turtlebot@ubuntu:~/catkin_ws$ rosrun rosaria
```

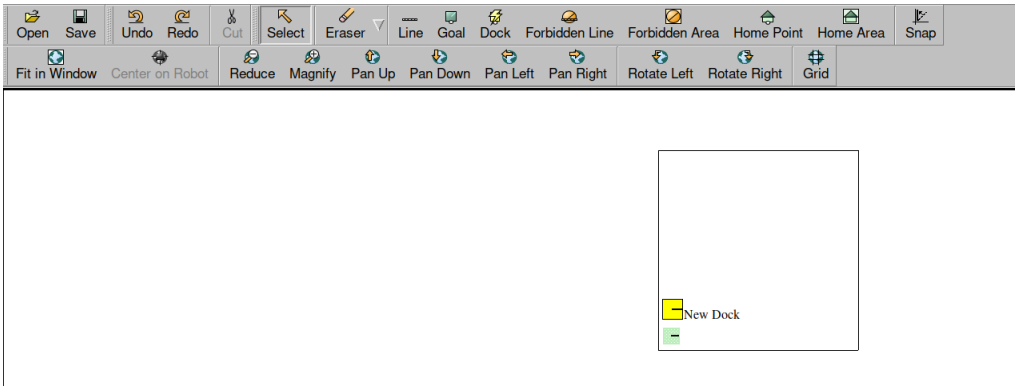
Vous pourrez voir comme ci-dessus le lien *simpleConnect* qui a été ajouté. Pour lancer votre programme, il ne vous suffit plus qu'à en compléter la commande comme ceci :

rosrun rosaria simpleConnect.

Pour tester votre programme, vous avez deux solutions, sur le robot directement ou sur un simulateur. La première possibilité n'est pas une bonne idée, car vous ne connaissez pas la réaction du robot. C'est pourquoi il est préférable de tester d'abord sur un simulateur.

3. Prise en main de MobileSim et Mapper 3 :

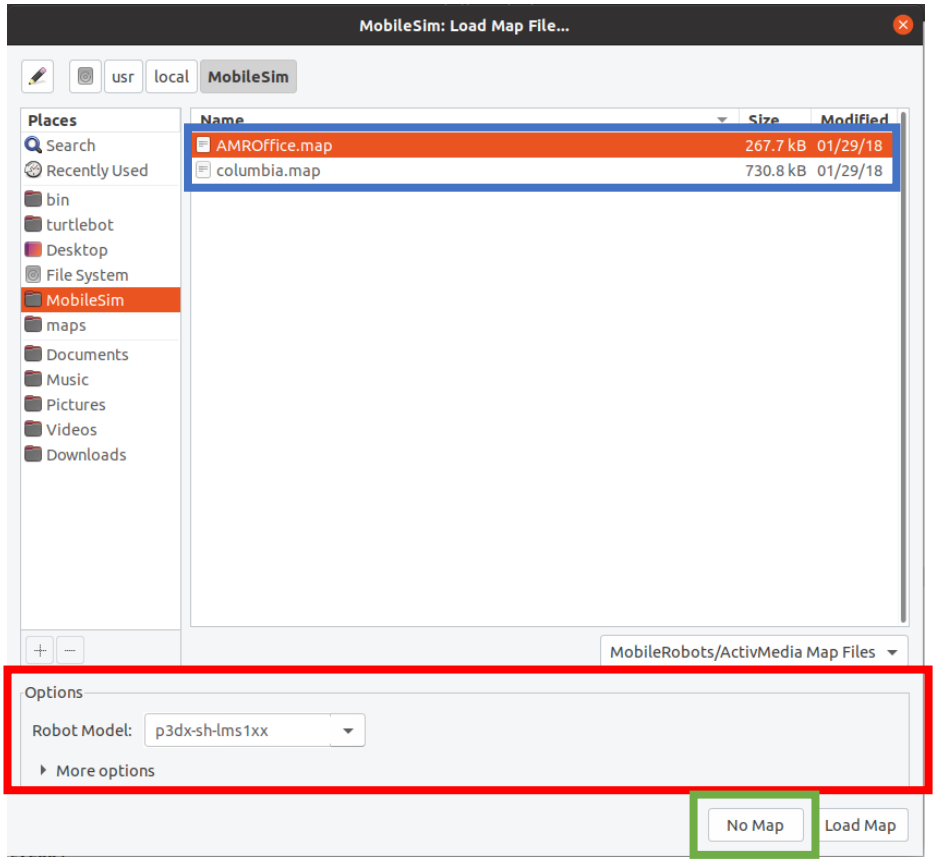
Vous pouvez utiliser mapper3 pour créer des cartes et simuler avec une carte qui ressemble à votre environnement. Que ce soit une salle de classe, un hall ou autre.



Ainsi, afin de tester un programme, lancez MobileSim et choisissez les options pour la carte et les options du robot.

Pour la carte, vous pouvez choisir entre deux cartes déjà présentent sur MobileSim qui sont AMROffice et columbia, sinon vous pouvez utiliser les cartes que vous avez créées (encadrer en bleu ci-dessous) ou vous pouvez lancer sans carte en cliquant sur le bouton *No Map* (encadrer en vert ci-dessous).

Vous pouvez aussi modifier les options et les paramètres de votre/vos robots (encadrer en rouge ci-dessous).



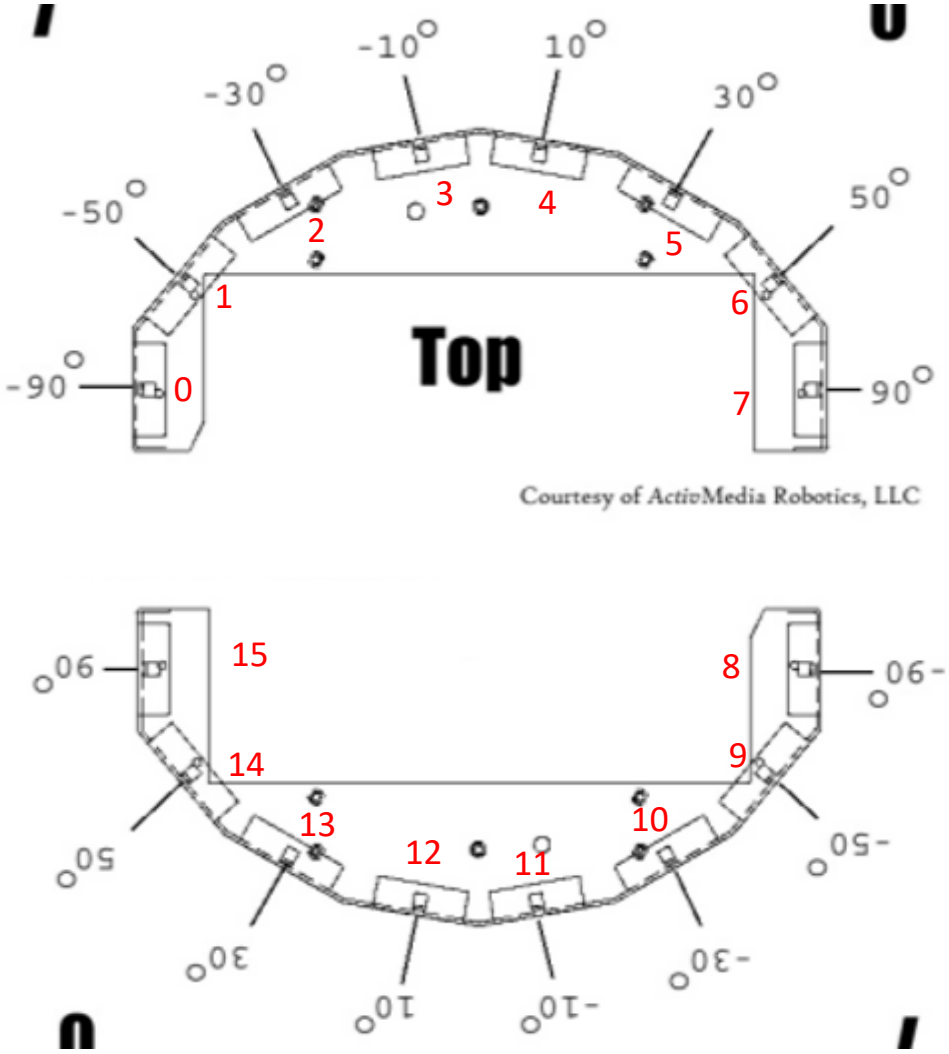
Si vous prenez l'option *No Map* et que vous lancez le programme simpleConnect vous pourrez voir dans la zone (encadrer en rouge ci-dessous) les informations en lien avec la connexion entre vous et le simulateur.

Après avoir essayé votre programme et vu que tout fonctionne correctement vous pouvez essayer votre programme sur votre robot et voir le vrai comportement du robot.

3.Prise en main du robot

Le robot Pioneer 3 DX est un robot qui a été créé par l'entreprise Mobile Robot, cette entreprise a dû fermer, c'est pour cela qu'il est difficile de trouver les logiciels et la documentation.

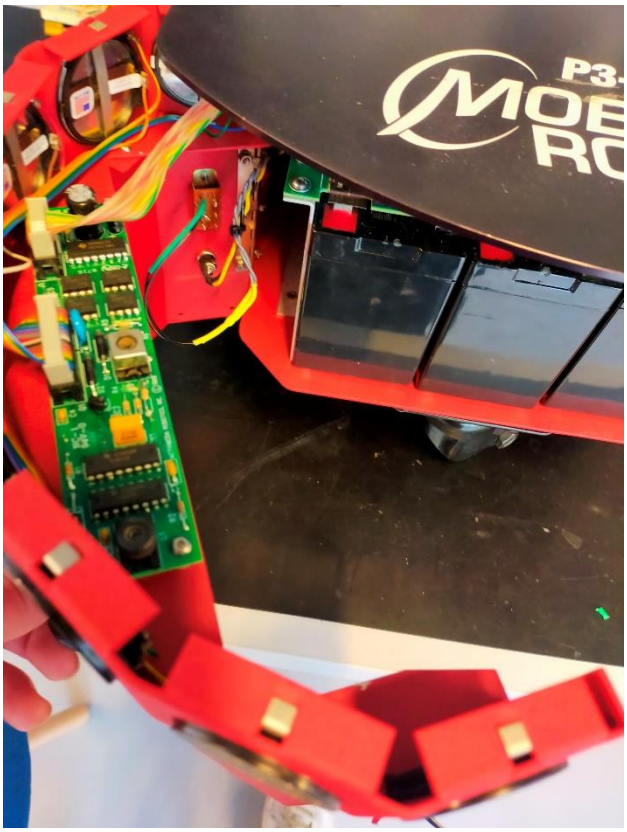
Ce robot équipé de deux roues motrices et une roue secondaire située à l'arrière. Il est aussi équipé de bumpers, d'un LiDAR et de 16 capteurs ultrason qui sont disposés tout autour du robot comme sur l'image ci-dessous. Ces capteurs ultrasons permettent de récupérer la distance entre le robot et les obstacles et nous permettent au robot de ne pas percuter d'objet ou de mur par exemple.



Vous pouvez ouvrir le robot grâce à une porte à l'arrière de celui-ci, vous avez possibilité aussi de voir l'intérieur du robot en enlevant la partie en plastique située sur le dessus.

En ouvrant la porte, vous pourrez avoir accès à la partie détection reliée aux capteurs ultrason de l'arrière du robot, aux batteries et au système d'alimentation. Un problème est survenu avec l'interrupteur pour allumer le robot, la soudure c'est cassé à force d'ouvrir la porte.

Si le robot ne s'allume pas regarder si la soudure est bien en place, si ce n'est pas le cas, il faudra changer la soudure, cela est assez simple et tout le matériel est présent à l'I-LAB.



Vous pouvez voir sur la droite les batteries, à gauche,
la carte de détection et au milieu les câbles reliés à l'alimentation.

La partie alimentation est reliée à l'interface présente sur la gauche du robot. Cette interface est constituée de LED permettant de connaître l'état du robot (batterie, communication, ...) vous avez l'interrupteur permettant d'allumer ou d'éteindre et tout en bas de quoi charger le robot. Pour finir, vous trouverez aussi le port série qu'il faut utiliser pour se connecter et transmettre les informations au robot.

Caractéristiques :

Vitesse maximum vers l'avant et l'arrière : 1,2 m/s

Vitesse de rotation max : 300 °/sec

Hauteur du robot : 237 mm

Longueur du robot : 455 mm

Largeur du robot : 381 mm

Diamètre des roues : 195 mm

4. Caractérisation de l'erreur de déplacement du robot

1. Explication

Comme expliqué juste au-dessus et comme vous pourrez le remarquer par vous-même, en lançant un programme qui fait rouler le robot sur plusieurs mètres, celui-ci dérive vers la gauche. Cette dérive durant le déplacement est un problème très fréquent sur les robots mobiles. Elle est due à plusieurs erreurs différentes que l'on peut scinder en deux catégories, les erreurs systématiques et les erreurs non-systématiques (3).

Premièrement, les erreurs systématiques sont les erreurs présentes sur tous les robots et elles vont avoir un impact à chaque fois que le robot roule. Ces erreurs peuvent venir par exemple d'une taille des roues différentes, la répartition du poids du robot ou encore de l'alignement des roues.

La deuxième catégorie sont les erreurs non-systématiques, elles ne sont pas sûres d'arriver, elles peuvent venir du fait que les deux roues ne roulent pas sur le même type de sol, de la ou des roues secondaires du robot (roues non motrices), d'objet au sol ou d'autres possibilités.

Dans notre cas, nous voulons réaliser un fingerprint, le robot doit alors se déplacer précisément et s'arrêter à des points précis afin de prendre des mesures. C'est pour cela que nous voulons supprimer cette dérive, qui, nous le savons est constitué des deux catégories d'erreurs précédemment citées. Nous pouvons voir cela, car la dérive se ressemble à chaque fois que nous utilisons le robot, mais n'est pas exactement la même, nous pouvons donc en conclure que nous avons des erreurs systématiques plus conséquentes que les erreurs non-systématiques. C'est pourquoi nous pouvons corriger cette dérive.

Nous avons alors cherché et réalisé plusieurs tests différents afin de modéliser cette erreur et pouvoir la corriger.

Les résultats des tests que nous avons faits et qui vont vous être présentés sont disponibles sur un Excel nommé *Résultats.*

2. Tests réalisés pour caractériser l'erreur

1. UMB MARK

Dans un premier temps, nous avons effectué un test nommé UMBMark, celui-ci est très souvent utilisé pour ce genre de problème et est très souvent cité (1). Il consiste à créer un carrée de 4x4m. Le robot commence d'un coin du carré et doit suivre le tracé de ce carré et revenir à sa position initiale voir ci-dessous.

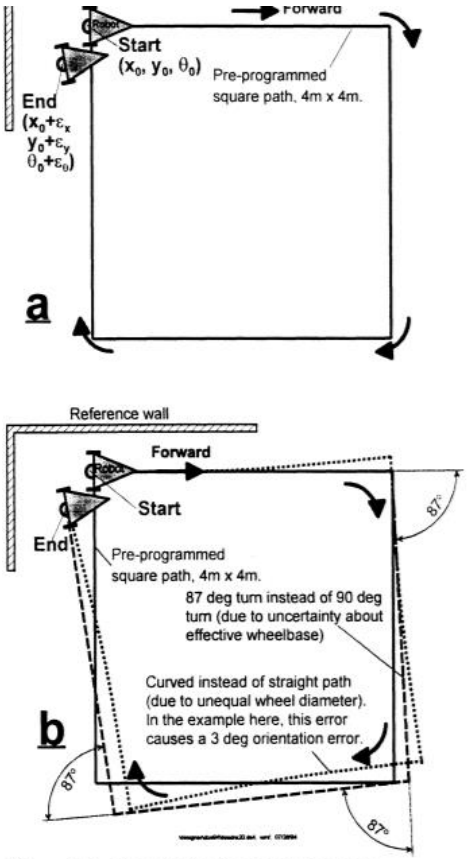


Figure 3.1: The unidirectional square path experiment.
a. The nominal path.
b. Either one of the two significant errors E_b or E_d can cause the same final position error.

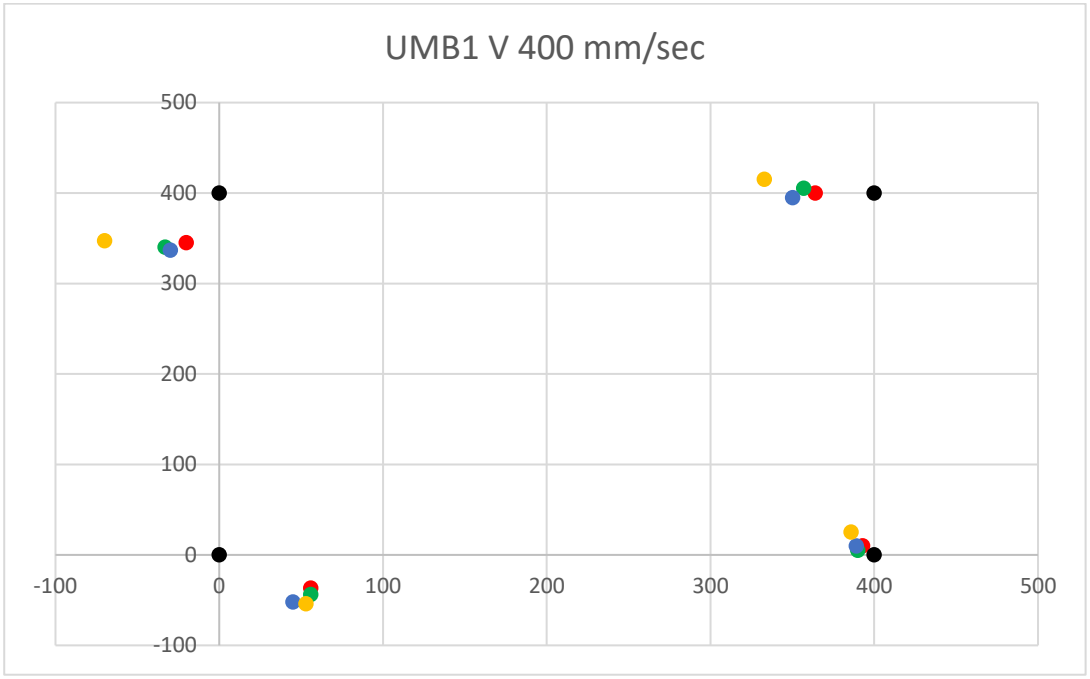
Le test permet de voir trois types d'erreurs du robot : quand il se déplace, quand il tourne et quand il s'arrête. Le robot peut en additionnant les erreurs, revenir à sa position de départ sans être précis durant son trajet. C'est pour cela que l'on récupère la position du robot à chaque coin du carré afin de voir la différence par rapport à la position où il devrait être.

Nous avons fait les tests dans une salle de classe avec 5 options différentes, chaque option a été réalisée 4 fois. Le robot part à chaque fois du point (0,0) et tourne dans le sens anti-horaire. Les résultats obtenus sont ci-dessous. Chaque couleur correspond à un des 4 tests, les points noirs correspondent aux coins des carrés et à l'endroit où le robot devrait s'arrêter. Nous allons d'abord tester avec 2 vitesses différentes pour savoir si la vitesse à une influence sur notre erreur, pour le troisième test, nous améliorerons la façon dont le robot effectue sa rotation. Au quatrième test, nous utilisons l'option précédente, que nous améliorerons en changeant le comportement du robot

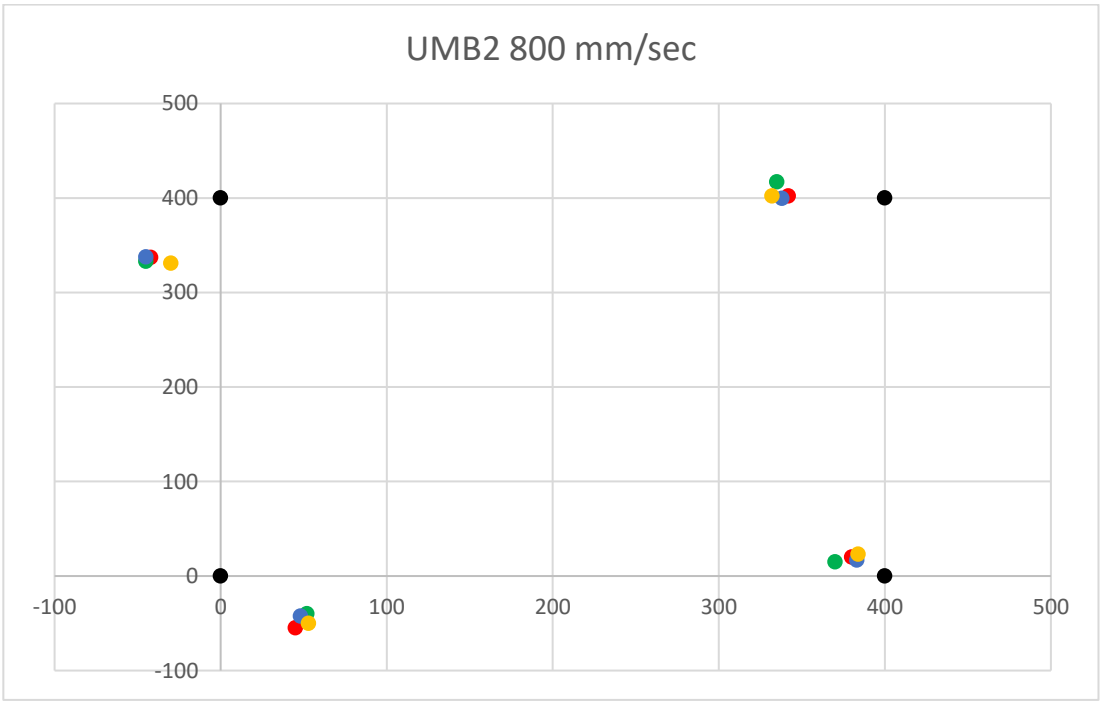
pendant son déplacement. Pour le dernier, nous changeons le comportement du robot pendant son déplacement, mais nous n'améliorons plus la façon dont tourne le robot.

Le code de chaque option est à retrouver dans le dossier UMBMARK

Option n°1



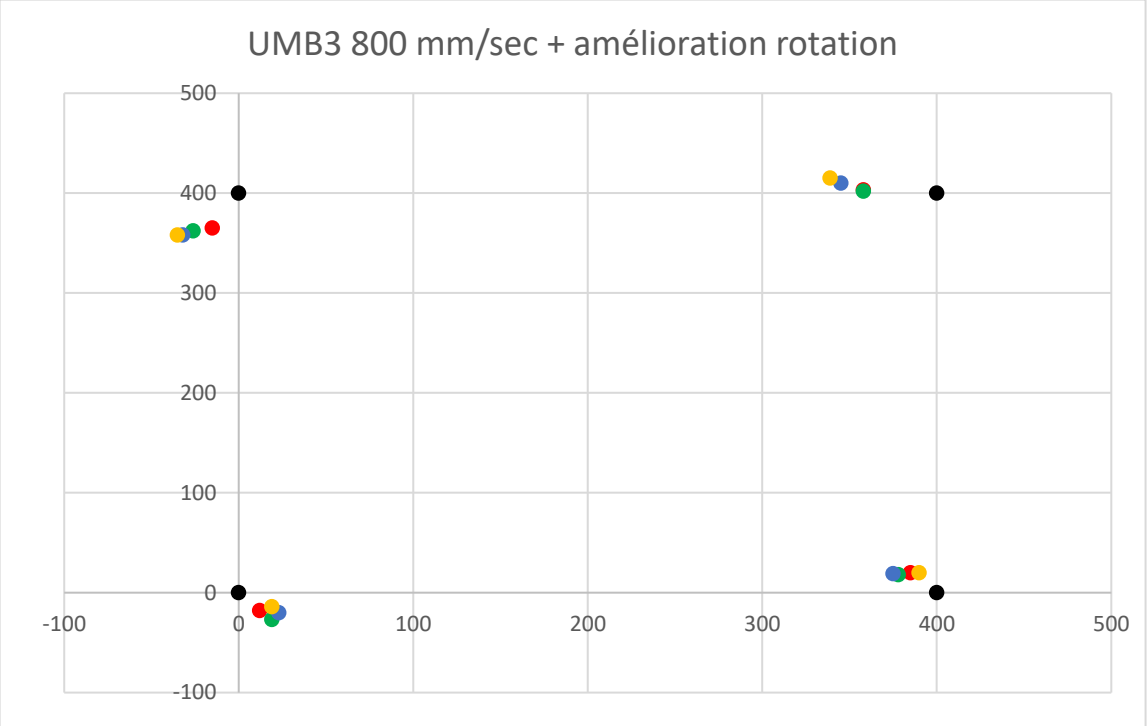
Option n°2



Les deux premiers tests ont été faits avec des vitesses différentes, mais sans autres améliorations particulières. Nous voulions voir si la vitesse influençait la façon qu'avait le robot de se comporter.

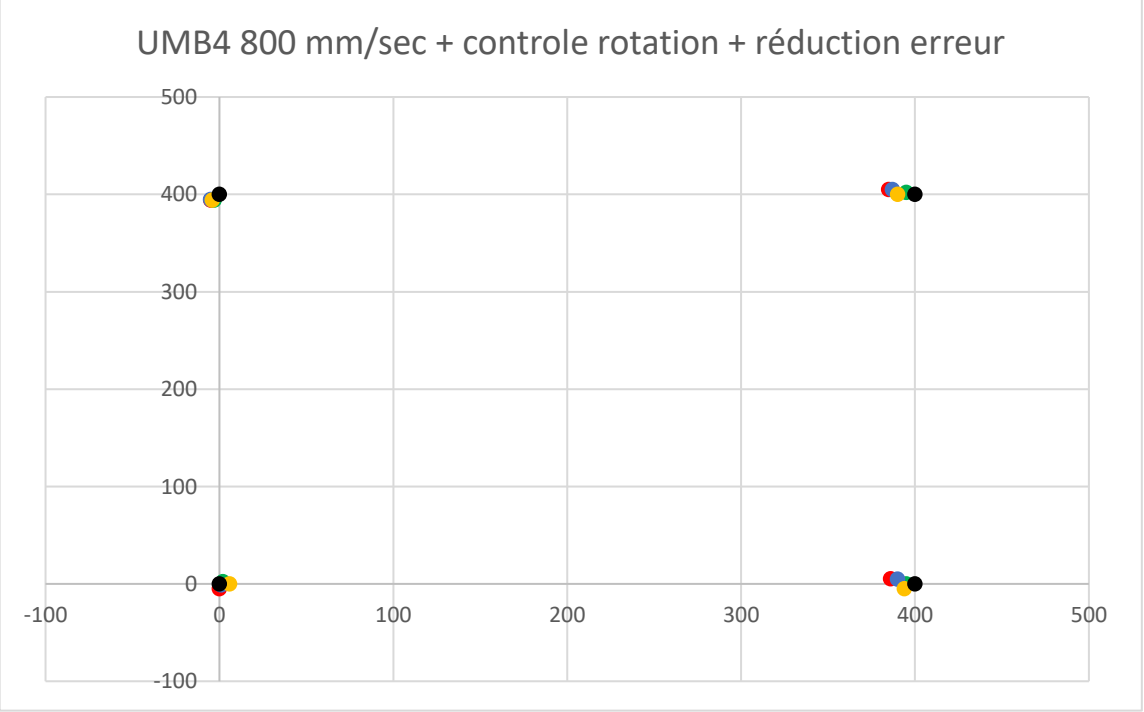
Nous pouvons voir que les résultats sont très similaires entre les deux options même si l'option N°1 est mieux. Nous pouvons aussi voir grâce au test représenté en jaune sur le graphique N°1 que le cumul d'erreurs peut vraiment donner des résultats critiques.

Option n°3



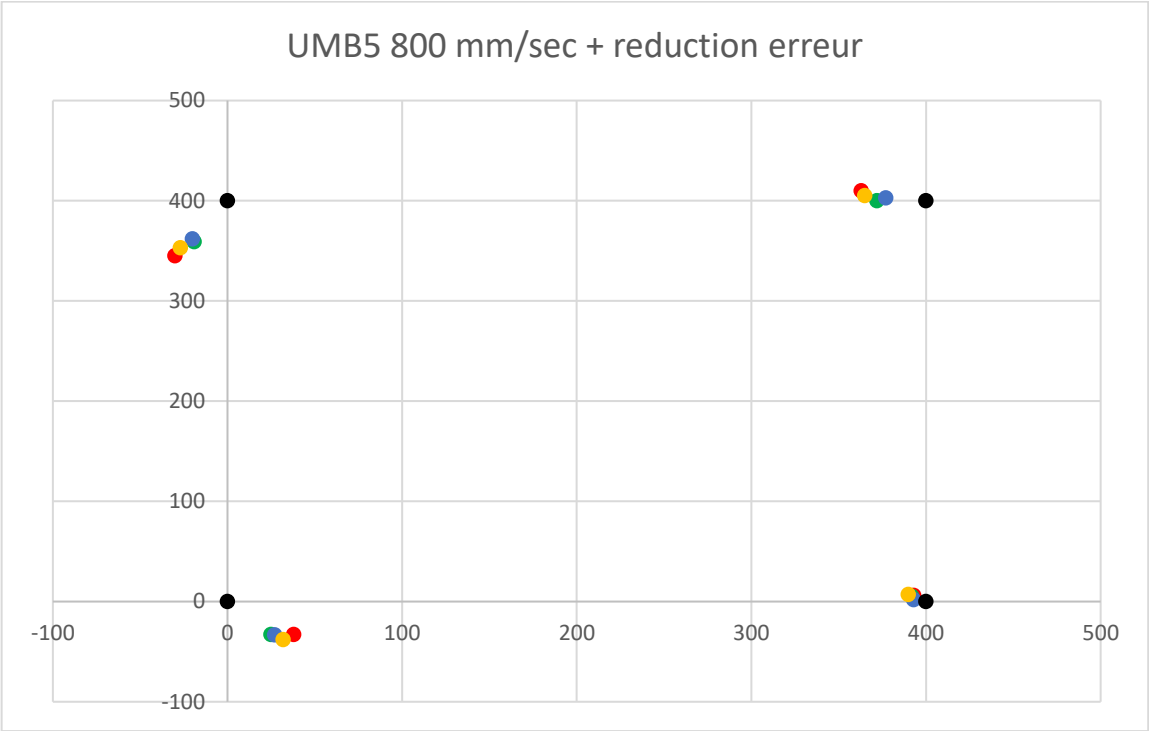
Le troisième test lui a été fait avec une amélioration quand le robot effectue une rotation. Nous avons fait en sorte que plus le robot est proche de son but plus il ralentit, grâce à ça, nous améliorons la précision.

Option n°4



Ici, nous avons en plus de l'amélioration de l'option n°3 ajouté une vitesse de rotation de 1° durant 1 m pour chaque côté du carré afin de réduire l'erreur de déplacement et voir si cela améliore nos résultats.

Option n°5



Pour finir, nous avons fait nos tests en gardant l'amélioration de l'option n°4 qui impact sur l'erreur de déplacement et en enlevant l'amélioration de l'option n°3 qui impact l'erreur quand il tourne.

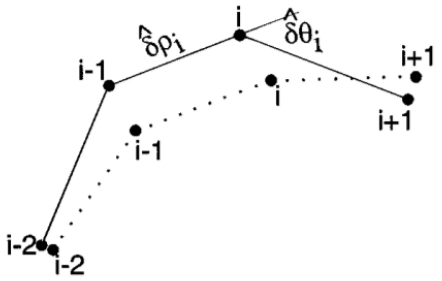
Nous pouvons voir sur les résultats des trois dernières options, une amélioration significative. Le fait d'améliorer même de façon imprécise la manière dont se comporte le robot nous donne de meilleurs résultats. L'option 3 améliore de 30 cm les résultats pris au point (0,400) et au dernier point. L'option 5, elle améliore de 20 cm. L'option 4 qui combine les deux nous donne des résultats très bons et ne dépasse pas 16 cm d'erreur sur tous les tests et est en moyenne à 8 cm des points où il devrait être.

Ces tests sont très encourageants, mais ont un inconvénient pour nous, nous voulons modéliser l'erreur de déplacement, or le fait que le test soit fait en carré nous oblige à prendre en compte l'erreur due à l'imprécision de la rotation et des arrêts durant lesquels le robot se dandine. C'est pour cela, que nous ne pouvons pas modéliser l'erreur avec ce type de test. Mais ils nous ont permis d'avoir un premier aperçu sur les problèmes du robot.

2. Test en ligne droite avec arrêt

La deuxième batterie de tests effectués, sont faits en allant en ligne droite avec des arrêts réguliers (2) définis et marqués au sol. Dans notre cas, le robot va parcourir 15 m au total et s'arrêter tous les mètres. Nous avons fait le test dans un couloir de 2 m de large, en plaçant toujours au même point de

départ le robot. Il commence à 30 cm du mur de droite devant la salle PAC-MAN, nous prendrons la distance avec le mur de droite à chaque arrêt. Le fait d'arrêter le robot à des points intermédiaires nous permet de voir comment se comporte le robot et nous permet aussi de voir l'évolution de l'erreur en fonction de la distance comme sur l'image ci-dessous.

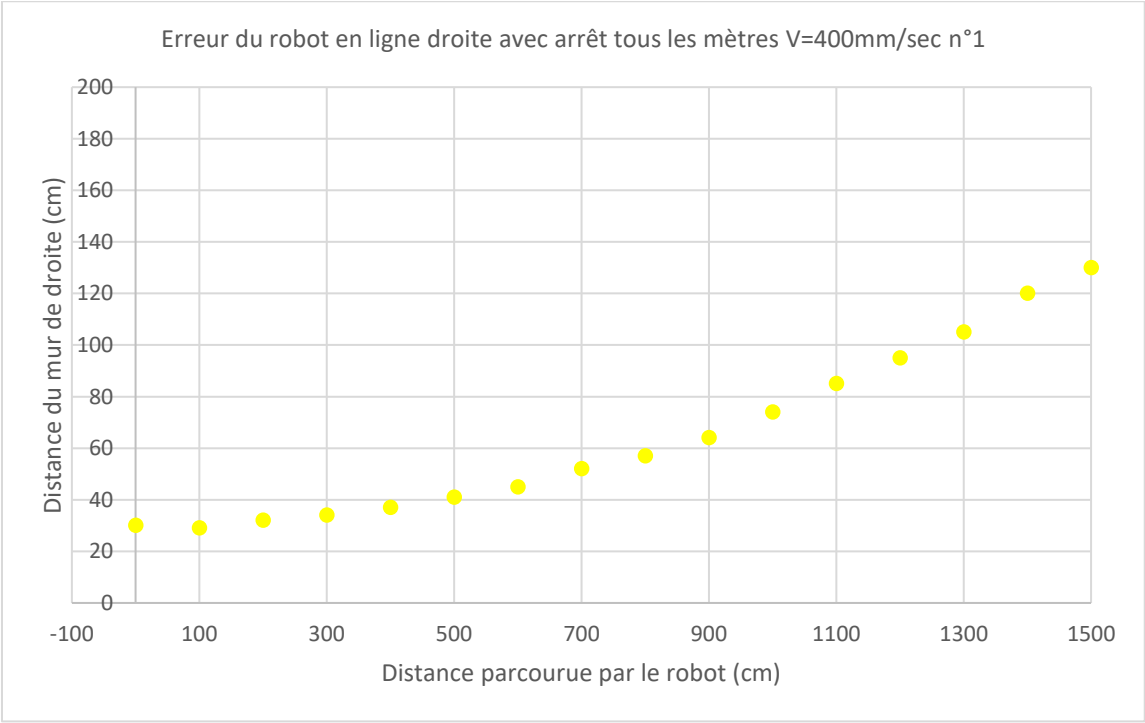


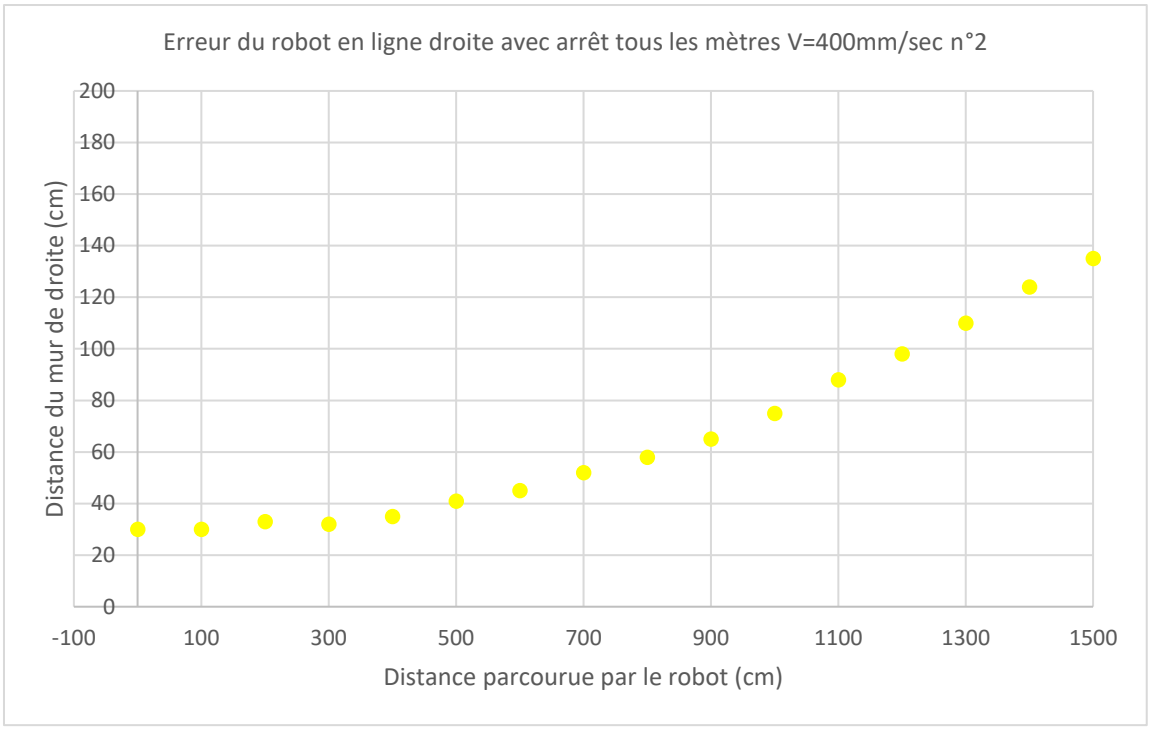
Ligne continue : trajet effectué par le robot

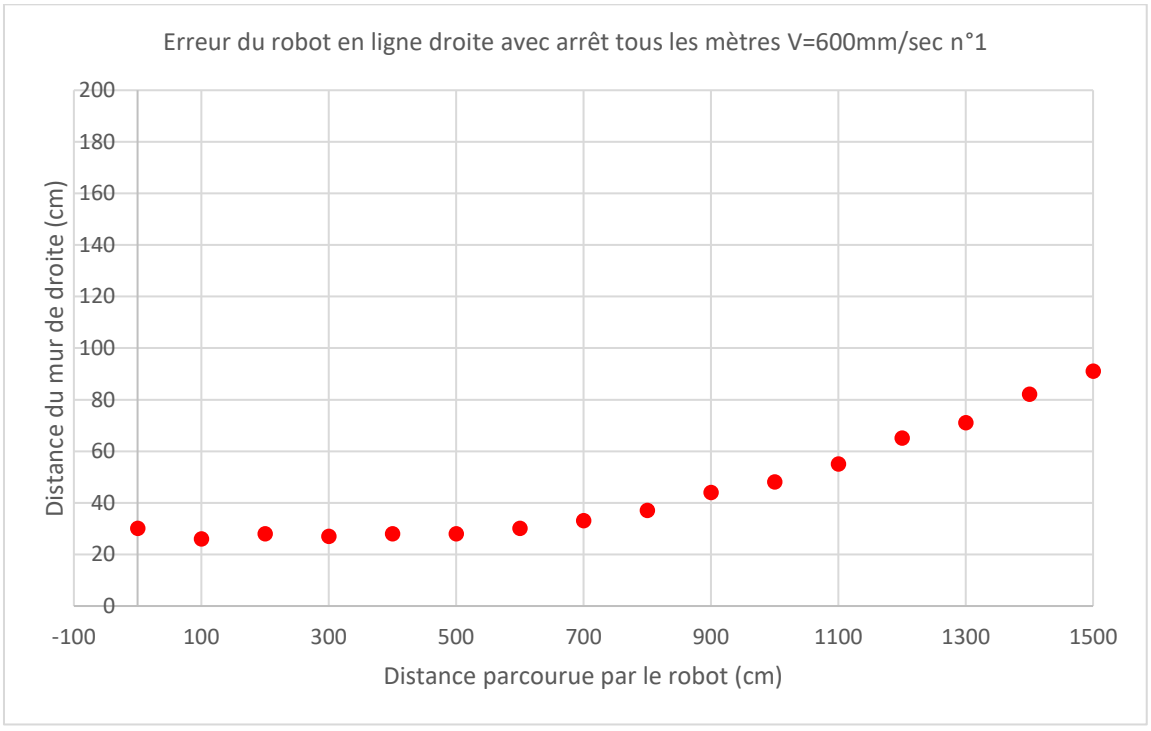
Ligne pointillée : trajet demandé

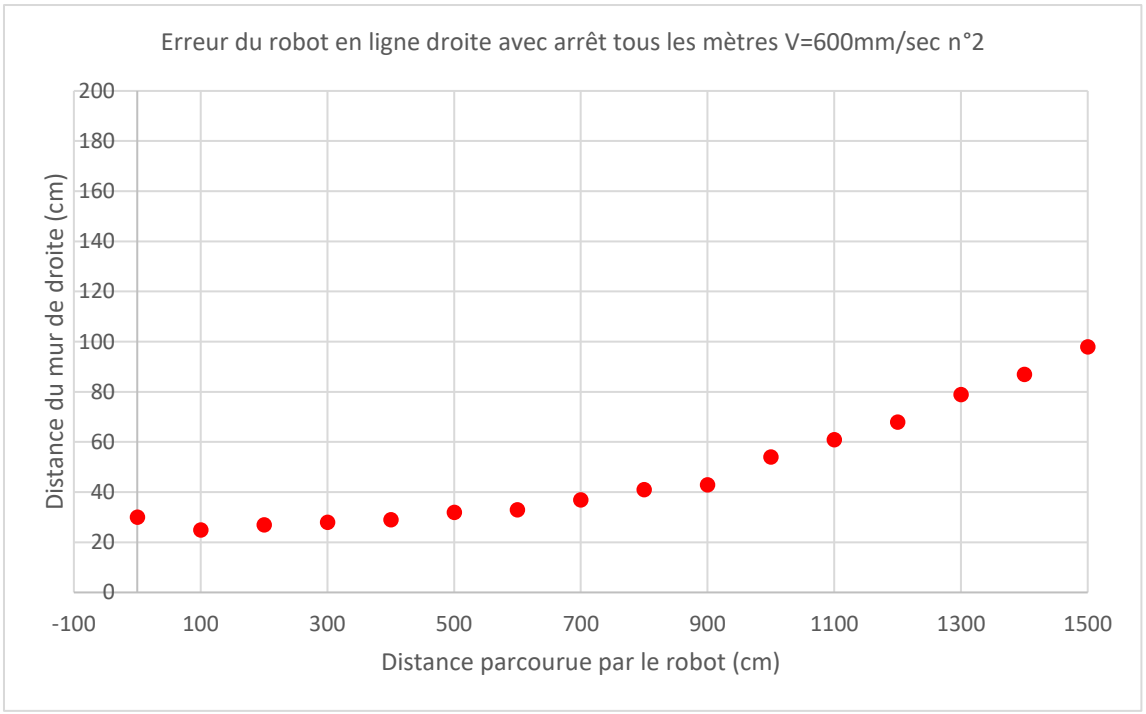
Les erreurs de rotation présentes pour les tests UMBmark ne le sont plus. Il est alors plus simple de faire la modélisation de l'erreur. Nous avons réalisé ces tests deux fois avec trois vitesses différentes 400, 600 et 800 mm/sec encore une fois pour voir si la vitesse influence notre erreur.

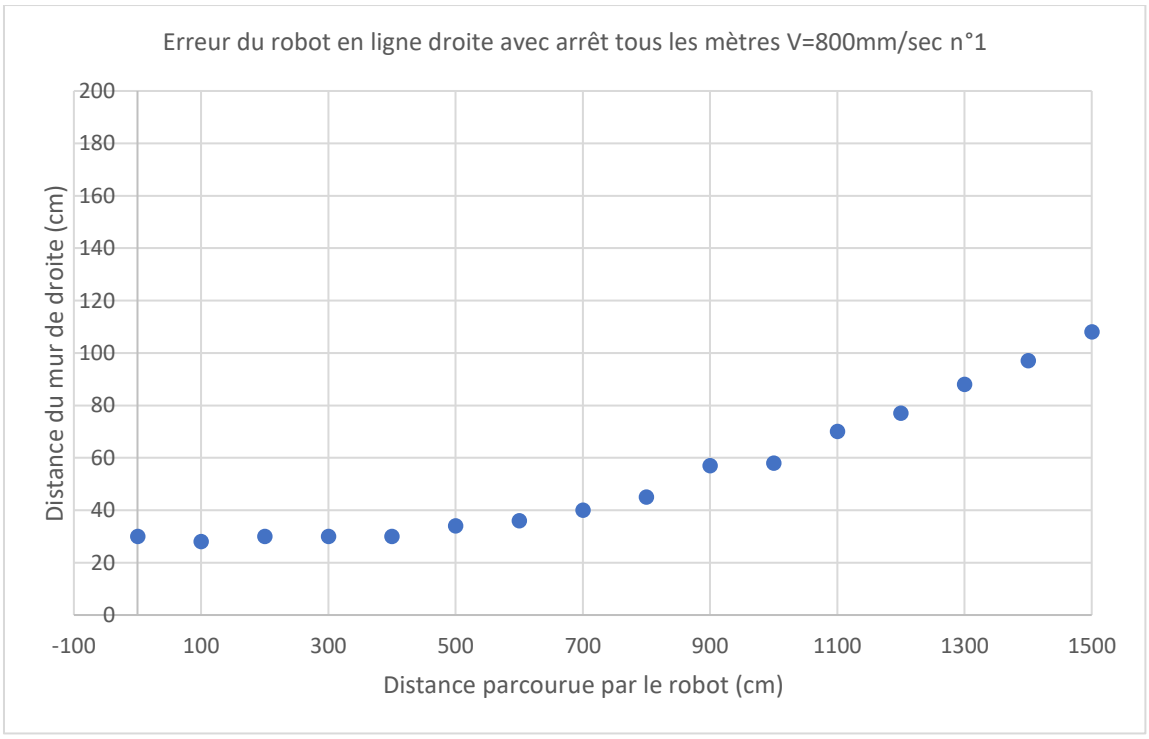
Sur tous les graphiques ci-dessous, le 0 de l'axe des ordonnées représente le mur de droite et à 200 cm se trouve le mur de gauche.

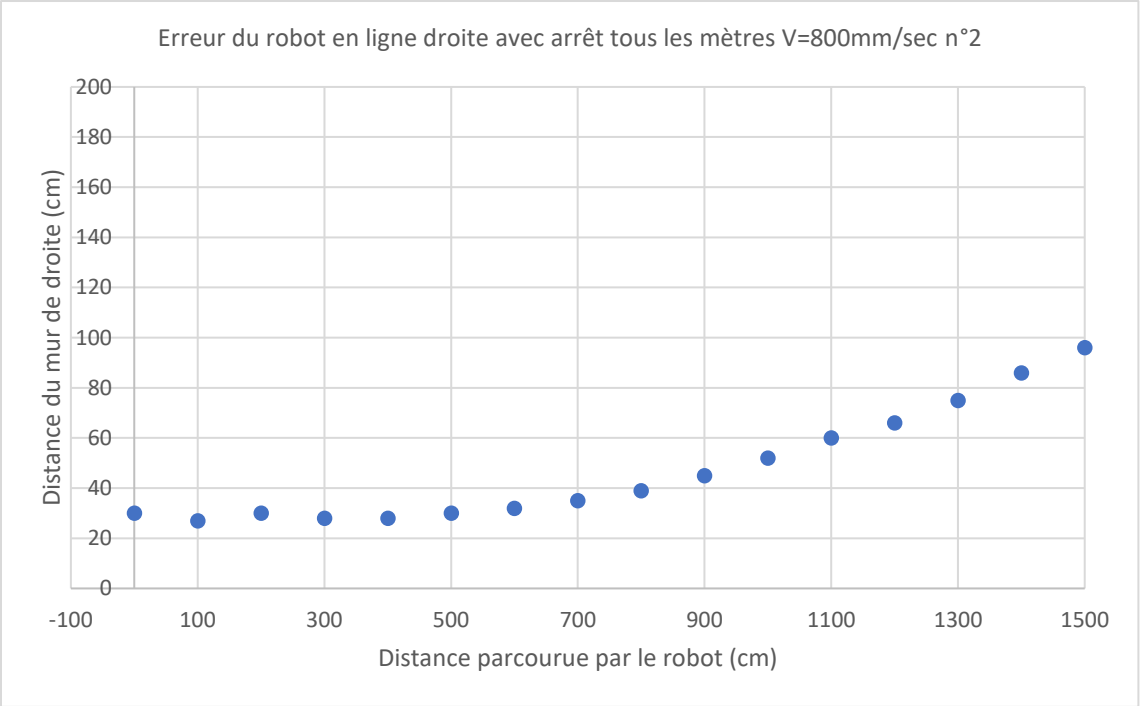


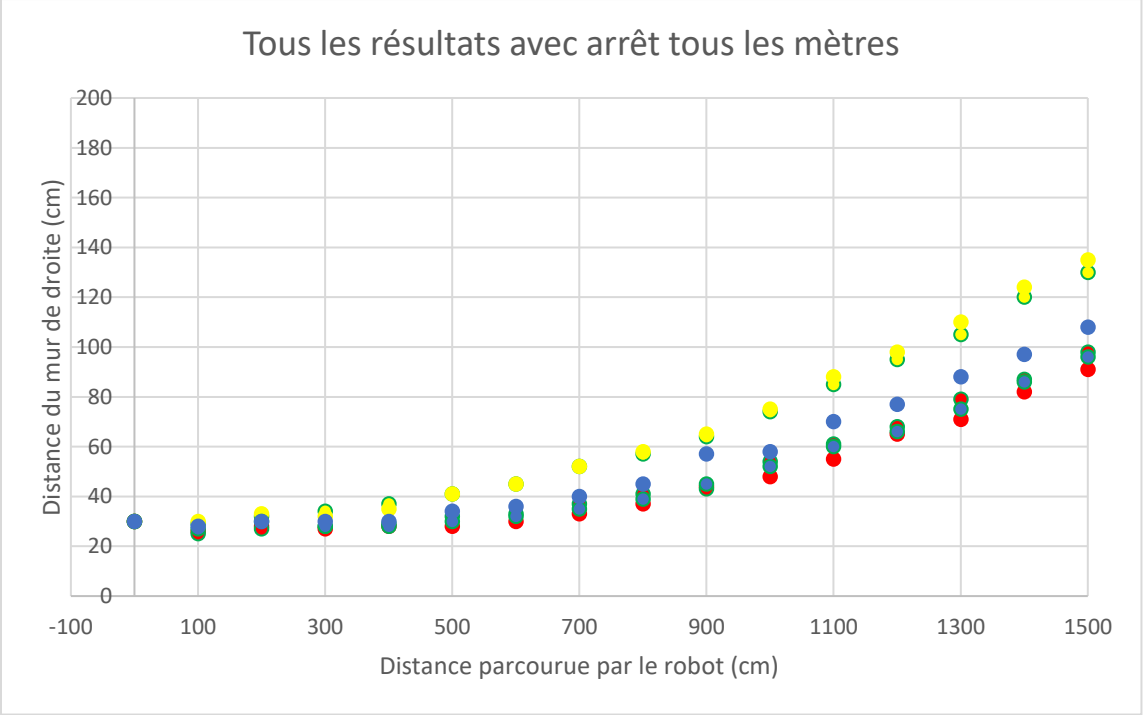

Erreur du robot en ligne droite avec arrêt tous les mètres V=400mm/sec n°2
Distance du mur de droite (cm)
200
180
160
140
120
100
80
60
40
20
0
-100
100
300
500
700
900
1100
1300
1500
Distance parcourue par le robot (cm)


Erreur du robot en ligne droite avec arrêt tous les mètres V=600mm/sec n°1
Distance du mur de droite (cm)
200
180
160
140
120
100
80
60
40
20
0
-100
100
300
500
700
900
1100
1300
1500
Distance parcourue par le robot (cm)


Erreur du robot en ligne droite avec arrêt tous les mètres V=600mm/sec n°2
Distance du mur de droite (cm)
200
180
160
140
120
100
80
60
40
20
0
-100
100
300
500
700
900
1100
1300
1500
Distance parcourue par le robot (cm)

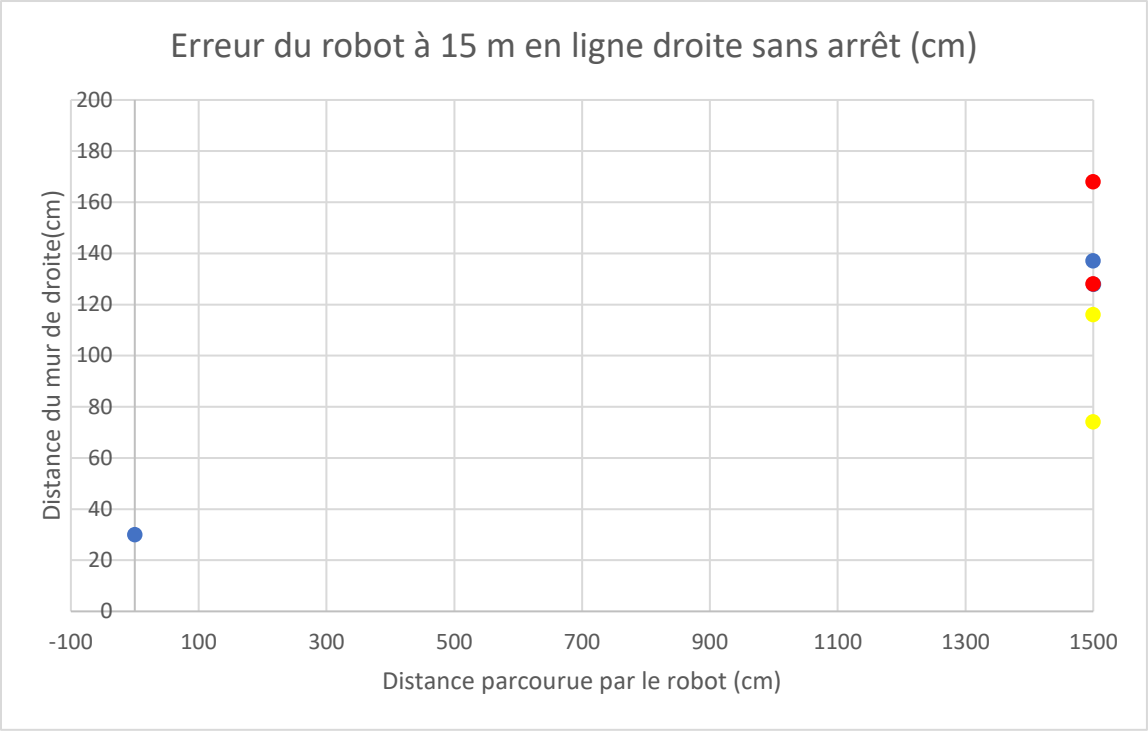

Erreur du robot en ligne droite avec arrêt tous les mètres V=800mm/sec n°1
Distance du mur de droite (cm)
200
180
160
140
120
100
80
60
40
20
0
-100
100
300
500
700
900
1100
1300
1500
Distance parcourue par le robot (cm)





Les résultats en jaune représentent les tests effectués à une vitesse de 400mm/sec, en rouge ceux fait à une vitesse de 600 mm/sec et en bleu ceux fait à une vitesse de 800mm/sec. Pour distinguer les résultats d'une même vitesse, les couleurs du contour des points ont été changés pour les tests n°2.

Comme nous pouvons le voir ci-dessus, les résultats ont tous la même façon d'évoluer même s'il y a des écarts faibles entre les résultats d'une même vitesse. Nous étions juste de penser que l'erreur est modélisable, car même s'il y a des écarts, l'erreur se ressemble à chaque essai d'une même vitesse, et même de vitesses différentes. Nous ne pouvons pas conclure sur la modification de l'erreur en fonction de la vitesse, car il n'y a pas assez de tests réalisés. Surtout que juste après nous avons testé pour chaque vitesse de faire la même distance sans arrêt. (Voir ci-dessous)

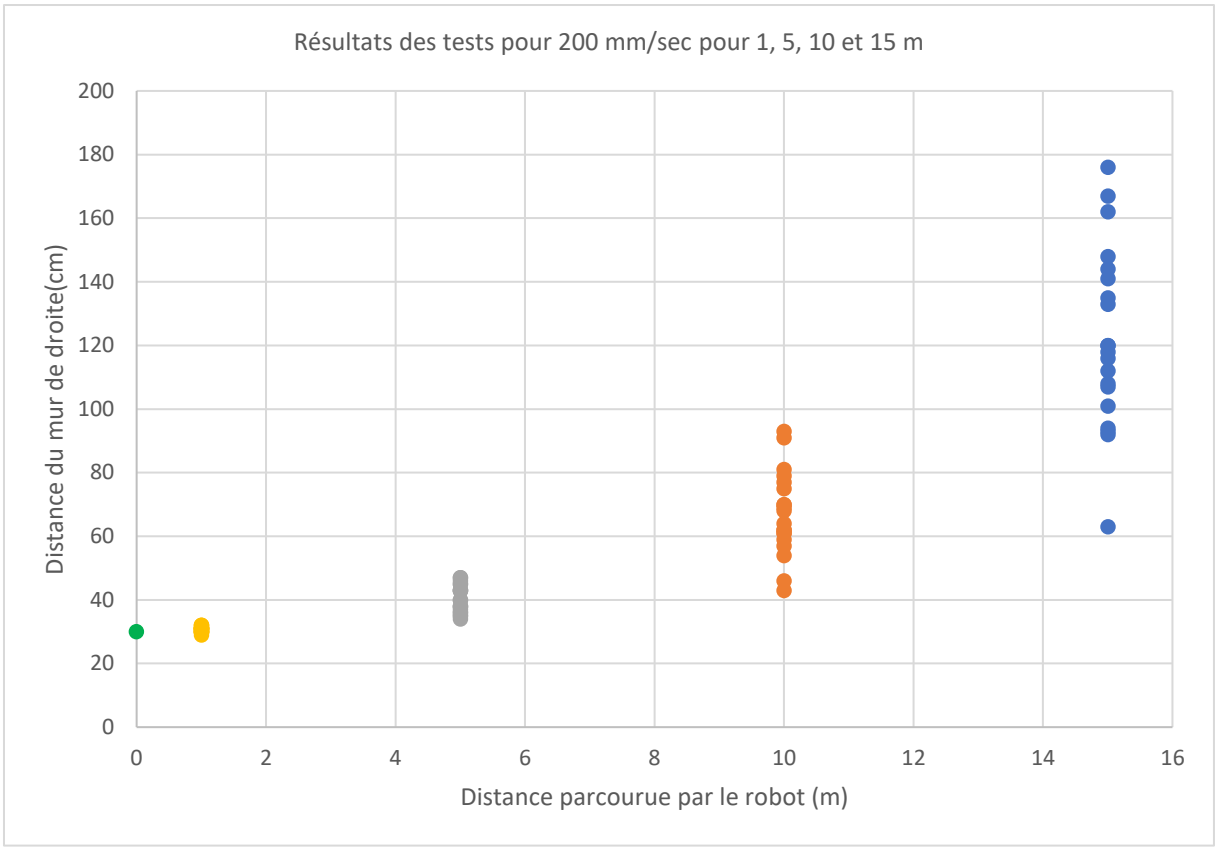


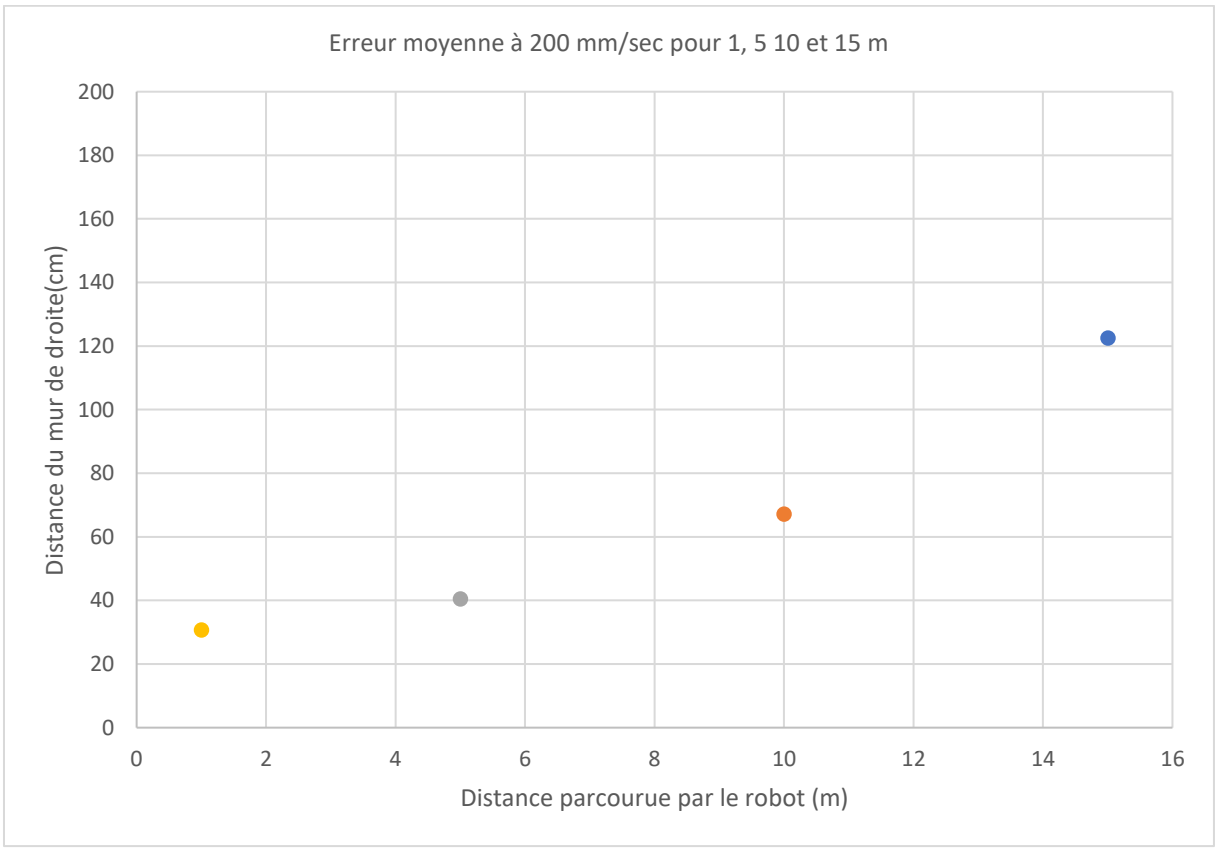
Les résultats obtenus sont complètement inversés avec ceux fait juste avant, cette différence peut venir de l'arrêt du robot, mais aussi du peu de tests effectués pour vérifier cela, nous allons réaliser une autre batterie de tests d'une façon différente afin de comparer nos résultats.

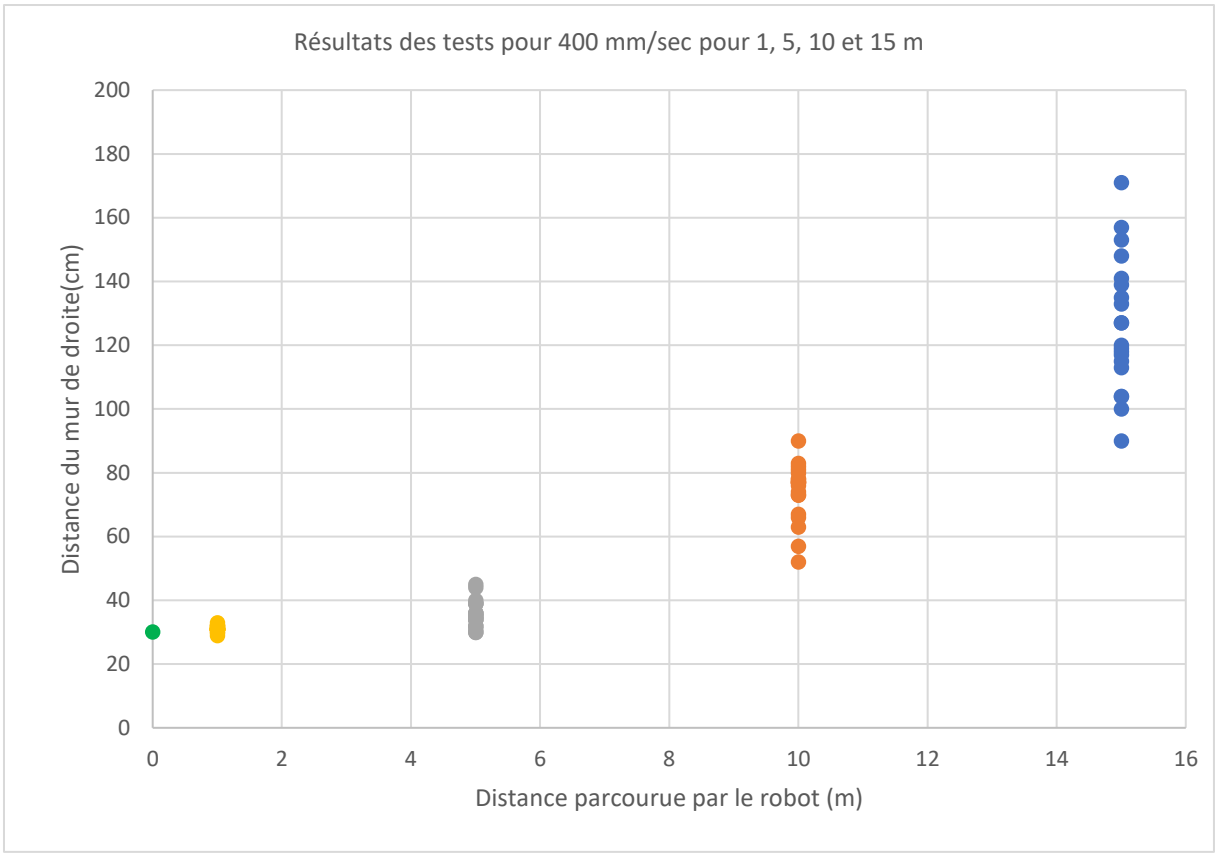
3. Test en ligne droite sans arrêt.

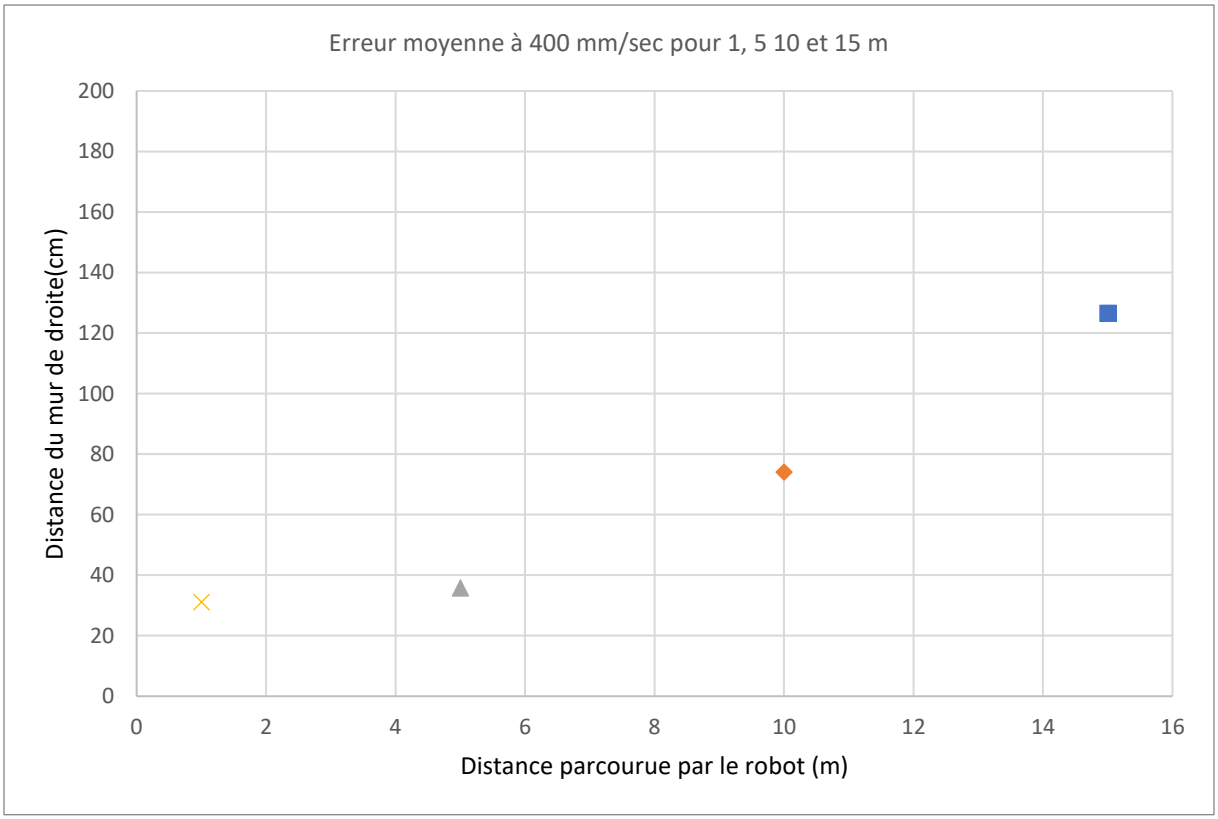
Dû à l'addition d'erreur qu'engendrent les arrêts, nous avons pensé à faire les tests d'une autre façon afin de supprimer les erreurs liées à l'immobilisation du robot. Nous allons donc réaliser la distance entière et se stopper à la distance voulue.

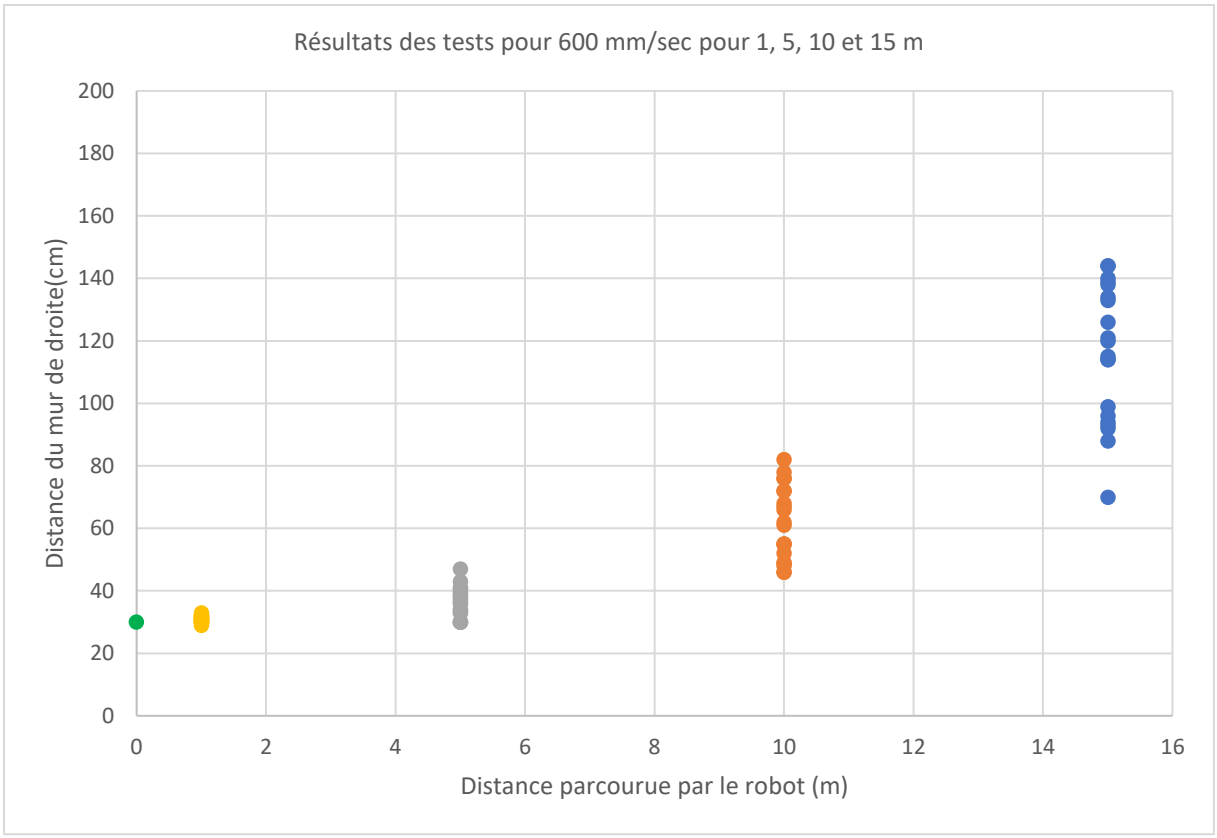
Nous avons réalisé ce type de test dans les mêmes conditions de départ que le précédent, nous avons placé le robot dans un long couloir de 2 m de large à 30 cm du mur de droite devant la salle PAC-MAN. Nous avons réalisé les tests avec 5 vitesses différentes 200, 400, 600, 800 et 1000 mm/sec et nous avons arrêté le robot à 4 distances différentes qui sont 1, 5, 10 et 15 m. Pour chaque vitesse et chaque distance, nous avons effectué 20 tests. Voici les résultats obtenus et la moyenne de nos résultats pour chaque vitesse :

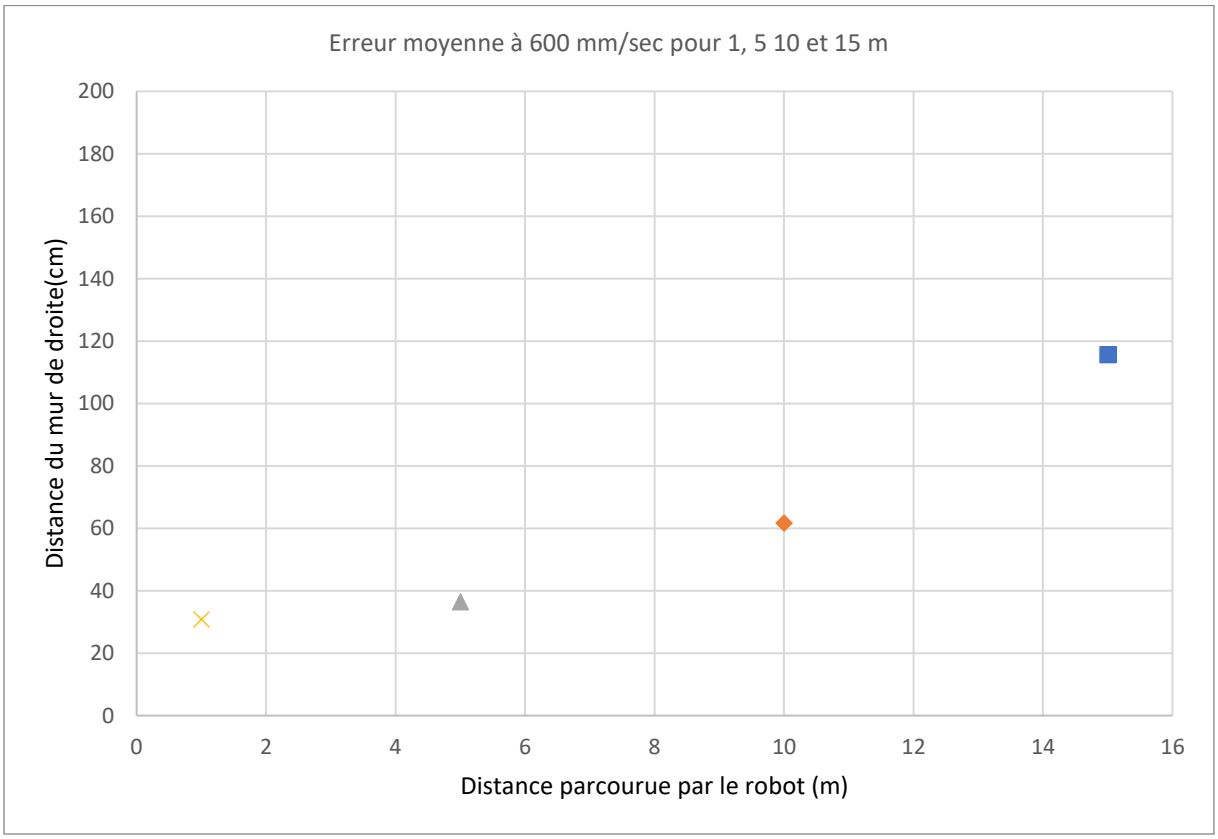

Résultats des tests pour 200 mm/sec pour 1, 5, 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16

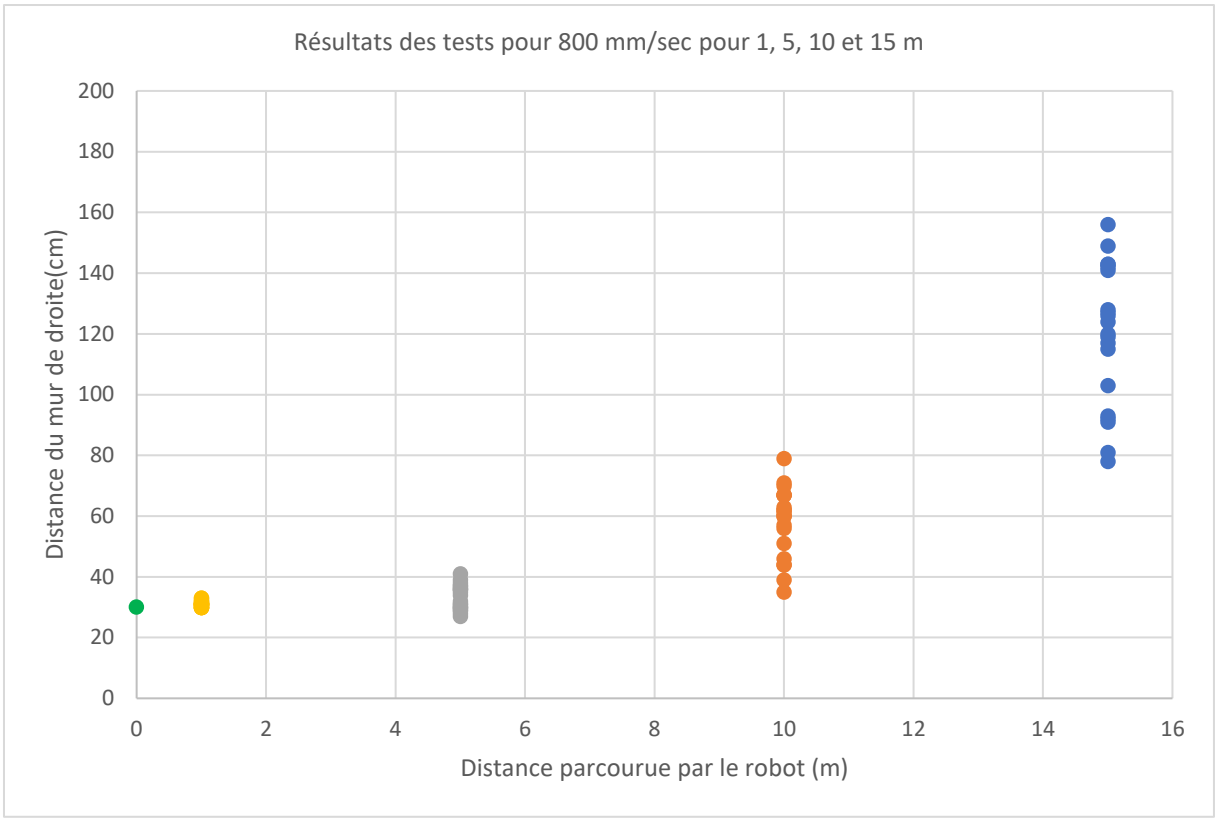

Erreur moyenne à 200 mm/sec pour 1, 5 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16

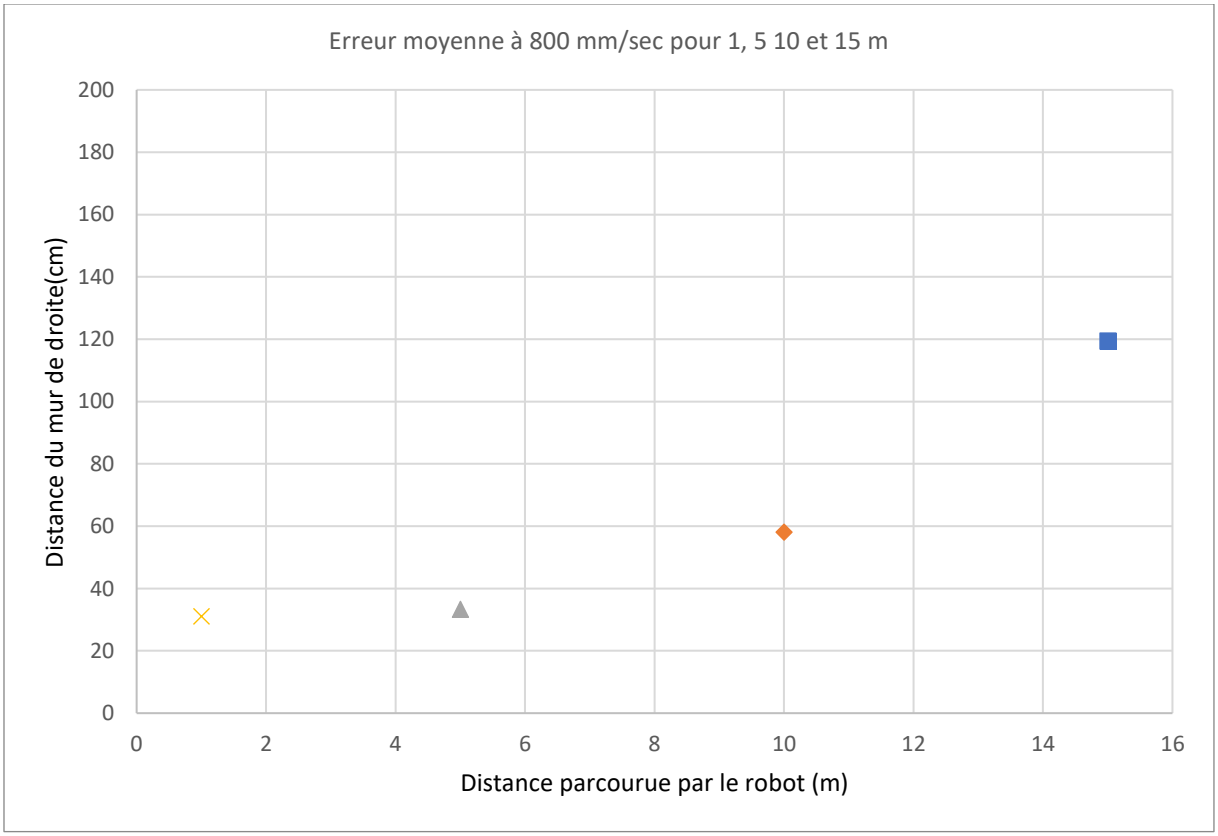

Résultats des tests pour 400 mm/sec pour 1, 5, 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16

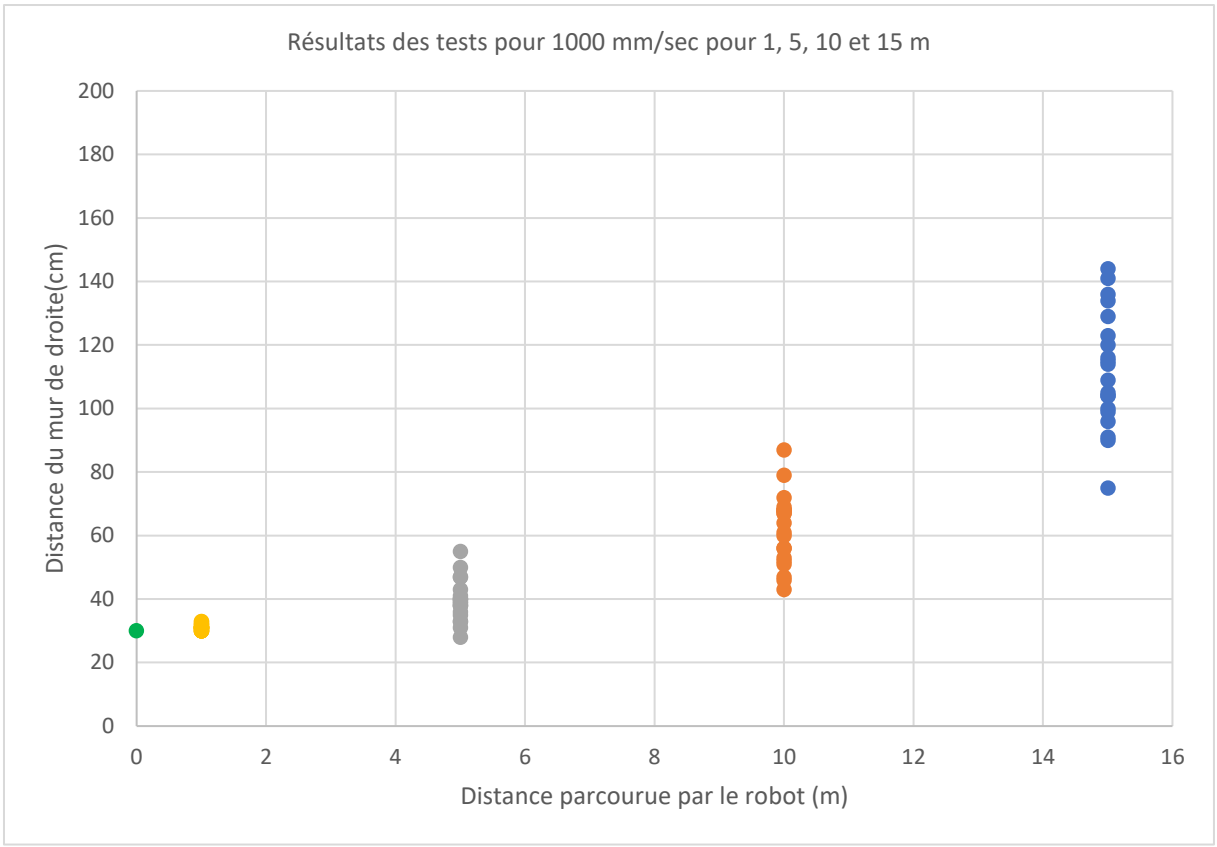

Erreur moyenne à 400 mm/sec pour 1, 5 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16

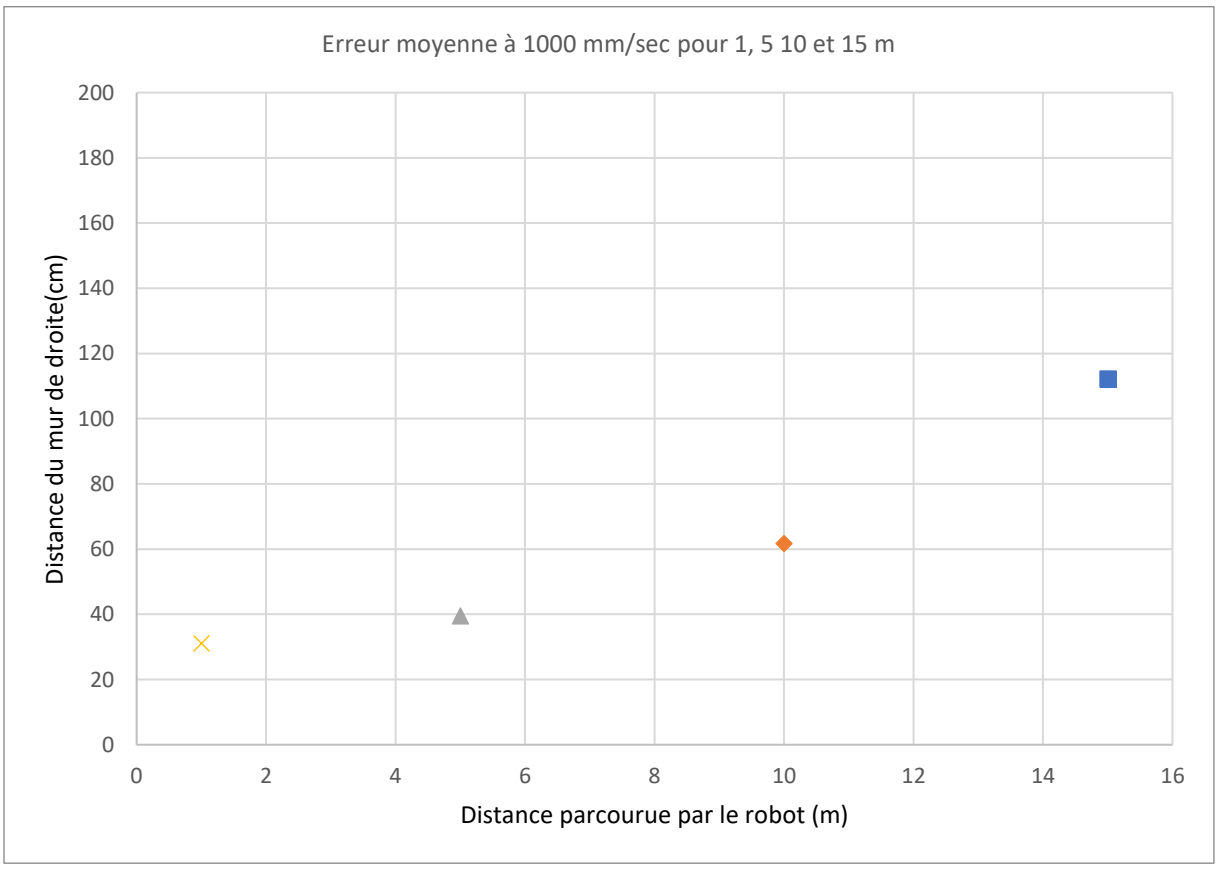

Résultats des tests pour 600 mm/sec pour 1, 5, 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16

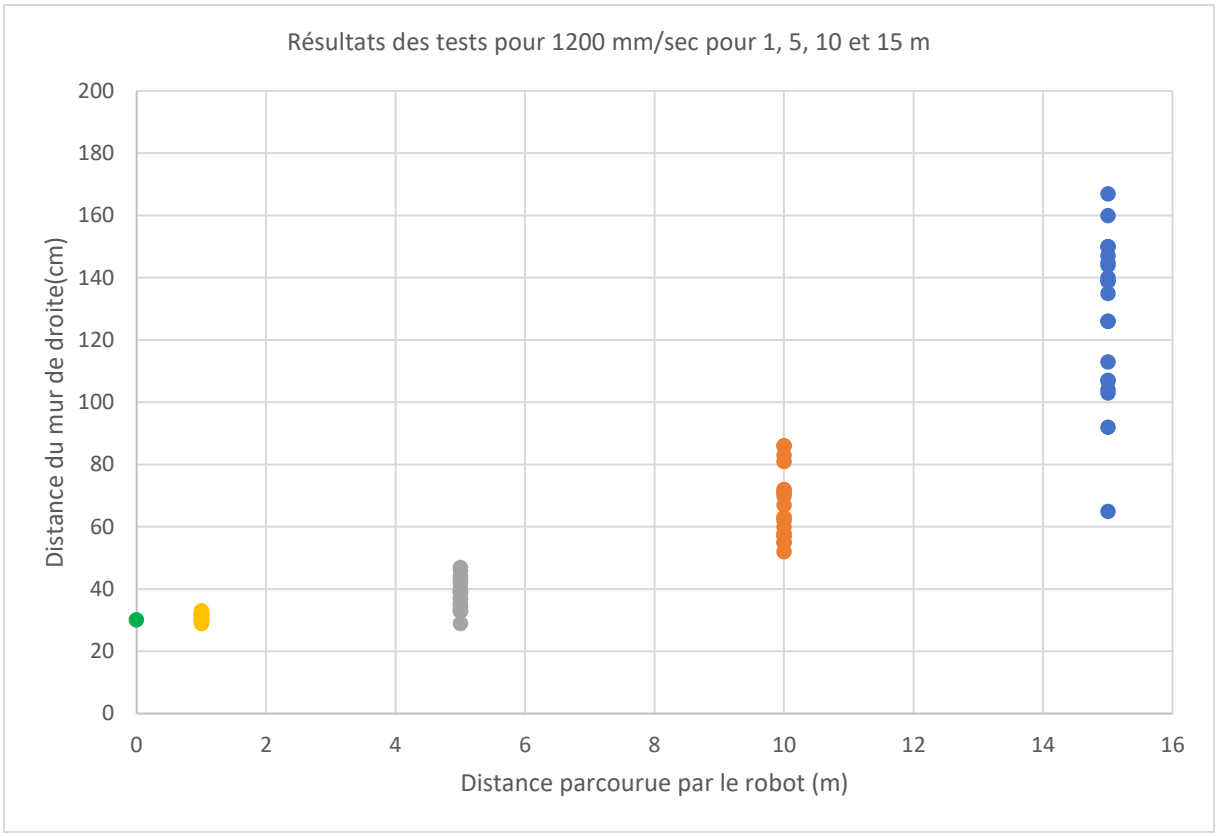

Erreur moyenne à 600 mm/sec pour 1, 5 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16

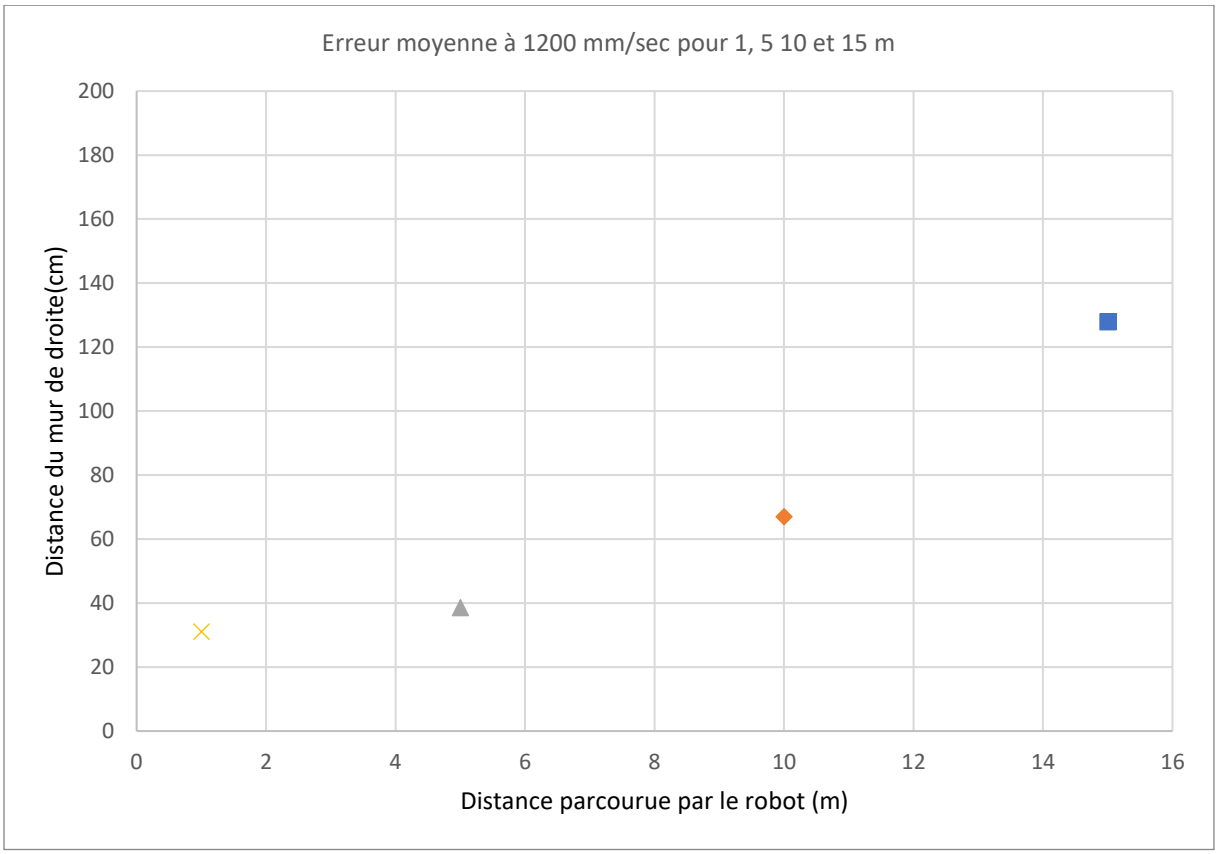

Résultats des tests pour 800 mm/sec pour 1, 5, 10 et 15 m
Distance du mur de droite(cm)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16
Distance parcourue par le robot (m)


Erreur moyenne à 800 mm/sec pour 1, 5 10 et 15 m
Distance du mur de droite(cm)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16
Distance parcourue par le robot (m)


Résultats des tests pour 1000 mm/sec pour 1, 5, 10 et 15 m
Distance du mur de droite(cm)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16
Distance parcourue par le robot (m)


Erreur moyenne à 1000 mm/sec pour 1, 5 10 et 15 m
Distance du mur de droite(cm)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16
Distance parcourue par le robot (m)

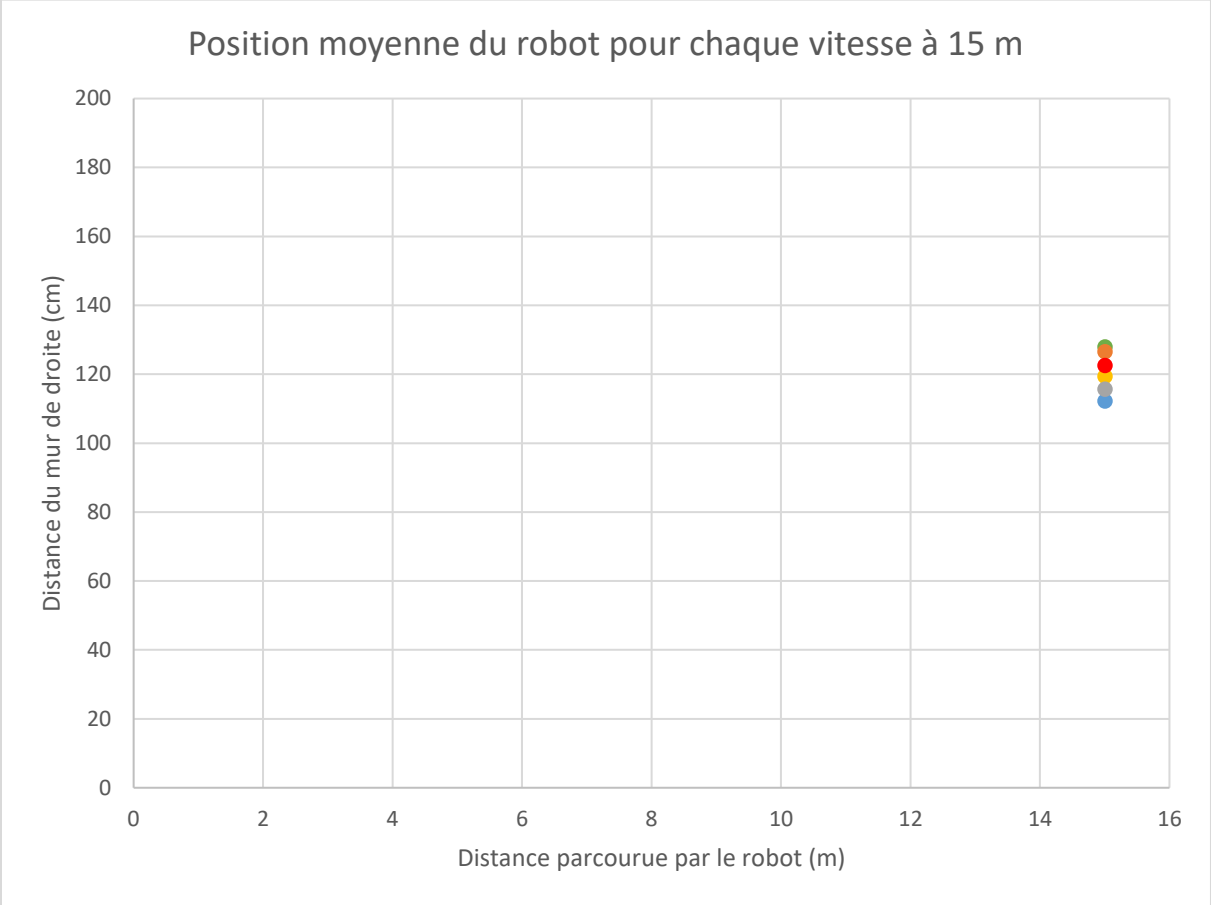

Résultats des tests pour 1200 mm/sec pour 1, 5, 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16


Erreur moyenne à 1200 mm/sec pour 1, 5 10 et 15 m
Distance du mur de droite(cm)
Distance parcourue par le robot (m)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
14
16

Ces résultats nous montrent que pour une même vitesse les résultats à 15 m peuvent avoir une différence importante avec un écart maximum entre deux résultats de 113 cm (résultats pour une vitesse de 200 mm/sec).

Cet écart est dû aux erreurs non-systématiques, nous pouvons nous apercevoir que plus le robot fait de distance plus il y a un écart important entre les résultats. Pour chaque mesure, nous pensons avoir une marge d'erreur d'environ 3 cm à cause de l'erreur humaine, du matériel, du mur qui n'est pas droit ainsi qu'aux erreurs de placement du robot au départ.

Bien que les résultats soient très dispersés pour une même vitesse, les résultats moyens de chaque vitesse sont très proches, nous avons 14 cm d'écart au maximum (au bout de 15 m) sur ces résultats moyens comme nous pouvons le voir sur le graphique ci-dessous.



Code couleur : vert= 1200mm/sec, orange= 400mm/sec, rouge= 200mm/sec, jaune= 800mm/sec, gris= 600mm/sec, bleu= 1000mm/sec.

Le robot est parti à 30 cm du mur de droite.

Nous remarquons avec ces résultats que l'erreur n'est pas due à la vitesse, car les résultats à 200 mm/sec sont meilleurs que 400 mm/sec, mais moins bon que 600 mm/sec. Nous pouvons en conclure que la vitesse n'a pas d'impact sur l'erreur du robot. Grâce à ces résultats, nous pouvons modéliser l'erreur moyenne et ainsi pouvoir récupérer la fonction de cette dérive.

3.Correcteur

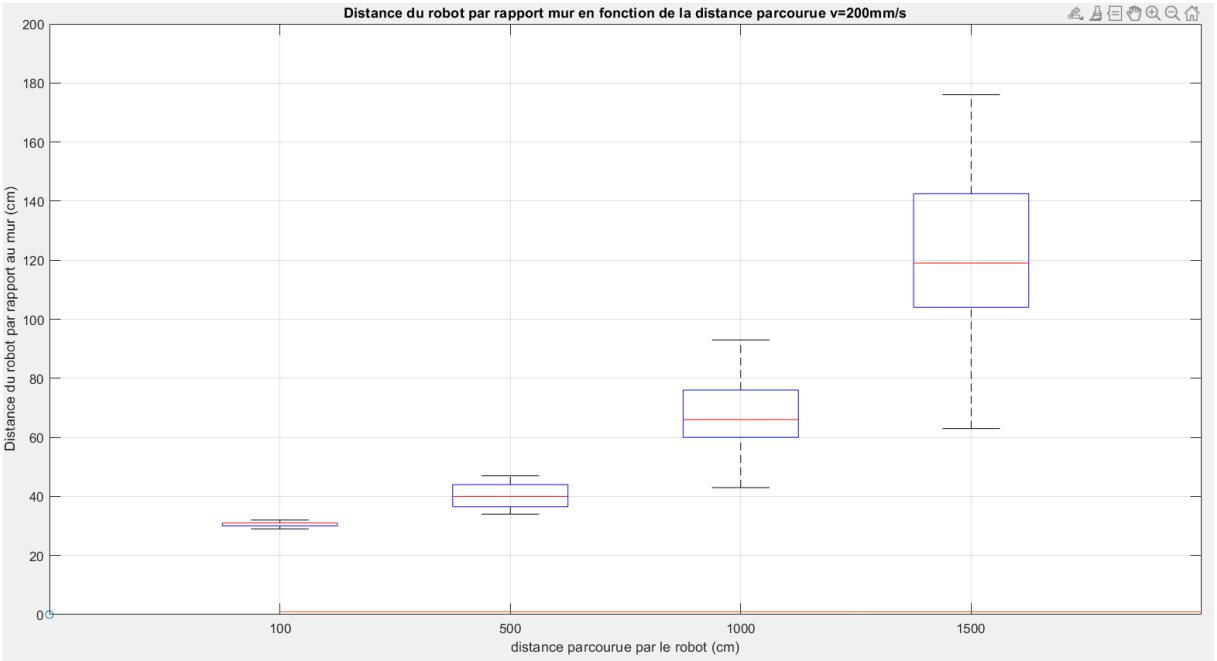
Nous allons maintenant utiliser Matlab afin d'analyser nos résultats et modéliser l'erreur grâce à la fonction nommée *Polyfit*. Nous allons, tout d'abord, avoir besoin des résultats qui sont sur le document Excel. Pour cela, nous utilisons la fonction *readmatrix* qui permet de récupérer des données qui viennent d'autres documents et qui les converties en matrice.

```
R200s5m = readmatrix('C:\Users\lucas\Documents\Resultats',...
    'Range','G76:G95',...
    'sheet',3);
```

Dans l'exemple ci-dessus, nous récupérons les données comprises dans les cellules allant de G76 à G95, de la feuille 3 soit la feuille '200|All test' du document *Résultats, ces données représentent les valeurs obtenues à 5 m de distance durant nos tests à la vitesse 200mm/sec*. Ensuite, nous allons enregistrer ces données dans une liste et les affichés avec la fonction *boxplot*. Cette fonction indique en rouge la médiane des résultats, les cotés en haut et en bas du rectangle nous donnent les percentiles 25 et 75, les traits noirs horizontaux nous donnent les valeurs maximum et minimum comme nous pouvons le voir sur le graphique ci-dessous. Cela nous permet de voir si les résultats sont contenus dans une petite zone ou alors s'ils sont très éparpillés.

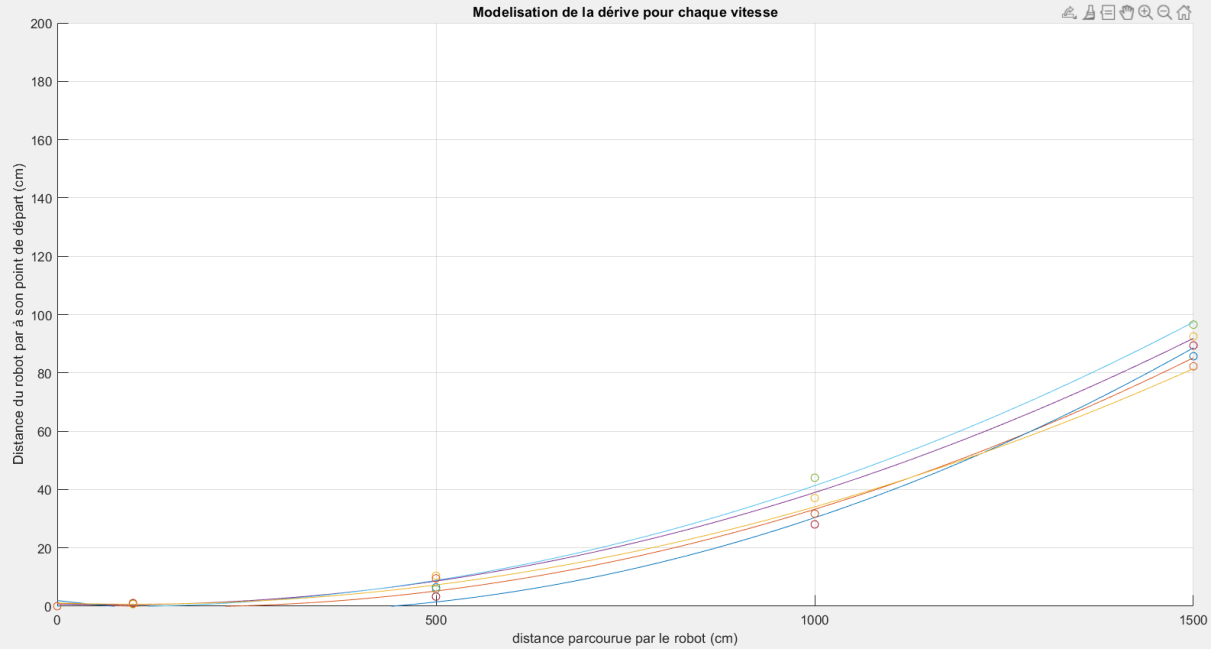
```
R200T=[R200s1m,R200s5m,R200s10m,R200s15m];
Xlab=[100,500,1000,1500];
boxplot(R200T,Xlab)
ylim([0 200])
xlim([0 5])
title('Distance du robot par rapport mur en fonction de la distance parcourue v=200mm/s')
ylabel('Distance du robot par rapport au mur (cm)')
xlabel('distance parcourue par le robot (cm)')
```

Code permettant d'afficher les résultats avec la fonction *boxplot*.



Résultats obtenus avec la fonction *boxplot*.

Nous avons dans un second temps modélisé l'erreur moyenne pour chaque vitesse comme nous pouvons le voir ci-dessous afin de vérifier que la dérive du robot se ressemble pour chaque vitesse.



Vitesse du robot pour chaque fonction de haut en bas : 1- 400 mm/sec 2- 200 mm/sec 3- 800 mm/sec 4- 600 mm/sec 5- 1000mm/sec

Pour créer ces courbes, nous avons utilisé les fonctions *polyfit* et *polyval,* la première permet de récupérer les coefficients d'un polynôme, la deuxième permet de donner les valeurs du polynôme à chaque instant et nous permet de créer la courbe. Le polynôme est de type : $p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$.

Donc, pour chaque courbe du graphique ci-dessus, nous avons fait les étapes suivantes :

- Tout d'abord, nous avons pris la moyenne des résultats obtenus aux distances : 1, 5, 10 et 15 mètres.

```
MoyR200=[30,mean(R200T)];
```

- La valeur 30 représentant la distance du mur au départ du robot. Nous enlevons ensuite 30 à toutes les valeurs afin que le point de départ du robot se trouve à 0. Cela a une influence sur le polynôme, si nous ne le faisons pas cela fait apparaître un biais qui une fois implanté dans le code du robot va nous poser des problèmes.

```
MoyR200=MoyR200-30;
```

- Ensuite, nous définissons comme ci-dessous, les distances auxquelles nous avons pris nos mesures. Puis nous utilisons la fonction polyfit.

```
Xlab=[0,100,500,1000,1500];
p200=polyfit(Xlab,MoyR200,2);
```

Variable de type P(Xlab)=

Données liées à Xlab

Degrés du polynôme

- Enfin, nous définissons x2 qui est un intervalle compris entre 0 et 1500 avec un pas de 0.1. La fonction polyval va nous donner une valeur du polynôme à chaque pas dans cet intervalle.

```
x2 = 0:.1:1500;
y4=polyval(p200,x2);
```

- Il ne nous reste plus qu'à afficher le résultat :

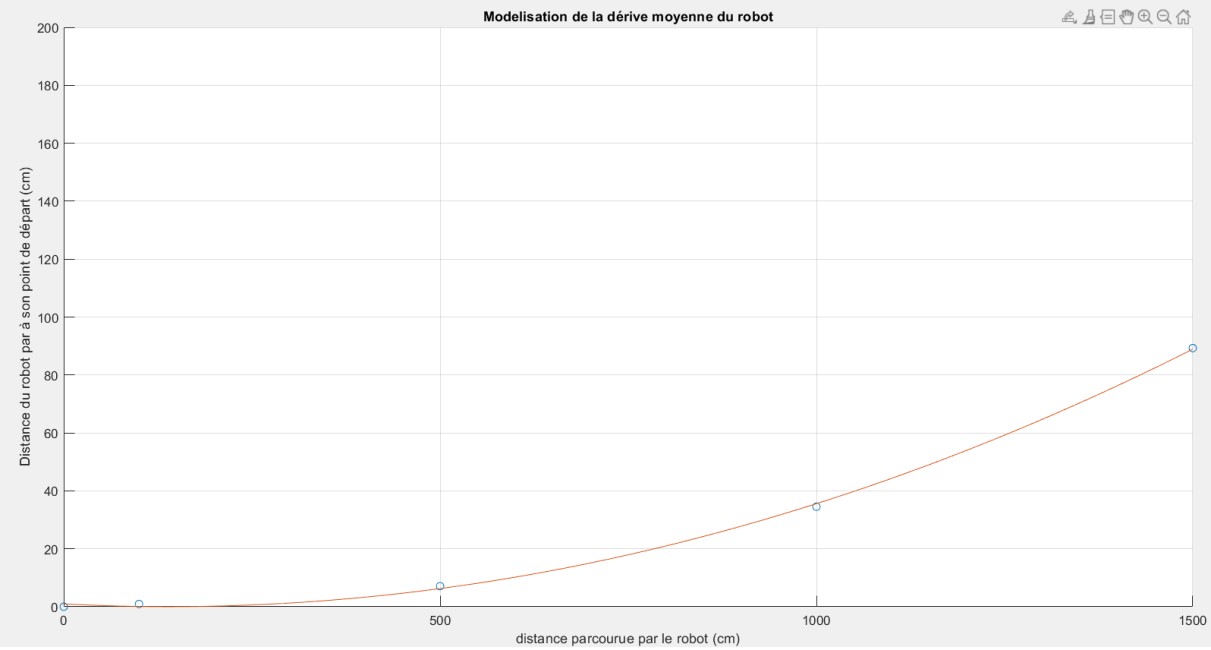
```
plot(Xlab,MoyR200,'o',x2,y4)
```

Nous allons maintenant appliquer ces étapes à une moyenne de tous les résultats obtenus à chaque distance sans se soucier de la vitesse, car comme préciser dans la partie précédente, la vitesse n'influe pas ou très peu sur la dérive du robot.

Pour cela, il suffit de refaire les étapes établies juste avant, la seule différence est au départ, car il faut récupérer les valeurs de chaque vitesse.

```
Tvit=[R200T;R400T;R600T;R800T;R1000T];
MoyAll=[30,mean(Tvit)];
```

Après avoir effectué toutes étapes voici les résultats obtenus :



Courbe représentant la dérive moyenne du robot, les cercles bleus représentent la valeur moyenne calculée à cette distance.

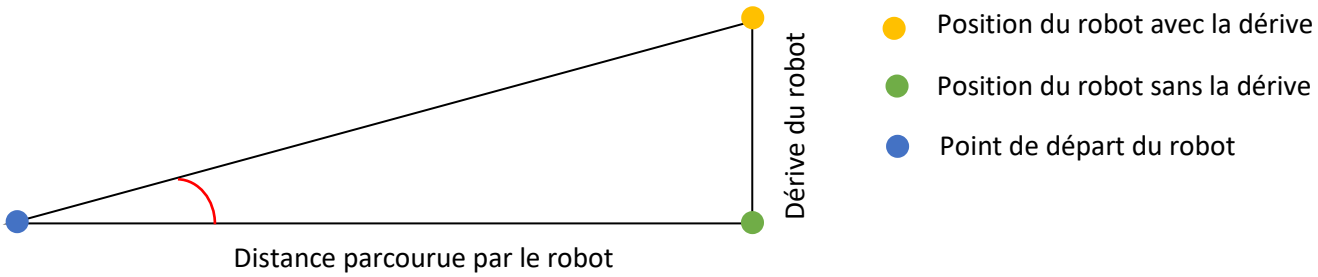
On peut voir que la courbe obtenue est bonne même si elle ne passe pas exactement par chaque cercle bleu, elle semble bien définir la dérive moyenne du robot. Nous n'avons plus qu'à récupérer les coefficients de notre polynôme stocker dans la variable *pall1* afin de l'utiliser dans notre code.

Notre polynôme sera de la forme suivante : $p(x) = a * x^2 + b * x + c \; avec \; p1 = 4.8009 * 10^{-5}$, $p2 = -0.0134 \; et \; p3 = 0.9805$ avec a, b et c qui sont des coefficients.

Notre dérive est donc représentée par la fonction :

$$p(x) = 4.8009 * 10^{-5} * x^2 + -0.0134 * x + 0.9805$$

Ces coefficients permettent d'avoir la position du robot en fonction de la distance qu'il a parcouru. Mais nous devons trouver une façon d'implémenter l'opposé de cette fonction afin d'enlever l'erreur. Nous avons choisi d'influencer l'angle du robot à partir de ce polynôme, en effet à chaque instant, nous pouvons récupérer la distance parcourue par le robot et sa dérive ce qui nous permet de former un triangle rectangle comme ceci :



L'angle que nous voulons récupérer est représenté en rouge sur la figure ci-dessus, pour trouver sa valeur, il faut appliquer les fonctions de trigonométrie. Avec les valeurs que nous avons, nous allons récupérer la tangente définit par le $\frac{coté \; opposé}{coté \; adjacent}$ dans un triangle rectangle. Seulement, la tangente d'une fonction est définie par $y_{tan} = f'(x_{abs})(x_{tan} - x_{abs}) + f(x_{abs})$ avec $x_{abs} = abscisse \; de \; f$ et $x_{tan} = abscisse \; de \; la \; tangente$. C'est donc ce calcul que nous allons utiliser.

$$\text{Avec } f(x) = a{x_{abs}}^2 + bx_{abs} + c$$

$$\text{et } f'(x_{abs}) = 2ax_{abs} + b$$

$$\text{Donc } y_{tan} = (2a + b)(x_{tan} - x_{abs}) + {ax_{abs}}^2 + bx_{abs} + c.$$

La valeur de x_{tan} nous est encore inconnue, nous cherchons sa valeur pour $y_{tan} = 0$

$$(2ax_{abs} + b)xtan - (2ax_{abs} + b)x_{abs} + {ax_{abs}}^2 + bx_{abs} + c = 0$$

$$(2ax_{abs} + b)xtan = (2ax_{abs} + b)x_{abs} - ({ax_{abs}}^2 + bx_{abs} + c)$$

$$(2ax_{abs} + b)xtan = {ax_{abs}}^2 - c$$

$$xtan = \frac{{ax_{abs}}^2 - c}{2ax_{abs} + b}$$

Maintenant que nous connaissons toutes nos variables, nous allons coder notre correcteur. Nous allons d'abord initier les coefficients :

```
36  float Yr,X,Xtan,thetar,thetarb=0,thetaresult=0;
37  float a=4.8009*pow(10,-5);//4.8009
38  float b=-0.0134;//-0.0134
39  float c=0.9805;//0.9805
```

Après avoir codé la connexion avec le robot nous allons récupérer la distance qu'il a parcouru avec la fonction *robot.getX()* que nous allons diviser par 10 pour obtenir des centimètres afin d'avoir la même grandeur qu'utiliser dans le calcul du polynôme. Ce polynôme est enregistré dans la variable Yr puis nous initions $xtan$.

```
97  X=robot.getX()/10;      //Dist
98  Yr=a*pow(X,2)+b*X+c;
99  Xtan=(a*pow(X,2)-c)/(2*a*X+b);
```

A cette étape, nous nous sommes rendu compte de 3 problématiques :

- Premièrement, si nous effectuons le calcul de l'angle avec seulement Xtan comme nous voulions le faire au départ, nous allons commettre une erreur en effet nous aurions calculé l'angle au point Xtan, or le calcul de l'angle avec la tangente se fait au croisement de l'abscisse par la droite.
- En second, aucune fonction n'existe sur ce robot pour donner un angle, la seule possibilité est de donner une vitesse angulaire.
- La fonction permettant de donner une vitesse angulaire ne prend pas des valeurs plus faibles que 1. Etant donné que nous actualisons nos valeurs très fréquemment, la vitesse que nous allons donner sera inférieur à 1.

Pour le premier point, afin de calculer l'angle nommé thetar de la façon décrite juste avant nous effectuons le calcul suivant :

```
179  if(X>=10){
180  thetar=atan(Yr/(X-Xtan))*180/PI;
181  }
182  else{
183  thetar=0;
184  }
```

En C++, la fonction atan nous donne un angle en radian, le fait de multiplier par 180 et diviser par Pi permet d'avoir l'angle en degrés. Le if X>=10 nous permet de ne pas avoir une division par 0 et/ou d'une valeur proche de 0. Si la valeur est plus faible que 10 mm, nous mettons la vitesse de rotation à 0. La dérive étant très faible au départ, le robot va facilement rattraper son retard.

Pour la seconde problématique, étant donné que nous avons une vitesse angulaire et pas un angle, nous allons récupérer l'angle précédent et faire la différence entre l'angle calculé à l'instant présent et l'angle précédent et le stocker le résultat dans thetaresult :

```
thetaresult=thetar-thetarb;
```

La solution mise en place pour la troisième problématique va nous servir pour régler des problèmes possibles avec cette solution. Elle va permettre de changer la vitesse angulaire en cas de retard ou d'avance :

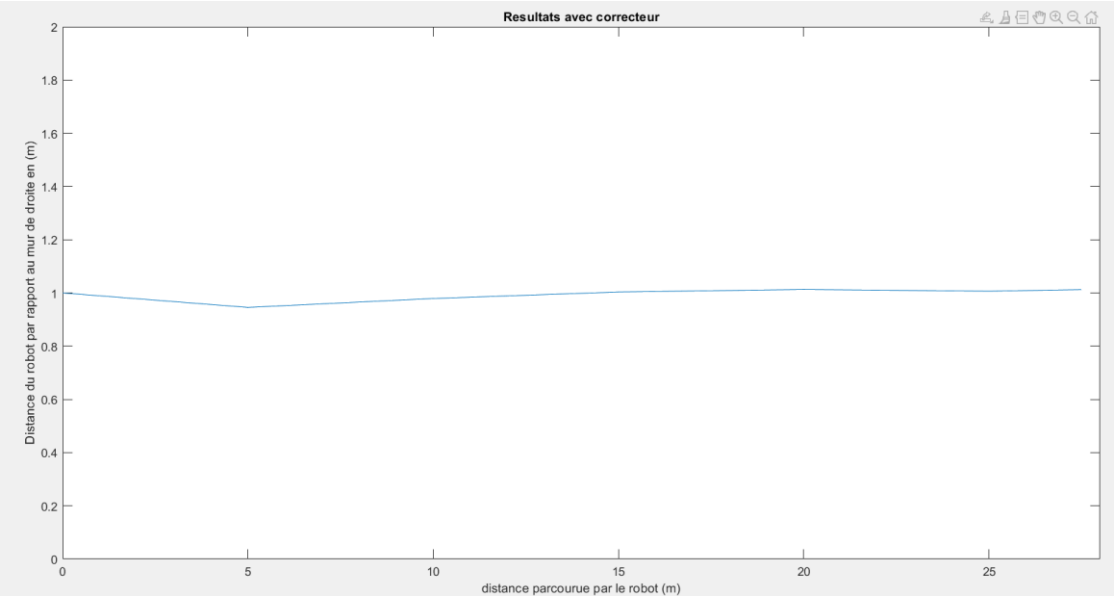
```
if(abs(robot.getTh())>abs(thetar)){
robot.setRotVel(0);
}
else if((abs(thetar)-abs(robot.getTh()))>0.2){
robot.setRotVel(-1);
}
else{
robot.setRotVel(-(thetaresult));
}
```

Si l'angle du robot est plus grand que celui calculé, la vitesse angulaire est mise à 0. Si l'angle du robot est plus petit que celui de calculé alors nous mettons à -1 la vitesse de rotation. Sinon nous attribuons -thetaresult qui est l'opposé de la valeur calculé thetaresult. Pourquoi mettre l'opposé ? Il ne faut pas oublier que la valeur contenue dans thetar que nous avons utilisée pour calculer thetaresult est l'angle venant de la dérive du robot, or nous voulons son opposé pour trouver le correcteur. Mettre une valeur négative dans la fonction setRotVel le fait tourner à droite.

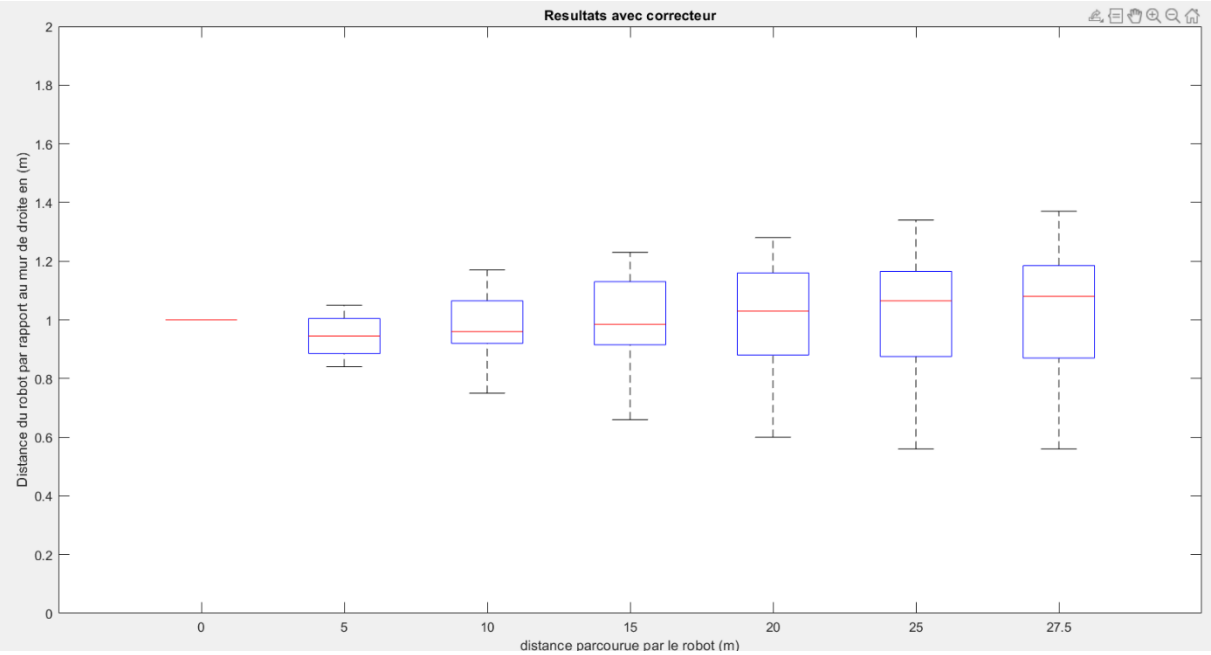
Nous avons testé dans un premier cas notre correcteur sur une ligne droite sans arrêt. Le test a été concluant, nous avons réussi à faire rouler le robot sur une longueur d'un peu moins de 30 mètres et le robot semblait suivre une ligne droite. Pour en être sûr, nous allons effectuer 20 tests en ligne droite en arrêtant le robot à 5, 10, 15, 20,25 et 27,5 mètres.

Les résultats ont été mis dans la feuille Excel nommé *Résultats avec correcteur*, nous avons ensuite affiché la fonction représentant la moyenne de nos résultats avec Matlab.

Résultats :



Nous pouvons voir sur le graphique ci-dessus, qu'en moyenne durant les 5 premiers mètres le robot va un peu sur la droite puis il dévie petit à petit vers la gauche en restant dans la zone des 1 m. Les tests ayant été faits dans un couloir de 30 m de longueur, nous ne savons pas comment se comporte le robot après. Nous avons encore une fois utilisé la fonction boxplot pour voir afficher plus d'informations :



Nous pouvons voir que plus nous nous éloignons, plus grande est la zone où le robot peut se situer. À 27,5 m, dans 50 % des cas (valeurs comprises entre le percentile 25 et 75), le milieu robot se situe entre 1,18 m et 0,87 m du mur de droite. Le résultat médian étant de 1,08 m.

Ces résultats sont très bons étant donné que le robot fait 38 cm de long, cela veut dire qu'après 27,5 m de distance parcourue, dans 50% des cas, le point voulu est sous le robot. Dans les autres cas, la position du robot a dévié de 44 cm vers la droite ou 37 cm vers la gauche (voir schéma ci-dessus). La valeur de la dérive a été prise au milieu du robot. Soit une dérive d'environ 1 robot.

L'amélioration obtenue grâce au correcteur est très importante, avant d'avoir été mis en place quand le robot roulait en ligne droite sur une distance de 15 m, nous étions obligés de le mettre le plus proche du mur de droite, car il dérivait de 90 cm sur la gauche. Entre 15 et 20 m, le robot prenait quasi tout le temps le mur si nous ne l'arrêtions pas. Maintenant, à 15 m, le robot à une dérive maximum de 34 cm vers la droite et 23 cm vers la gauche avec une position moyenne à 1 m. Nous allons donc pouvoir réaliser notre fingerprint avec une bonne précision car le robot est tout le temps au-dessus du point voulu même si des améliorations sont possibles car le milieu du robot n'est pas exactement au-dessus du point.

Dans les améliorations possibles, nous pouvons citer l'utilisation de thread. En effet, l'actualisation des données se fait à chaque passage dans la boucle, mais les threads permettraient de faire plusieurs taches en même temps et nous permettraient d'actualiser à chaque instant. L'utilisation d'un filtre de Kalman avec des capteurs comme un gyroscope et un accéléromètre permettrait aussi d'améliorer notre système.

5. Prise en main du matériel RFID

La technologie que nous voulons tester à l'aide de notre fingerprint s'appelle RFID. La RFID est une technologie permettant de communiquer des informations entre un émetteur et un récepteur. Cette technologie est utilisée dans différents systèmes comme les passeports, les puces d'animaux de compagnie ou encore les transpondeurs pour le péage. La puissance du signal changeant en fonction de la distance, nous voulons donc tester si ce changement est assez grand et assez précis pour pouvoir localiser avec exactitude notre position.

1. Le matériel

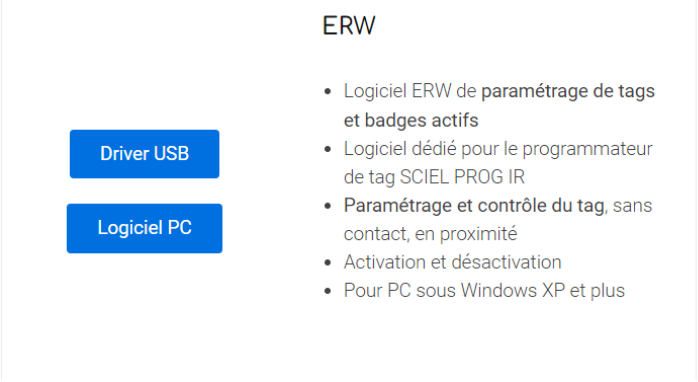
Au niveau matériel, nous aurons besoin de plusieurs balises RFID, d'un programmateur de tag et d'un lecteur RFID. Ces balises enverront des informations au récepteur, nous pourrons ainsi récupérer des données comme son ID, la puissance du signal, la date à laquelle à été envoyé le signal.

Lecteur RFID

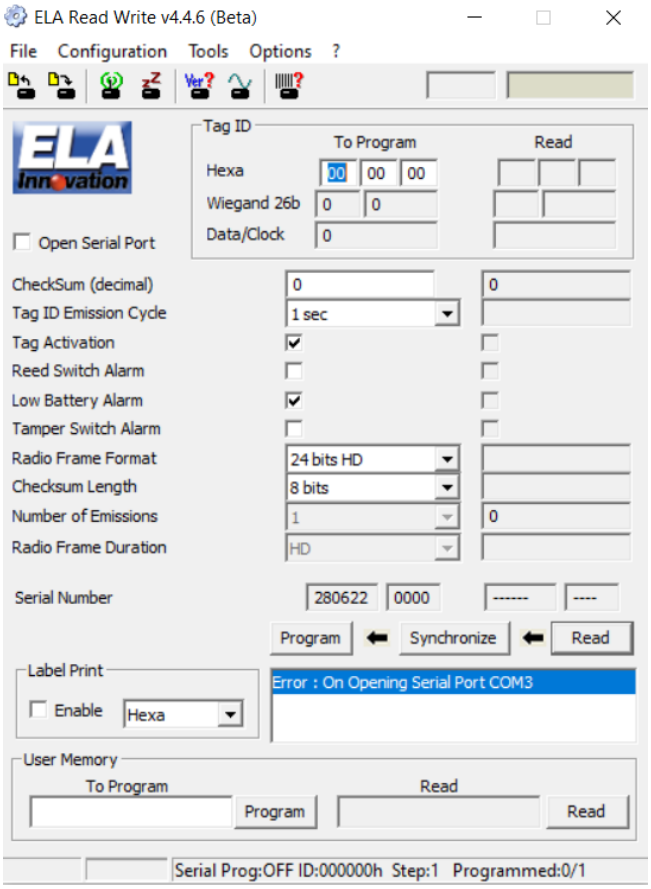
Balise RFID

2. Les logiciels

Nous devons configurer les balises avant de commencer à récupérer les données. Pour cela, nous allons utiliser un programmateur de tag comme celui-ci *https://elainnovation.com/wp-content/uploads/2021/09/FP-SCIEL-PROG-IR-02C-EN-1.pdf* et un logiciel nommé ERW que vous pouvez récupérer sur le site suivant *https://elainnovation.com/suite-logiciels-rfid/*. Il nous suffit de cliquer sur le bouton *logiciel PC* de la partie ERW comme ci-dessous :



Il n'y a plus qu'à brancher le programmateur sur un port USB et mettre une balise au-dessus de celui-ci et depuis le logiciel que nous pouvons voir ci-dessous, nous allons pouvoir allumer et éteindre les balises, modifier leurs ID, la fréquence d'envoie des informations, etc…



Les balises ont normalement déjà toutes été configurées, il nous suffit juste de les activer. Pour le lecteur, nous avons juste à le brancher à votre ordinateur et à une alimentation.

Maintenant, nous pouvons depuis notre ordinateur, lancer un programme comme *Infinite acquisitions (version 1),* une fois lancé, une interface va apparaître où nous allons pouvoir régler des paramètres comme la position initiale du robot, la position de la balise, le nombre d'acquisitions, etc... En appuyant sur le bouton *Measure* l'acquisition des données commence. Nous pouvons récupérer les données obtenues dans le fichier *Mesures.xml,* dans lequel nous retrouvons les paramètres que nous avons initialisés, l'ID de la balise, la date et la puissance du signal mesuré par chacune des antennes du lecteur.

```
<Measure numero="1" x="0" y="0" teta="0" repere="0" Present_X="0" Present_Y="0" id="000022" date="03/03/2022 10:12:58:257" value="132/186">
</Measure>
```

Pour recommencer un test, nous devons enlever les données incluses dans le fichier *Mesures.xml.* Pour cela, il faut enlever toutes les lignes du fichier sauf les deux premières et la dernière comme ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<Mesures>
</Mesures>
```

Maintenant que nous savons comment fonctionne le matériel RFID, nous pensions effectuer le fingerprint, mais nous avions oublié un détail.

6.Synchronisation des programmes

Pour effectuer le fingerprint nous devons accrocher une ou plusieurs balises sur les murs, mettre le lecteur sur le robot. Ensuite, nous avons besoin de lancer nos programmes permettant de récupérer les données à l'aide du lecteur RFID et lancer le programme permettant de faire avancer le robot mais nous avons un problème. En effet, nous voulons que la prise de données se fasse seulement quand le robot est arrêté. Les programmes n'étant pas sur la même machine. D'un côté le programme permettant la récupération des données se trouvant sur l'ordinateur (Windows) et est codé en langage C#. De l'autre, le programme contrôlant le robot est en C++, utilise ROS et ne peut être lancer que sous une machine Linux. Il a fallu trouver une façon de synchroniser les programmes.

L'idée que nous avons trouvée est d'utiliser les sockets afin de communiquer entre les deux programmes, la machine virtuelle utilisée étant sur le même ordinateur que la machine sous Windows, nous pouvons alors communiquer entre les deux. Notre but est que le client qui sera le programme pour le RFID envoie un message au serveur qui est le code contrôlant le robot. Le programme RFID va récupérer les données à la position initiale, puis, dès qu'il a fini, il va envoyer un message, à l'arrivée de celui-ci, le robot se déplace jusqu'au point suivant et renvoie une donnée au client qui effectue de nouveau la récupération de données et ainsi de suite.

Pour la partie serveur, le code est nommé *Boucle1,* nous avons dans un premier temps à inclure les librairies dont nous avons besoin :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <string.h>
#include <string>
using namespace std;
```

Ensuite, nous créons le socket et nous initions différentes valeurs comme le port ou l'adresse à utiliser pour ce socket :

```
102   //Initialisation Socket
103   cout<<"en attente d'une connexion client"<<endl;
104     int listening = socket(AF_INET,SOCK_STREAM,0);
105   if(listening==-1){
106   cerr<<"cant create socket";
107   return -1;
108   }
109
110
111   sockaddr_in hint;
112   hint.sin_family=AF_INET;
113   hint.sin_port = htons(8000);
114   inet_pton(AF_INET,"0.0.0.0", &hint.sin_addr);
115
116   if(bind(listening, (sockaddr*)&hint, sizeof(hint))==-1)
117   {
118   cerr<<"cant bind to IP/port";
119   return -2;
120   }
```

La ligne 114 permet d'accepter la connexion de toute adresse IP se connectant au socket. Après ça, nous attendons qu'un client se connecte, nous validons ensuite sa connexion et nous récupérons certaines de ses informations :

```
122  //Mark the socket for listening in
123  if(listen(listening, SOMAXCONN)==-1){
124    cerr<<"cant listen";
125    return -3;
126  }
127
128  //accept call
129  sockaddr_in client;
130  socklen_t clientSize= sizeof(client);
131  char host[NI_MAXHOST];
132  char svc[NI_MAXSERV];
133
134  int clientSocket = accept(listening, (sockaddr*)&client, &clientSize);
135
136  if (clientSocket == -1)
137  {
138    cerr<<"Problem with client connecting";
139    return -4;
140  }
```

Maintenant qu'un client est connecté, nous arrêtons d'attendre un client et nous fermons donc le *listening socket :*

```
142  //close the listening socket
143  close(listening);
144  memset(host,0,NI_MAXHOST);
145  memset(svc,0,NI_MAXSERV);
146  int result = getnameinfo((sockaddr*)&client,sizeof(client),host, NI_MAXHOST, svc, NI_MAXSERV,0);
147  if(result){
148  cout<<host<<"connected on "<< svc<<endl;
149  }
150  else{
151  inet_ntop(AF_INET, &client.sin_addr,host, NI_MAXHOST);
152  cout<<host<<"connected on" << ntohs(client.sin_port)<<endl;
153  }
```

Maintenant, nous attendons un message du client :

```
154  // Initialisation pour recevoir un message
155   char buf[4096];
156   while(true){
157   memset(buf,0,4096);
158   //wait for message
159   int byteRecv= recv(clientSocket, buf, 4096,0);
160   if(byteRecv==-1){
161     cerr<<"Connection issue"<<endl;
162     break;
163   }
164   if(byteRecv==0){
165     cout<<"The client disconnected"<<endl;
166     break;
167   }
```

Si la valeur du message est *0000* alors le robot va effectuer la mission que nous lui avons donné :

```
168  //Display message
169  cout<<"received "<<string(buf, 0, byteRecv)<<endl;
170  if( string(buf, 0, byteRecv)=="0000" && t<100){ // code pour faire avancer le robot = 0000 sinon pas le bon message
171  cout<<"Message bon"<<endl;
172  //for(int i=0;i<600;i++){
173
174  X=robot.getX()/10;      //Distance parcourue en x par le robot
175  Yr=a*pow(X,2)+b*X+c;
176  Xtan=(a*pow(X,2)-c)/(2*a*X+b);   //fonction modélisant l'erreur du robot
177  cout<<"----------"<<endl;
178  cout<<"X="<<X<<endl<<"Yr="<<Yr<<endl<<"theta calc"<<thetar<<endl;
179  cout<<"Y="<<robot.getY()<<endl<<"theta robot"<<robot.getTh()<<endl<<"Vrot"<<thetaresult<<endl;
```

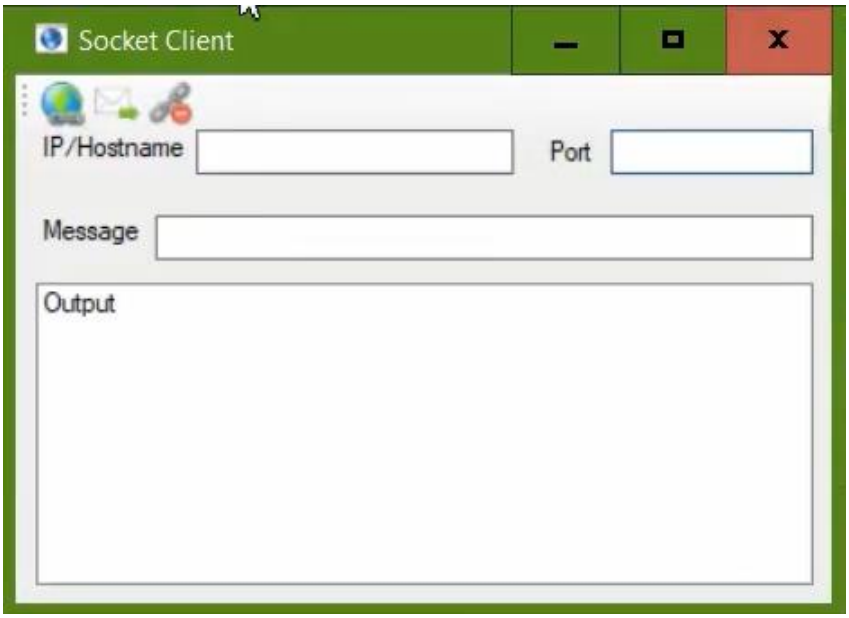
Après avoir réalisé sa mission nous renvoyons le message vers le client :

```
215      send(clientSocket, buf, byteRecv +1, 0);
216      cout<<"arret robot attente d'un message"<<endl;
217      }
218      else{
219      cout<<"Pas le bon message"<<endl;
220      }
221
222      //resend message
223      send(clientSocket, buf, byteRecv +1, 0);
224      }
225      //close socket
226      close(clientSocket);
```

Pour finir, si le message n'est pas le bon alors nous n'effectuons pas la mission. Dans chaque cas, nous arrêtons la communication après ça.

Pour tester si le code fonctionne, lancez-le et ouvrez une invite de commande et écrivez la commande *telnet localhost 8000.* Vous pouvez ensuite envoyer *0000* pour faire avancer le robot pendant un temps donné.

Coté client, voici la fenêtre que nous avons :



Pour nous connecter, nous devons indiquer l'IP et le port du serveur puis cliquer sur l'icône la plus à gauche, l'IP de la machine virtuelle peut être connue avec la fonction *ipconfig,* le port a été initié à 8000. Un message apparaîtra en dessous d'Output. Pour envoyer un message, il faut écrire dans la zone de même nom puis cliquer sur le l'icone du milieu. Enfin, pour arrêter la connexion, il faut cliquer sur l'icône la plus à droite.

Dans notre cas la connexion est donc possible et le serveur est déjà intégré dans le code contrôlant le robot. Il ne manque plus qu'à intégrer la partie client dans le code récupérant les données envoyées par le lecteur RFID.

7.La suite

Je n'ai malheureusement pas eu le temps d'intégrer la partie client dans le code côté RFID. Dès que cette intégration sera faite, il sera normalement possible de réaliser un fingerprint de façon automatiser. Après cela des améliorations seront possibles que ce soit coté RFID ou coté robot.

Nous pourrons ensuite faire les tests effectués par Elias HATEM de nouveau pour comparer nos valeurs.

Les tests de fingerprint pourront être fait de plusieurs manières afin de voir s'il y a des différences. Les changements peuvent être par exemple dans le nombre de balises utilisées ou la façon dont elles sont positionnées. Nous pouvons aussi voir s'il y a une différence dans nos valeurs en fonction du parcours effectué le robot. Ce qui nous permettrait de trouver la meilleure façon d'effectuer le fingerprint. Nous pourrons alors tester avec une autre technologie que la RFID comme le Bluetooth.

Il y a encore beaucoup de travail à effectuer sur ce projet.

Bibliographie

(1) Johann Borenstein et Liqiang Feng, The University of Michigan, UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots, en ligne <URL : http://www-personal.umich.edu/~johannb/Papers/paper60.pdf >, November 17, 1995.

(2) Agostino Martinelli, The Odometry Error of a Mobile Robot With a Synchronous Drive System, en ligne <URL : https://hal.inria.fr/inria-00359947/document>, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 18, NO. 3, JUNE 2002.

(3) Johann Borenstein, Internal Correction of Dead-reckoning Errors With the Smart Encoder Trailer, en ligne <URL : http://www.valentiniweb.com/piermo/robotica/doc/Borenstein/paper53.pdf>, International Conference on Intelligent Robots and Systems (lROS '94)- Advanced Robotic Systems and the Real World. , Munich, Germany, September 12-16, 1994, pp. 127-134.

Yong Liu, Robert L. Williams II, J. Jim Zhu, Jianhua Wu, Omni-directional mobile robot controller based on trajectory linearization, en ligne < URL : https://www.ohio.edu/mechanical-faculty/williams/html/PDF/JRAS08.pdf >, Robotics and Autonomous Systems 56 (2008) 461–479.

Kok Seng CHONG, Lindsay KLEEMAN, Accurate Odometry and Error Modelling for a Mobile Robot, en ligne <URL : https://ecse.monash.edu/centres/irrc/LKPubs/MECSE-1996-6.pdf >, Accurate Odometry and Error Modelling for a Mobile Robot, MECSE-1996-6.

Sung Kyung Hong and Sungsu Park, Minimal-Drift Heading Measurement using a MEMS Gyro for Indoor Mobile Robots , en ligne <URL : https://www.mdpi.com/1424-8220/8/11/7287/htm >,Published: 17 November 2008.

ROS, ROS – Robot Operating System, en ligne <URL : https://www.ros.org/ >.

Joaquín Ballesteros, Moodle, IA et ROS (M2 -2021S9), en ligne <URL : https://moodle.myefrei.fr/pluginfile.php/171097/mod_resource/content/1/Part_2_2020.pdf >.

VMWare, en ligne <URL : https://www.vmware.com/fr.html >.

University of Missouri, Vigir Missouri, Vision-Guided and Intelligent Robotics Laboratory, Software, en ligne <URL : http://vigir.missouri.edu/~gdesouza/Research/MobileRobotics/Software/ >.

University of Missouri, Vigir Missouri, Vision-Guided and Intelligent Robotics Laboratory, Pioneer 3 Operational Manual, en ligne <URL : http://vigir.missouri.edu/~gdesouza/Research/MobileRobotics/Software/P3OpMan5.pdf >.

MobileRobots, Pioneer Research/Academic Customer Support, en ligne <URL : https://web.archive.org/web/20160329153917/http://robots.mobilerobots.com/wiki/Main_Page >.

Web Archive, Internet archive way back machine, en ligne <URL : https://web.archive.org/ >.

Wiki ROS, ROSARIA, en ligne <URL : http://wiki.ros.org/ROSARIA>.

Adept mobilerobots, Pioneer 3-DX, en ligne <URL : https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf >.

Aria overview, MobileRobots Advanced Robotics Interface for Applications (ARIA) Developer's Reference Manual, en ligne <URL : https://www.eecs.yorku.ca/course_archive/2009-10/W/4421/doc/pioneer/aria/main.html#arexport >.

Cyberbotics, Adept's Pioneer 3-DX, en ligne <URL : https://cyberbotics.com/doc/guide/pioneer-3dx >.

Stackoverflow, en ligne< URL : https://stackoverflow.com/ >.

Lime Parallelogram, Socket Tutorial P.2 -- Communicating Between A Windows C# Client And A Python Server, en ligne <URL : https://www.youtube.com/watch?v=_k-E5GbOeQg >.

Sloan Kelly, Creating a TCP Server in C++ [Linux / Code Blocks], en ligne <URL : https://www.youtube.com/watch?v=cNdlrbZSkyQ >.