

<b>I RAPPORT DE STAGE</b>		<b>II M1</b>
<b><u>ELEVE</u></b>		
Nom: MA-PAW-YOUN	Prénom: Cédric	Filière: Systèmes Robotiques et Drones
<b><u>SUJET</u></b> Automatisation d'un fingerprint multi-variable à l'aide d'un robot mobile pour la localisation indoor		
<b><u>ENTREPRISE</u></b> Nom : AllianSTIC – EFREI Paris Adresse : 30- 32 Avenue de la République, 94800 Villejuif Adresse du lieu de stage si différent :		
<b><u>DATE DU STAGE</u></b>  Date de remise du rapport aux membres du Jury : 16/12/2019  Du : 24/04/2019                      au : 27/09/2019                      durée effective en semaines : 20		
<b><u>SOUTENANCE</u></b> Date : 18/12/2019                      Heure : 17 h 30 Composition du Jury : <ul style="list-style-type: none"><li>- Président (responsable EFREI ou ESIGETEL) : M. Belgacem BEN HEDIA</li><li>- Responsable du stage (Entreprise) : Mme. Elizabeth COLIN</li><li>- Invité(e) :</li></ul>		
<b><u>PUBLICATION DU RAPPORT DE STAGE</u></b>  Le responsable du stage :  autorise le stagiaire à publier le rapport de stage sur l'Intranet de l'Ecole.  Signature		
<b><u>Mots clés</u></b>		

# Sommaire

## Introduction générale

### **I      Prise en main du robot**

#### **A) ROSARIA**

#### **B) Fonctionnement de ARIA**

#### **C) Mes premiers programmes**

### **II     Problème de déviation du robot**

#### **A) Identification du problème**

- 1) Méthode d'identification du problème : parcours en ligne droite**
- 2) Méthode d'identification du problème : rotation**
- 3) Méthode d'identification du problème : Mesure des vitesses de rotation des roues**
- 4) Identification du profil de la déviation**
- 5) Tests parcours carré**

### **III    Mesures pour la localisation indoor**

#### **A) Les différents scénarios de mesures**

- 1) Scénario 1 : Single Tag Center**
- 2) Scénario 2 : Constellation mur 1**
- 3) Scénario 3 : Localisation**
- 4) Scénario 4 : Constellation sur chaque mur**
- 5) Scénario 5 : Single Tag Wall 2**
- 6) Scénario 6 : Single Tag Wall 4**
- 7) Scénario 7 : Constellation mur 4**

## Conclusion générale

## Introduction générale

Dans le cadre de mon stage de M1, j'ai eu la chance d'effectuer un stage de recherche au sein même de l'école dans le laboratoire AllianSTIC qui est le laboratoire de recherche de Efrei Paris. AllianSTIC a été créé suite au partenariat de deux écoles d'ingénieurs : l'ESIGETEL – école d'ingénieur généraliste en sciences du numérique et l'EFREI – école d'ingénieur généraliste en informatique. Suite à la fusion des deux écoles le laboratoire continue ses travaux à EFREI Paris.

Le laboratoire est dirigé par le professeur Katarzyna Wegrzyn-Wolska, Docteur en Informatique Temps Réel, Robotique et Automatique, habilitée à diriger des recherches. Le laboratoire est composé des enseignants-chercheurs de Efrei Paris.

Le laboratoire AllianSTIC travaille sur deux grands axes de recherche. Premièrement Big Data et Web Intelligence et deuxièmement Objets connectés et localisation indoor. Les enseignants-chercheurs du laboratoire répondent à différents projets qui s'inscrivent dans ces axes de recherche. Ces projets peuvent être d'ampleur nationale ou internationale en collaboration avec d'autres organismes partenaires

Mon projet s'inscrit dans un des axes de recherche du laboratoire, celui des objets connectés et de la localisation indoor. En effet, mon projet s'intitule « Automatisation d'un fingerprint multi-variable à l'aide d'un robot mobile pour la localisation indoor ». L'axe de recherche principal de ce projet concerne la localisation indoor. La localisation indoor désigne le fait de pouvoir localiser une personne ou un objet à l'intérieur d'un bâtiment de manière précise. A l'heure actuelle la localisation indoor est encore à l'état de recherche et aucune des solutions proposées ne satisfait suffisamment les attentes pour être commercialisé à grande échelle. La solution choisie par le laboratoire AllianSTIC est celle qu'on appelle communément « fingerprinting ». Cette méthode s'appuie sur les empreintes des signaux. Le principe de base du fingerprinting est relativement simple. En effet, cette méthode se décompose en deux étapes que sont l'étalonnage et la localisation. Tout d'abord, l'étalonnage consiste à mesurer certaines caractéristiques d'un signal, dans mon cas le RSSI (Received Signal Strength Indication) qui désigne la puissance reçue d'un signal à différents points de la zone dans laquelle on va faire la localisation. L'étalonnage consiste donc à dresser une base de données des RSSI des signaux à toutes les positions possibles. Suite à cet étalonnage on peut passer à la deuxième étape qui est la localisation. Pour la localisation, un dispositif mobile tel qu'un smartphone va faire une mesure de RSSI à l'endroit où il se situe et un algorithme va comparer cette valeur avec les valeurs de la base de données créée précédemment. Le smartphone pourra ainsi savoir à quelle position il se situe.

Dans cette méthode de fingerprinting, la phase d'étalonnage demande un temps important car il faut mesurer les RSSI des signaux à de nombreux points de la zone. Ces mesures constituent un travail fastidieux pour un homme. C'est pourquoi certains ont eu l'idée d'automatiser cette tâche. Ainsi mon stage s'inscrit dans cette idée. Afin d'étalonner une zone on peut utiliser un robot mobile qui fera les mesures à la place d'un homme. Si on monte l'outil de mesure de RSSI sur le robot et qu'on le programme afin de faire les mesures on peut automatiser l'échantillonnage de la pièce. Le but du stage est donc de prendre en

main le robot et le programmer afin qu'il fasse les prises de mesures du RSSI dans une zone définie. Le robot sur lequel je travaille est le Pioneer 3-DX de la marque MobileRobots.



*Pioneer 3-DX de MobileRobots*

## **I. Prise en main du robot**

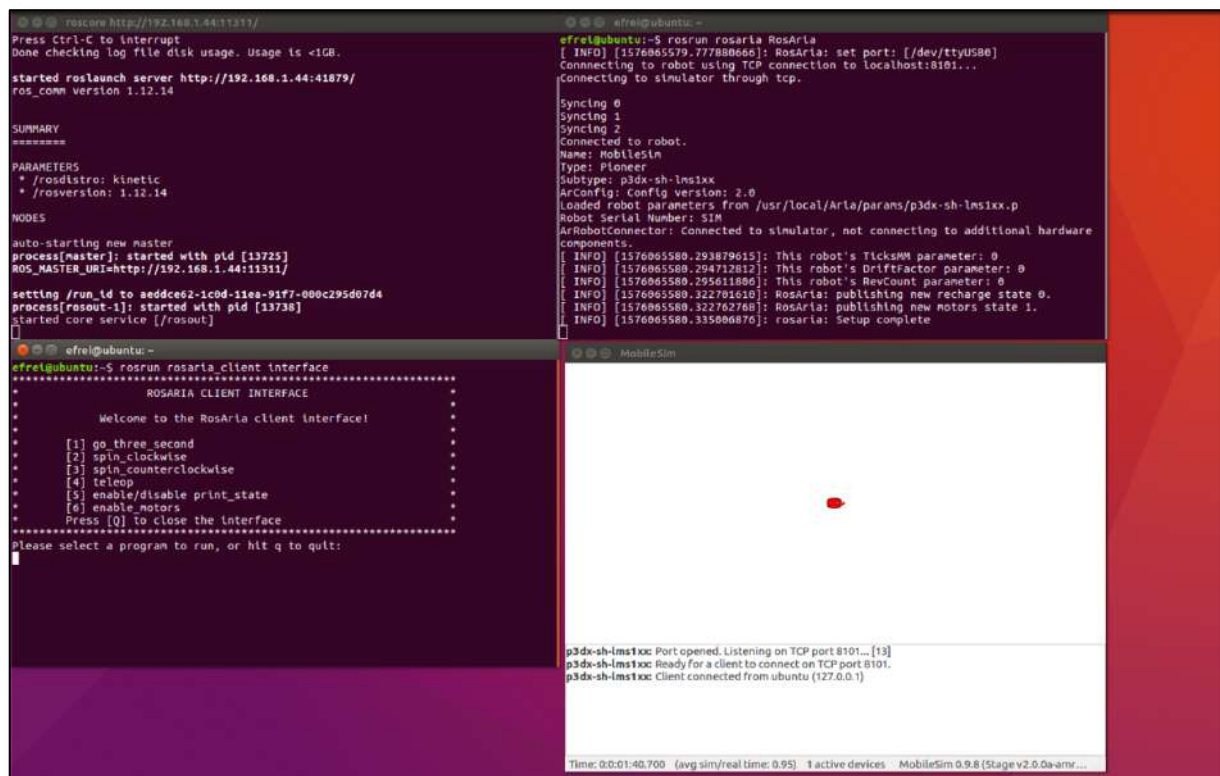
Le Pioneer 3-DX de MobileRobots nécessite d'installer le SDK fourni par la marque afin de pouvoir développer des programmes pour le robot. Advanced Robot Interface for Applications (ARIA) est une bibliothèque C++ pour toutes les plateformes de MobileRobots. ARIA permet de contrôler de façon dynamique les différents paramètres du robot tel que la vitesse ou le cap. ARIA permet également d'accéder aux différents capteurs du robot tels que les sonars ou les bumpers. ARIA est compatible avec Windows et Linux. Pour ma part j'ai choisi de l'installer sur une machine virtuelle Ubuntu car je suis plus familier avec l'environnement Linux. L'installation de ARIA s'est avérée plus complexe que prévu car le site du constructeur, où on est censé pouvoir télécharger le SDK, n'existe plus car l'entreprise a fermé depuis quelques années. J'ai heureusement pu trouver une version archivée du site avec le SDK encore disponible en téléchargement. Une fois ARIA installé, on peut commencer à contrôler le robot. Ma première prise en main du robot s'est effectuée avec ROS.

### **A) ROSARIA**

ROS ou Robot Operating System est un « Robotic Middleware » soit un ensemble de bibliothèques et d'outils open-sources qui permettent de créer des applications pour des robots. ROS est la couche centrale qui fait le lien entre la partie software, le code source, et la partie hardware, les actions du robot. Pour communiquer avec le robot il y a, dans ROS, une structure appelée « Node ». Un « node » est un fichier exécutable dans un paquet ROS. Il est possible d'exécuter plusieurs nodes en même temps. Chaque node va correspondre à une certaine action et ces nodes peuvent communiquer entre eux. Par exemple, on peut lancer un node pour contrôler les déplacements du robot avec les touches du clavier et, en même temps, on peut lancer un autre node pour afficher la vitesse linéaire du robot. Les deux nodes communiquent entre eux afin d'afficher la vitesse correspondante au déplacement du robot.

Dans le cas de notre robot, le Pioneer 3-DX, il existe un node appelé ROSARIA qui a été créé à partir de la bibliothèque ARIA de MobileRobots. Ce node contient plusieurs topics pour interagir

avec le robot. Dans ROS, les topics sont les données que les nodes envoient ou reçoivent, c'est-à-dire par exemple les commandes de déplacement envoyées ou encore les données reçues des capteurs. Dans le node ROSARIA, il y a un topic pour chaque élément des robots de MobileRobots. Ainsi, on a par exemple un topic pour lire les données renvoyées par les sonars ou alors, on a aussi un topic pour voir le niveau de batterie du robot.



```

efrei@ubuntu:~$ roslaunch rosaria RosAria
[ INFO ] [1576865579.777880666]: RosAria: set port: [/dev/ttyUSB0]
Connecting to robot using TCP connection to localhost:8101...
Connecting to simulator through tcp.

Syncing 0
Syncing 1
Syncing 2
Connected to robot.
Name: MobileSim
Type: Pioneer
Subtype: p3dx-sh-lms1xx
ArConfig: Config version: 2.0
Loaded robot parameters from /usr/local/Arlio/params/p3dx-sh-lms1xx.p
Robot Serial Number: SIM
ARRobotConnector: Connected to simulator, not connecting to additional hardware components.
[ INFO ] [1576865580.293879615]: This robot's TicksMM parameter: 0
[ INFO ] [1576865580.294712812]: This robot's Driftfactor parameter: 0
[ INFO ] [1576865580.295611806]: This robot's Revcount parameter: 0
[ INFO ] [1576865580.322701610]: RosAria: publishing new recharge state 0.
[ INFO ] [1576865580.322762768]: RosAria: publishing new motors state 1.
[ INFO ] [1576865580.335806876]: rosaria: Setup complete

efrei@ubuntu:~$ roslaunch rosaria_client interface
*****
ROSARIA CLIENT INTERFACE
*****
Welcome to the RosAria client Interface!
*****
[1] go three second
[2] spin clockwise
[3] spin counterclockwise
[4] teleop
[5] enable/disable print_state
[6] enable motors
Press [q] to close the Interface
*****
Please select a program to run, or hit q to quit:

```

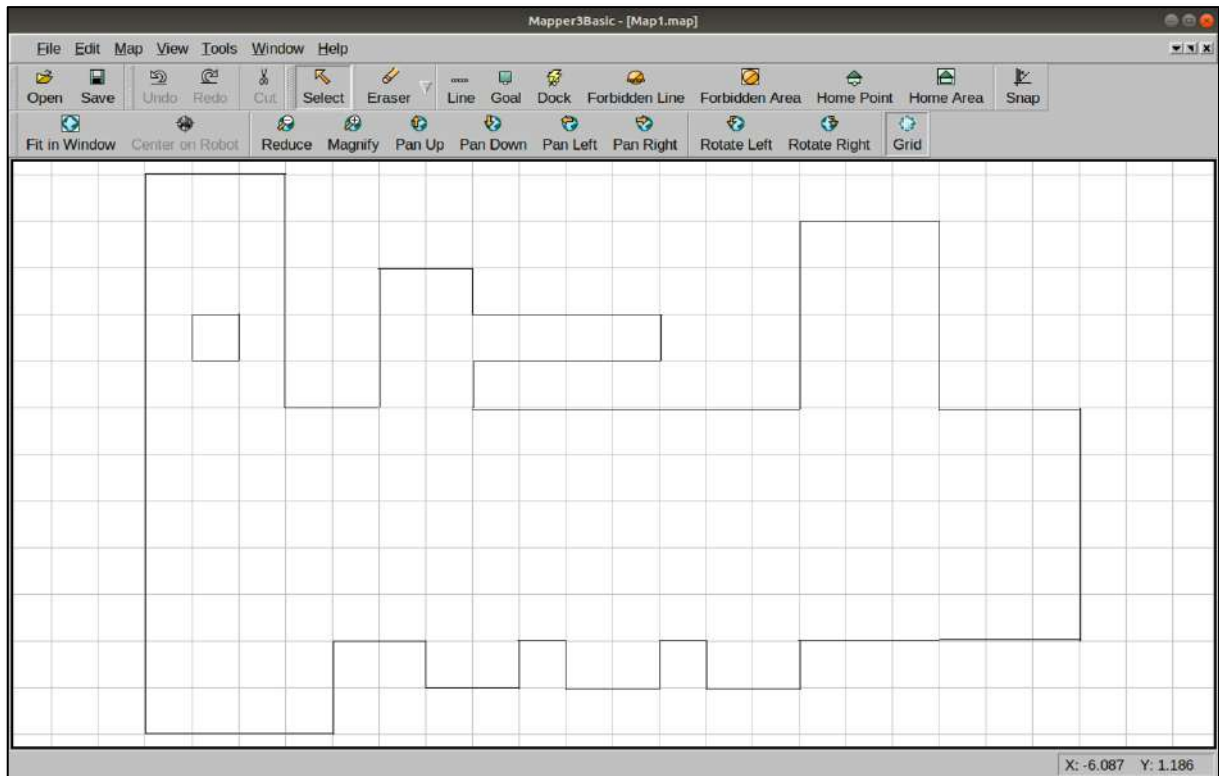
Commande du robot sur le simulateur avec ROS

## B) Fonctionnement de ARIA

Après avoir pris en main le robot avec ROS, il est temps maintenant de commencer à développer mes premiers programmes C++ à l'aide de la librairie ARIA. Tout d'abord il est important de comprendre comment ARIA fonctionne. ARIA nous offre deux façons d'envoyer des commandes au robot. Premièrement de façon simple avec des commandes bas niveau qui gèrent les instructions les plus simples de commande du robot tel que, par exemple, avancer, accélérer ou encore tourner. Si on veut effectuer des tâches plus complexes, ARIA dispose d'un système haut niveau appelé Actions. Les actions sont des objets individuels qui fournissent indépendamment des demandes de mouvement qui sont évaluées et ensuite combinées à chaque cycle pour produire un ensemble final de commandes de mouvement. Cela permet de construire des comportements complexes à partir de simples blocs de construction pour un contrôle dynamique et continu du mouvement. Pour créer une action il faut créer une sous-classe de la classe principale ArAction.

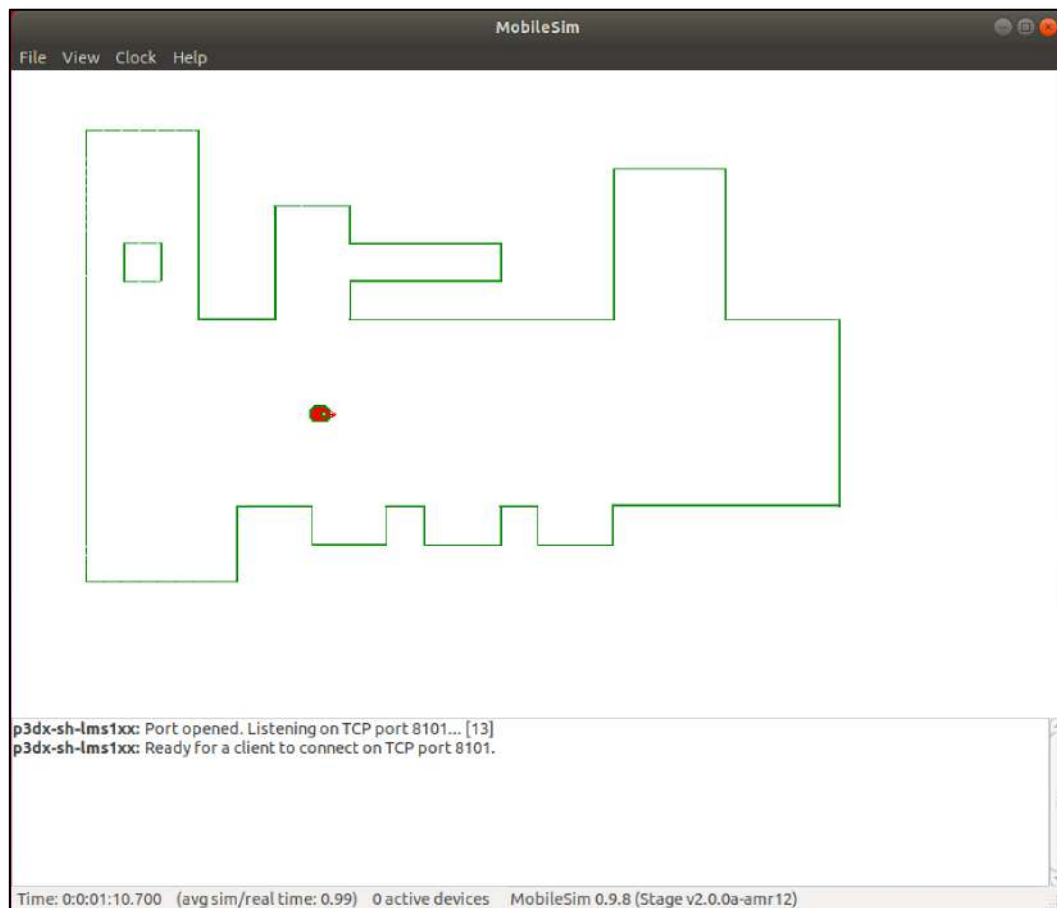
En complément de ARIA, MobileRobots fournit également le logiciel MobileSim qui est un simulateur dans lequel on peut tester ses programmes avant de les implémenter sur le robot. MobileSim permet de simuler tous les robots de la marque et donc notamment le Pioneer 3-DX. MobileSim simule tous les capteurs du robot. De plus, il est possible de charger la carte d'une zone afin de simuler le comportement du robot dans un environnement avec des

obstacles. A l'installation MobileSim comporte déjà des cartes d'exemples mais il est également possible d'en créer d'autre en utilisant un autre logiciel fourni par la marque, Mapper3. Mapper3 permet de créer des cartes de façon simple en dessinant les obstacles à l'aide des outils fournis.



*Interface de Mapper3*

En utilisant ces cartes dans MobileSim on peut savoir en amont le comportement du robot dans l'environnement voulu.



*Utilisation de la carte créée dans MobileSim*

### **C) Mes premiers programmes**

Afin de prendre en main la librairie ARIA, les premiers programmes que j'ai voulu développer étaient essentiellement des programmes simples de déplacement du robot à l'aide des fonctions basiques fournies par ARIA. Pour développer un programme pour le robot il faut suivre un template bien spécifique pour que le robot puisse interpréter les actions qu'on lui demande. Ce template est fourni avec ARIA. De plus, afin de bien comprendre comment on développe un programme pour le robot, ARIA fournit des programmes d'exemples qui permettent de tester les différentes fonctions fournies.

```

1  #include "Aria.h"
2
3  int main(int argc, char **argv)
4  {
5      Aria::init();
6      ArArgumentParser argParser(&argc, argv);
7      argParser.loadDefaultArguments();
8      ArRobot robot;
9      ArRobotConnector robotConnector(&argParser, &robot);
10
11     if(!robotConnector.connectRobot())
12     {
13         ArLog::log(ArLog::Terse, "Could not connect to the robot.");
14         if(argParser.checkHelpAndWarnUnparsed())
15         {
16             // -help not given, just exit.
17             Aria::logOptions();
18             Aria::exit(1);
19             return 1;
20         }
21     }
22
23     // Trigger argument parsing
24     if (!Aria::parseArgs() || !argParser.checkHelpAndWarnUnparsed())
25     {
26         Aria::logOptions();
27         Aria::exit(1);
28         return 1;
29     }
30
31     ArKeyHandler keyHandler;
32     Aria::setKeyHandler(&keyHandler);
33     robot.attachKeyHandler(&keyHandler);
34
35     puts("Write an explanation of the program here");
36
37     ArSonarDevice sonar;
38     robot.addRangeDevice(&sonar);
39
40     robot.runAsync(true);
41
42     /* Write all the commands here */
43
44     // wait for robot task loop to end before exiting the program
45     robot.waitForRunExit();
46
47     Aria::exit(0);
48     return 0;
49 }
50

```

Importation du fichier header qui contient toutes les classes nécessaires.

Initialisation de tous les objets.  
ArRobot est la classe la plus importante car c'est la classe qui permet de communiquer avec le robot et de le contrôler.

Si le programme n'arrive pas à se connecter au robot on affiche un message d'erreur et le programme s'arrête.

Vérifie que le programme a tous les arguments requis.  
Si ce n'est pas le cas, le programme s'arrête.

Activation des sonars du robot

Lance le traitement de l'instance dans un nouveau thread

Le programme attend que toutes les commandes soient exécutées avant de s'arrêter.

Template de base pour tous les programmes développés avec ARIA

Les exemples fournis à l'installation de ARIA nous permettent de comprendre aussi comment utiliser les actions dans le code. Il faut tout d'abord créer tous les objets des classes dont on a besoin et ensuite utiliser la fonction *addAction* de la classe principale *ArRobot* pour que le robot puisse exécuter les actions demandées.



```

55 // turn on the motors, turn off amigobot sounds
56 robot.enableMotors();
57 robot.comInt(ArCommands::SOUNDTOG, 0);
58
59 // add a set of actions that combine together to effect the wander behavior
60 ArActionStallRecover recover;
61 ArActionBumpers bumpers;
62 ArActionAvoidFront avoidFrontNear("Avoid Front Near", 225, 0);
63 ArActionAvoidFront avoidFrontFar;
64 ArActionConstantVelocity constantVelocity("Constant Velocity", 400);
65 robot.addAction(&recover, 100);
66 robot.addAction(&bumpers, 75);
67 robot.addAction(&avoidFrontNear, 50);
68 robot.addAction(&avoidFrontFar, 49);
69 robot.addAction(&constantVelocity, 25);

```

*Extrait du code wander.cpp : code d'exemple où le robot se balade en évitant les obstacles.*

### Mon premier programme : Test1.cpp

```

99 robot.lock();
100 ArLog::log(ArLog::Normal, "test1: Pose=(%.2f,%.2f,%.2f), Trans. Vel=%.2f, Rot. Vel=%.2f, Battery=%.2fV",
101 robot.getX(), robot.getY(), robot.getTh(), robot.getVel(), robot.getRotVel(), robot.getBatteryVoltage());
102 robot.unlock();
103
104 // Set forward velocity to 250 mm/s
105 ArLog::log(ArLog::Normal, "test1: Driving forward at 500 mm/s for 20 sec...");
106 robot.lock();
107 robot.enableMotors();
108 robot.setVel(500);
109 robot.unlock();
110 ArUtil::sleep(10000);
111
112 ArLog::log(ArLog::Normal, "test1: Stopping.");
113 robot.lock();
114 robot.stop();
115 robot.unlock();
116 ArUtil::sleep(1000);
117
118 ArLog::log(ArLog::Normal, "test1: 180° rotation");
119 robot.lock();
120 robot.setRotVel(10);
121 robot.unlock();
122 ArUtil::sleep(18000);
123
124 ArLog::log(ArLog::Normal, "test1: Stopping.");
125 robot.lock();
126 robot.stop();
127 robot.unlock();
128 ArUtil::sleep(1000);
129
130 // Set forward velocity to 250 mm/s
131 ArLog::log(ArLog::Normal, "test1: Driving forward at 500 mm/s for 20 sec...");
132 robot.lock();
133 robot.enableMotors();
134 robot.setVel(500);
135 robot.unlock();
136 ArUtil::sleep(10000);
137
138 ArLog::log(ArLog::Normal, "test1: Stopping.");
139 robot.lock();
140 robot.stop();
141 robot.unlock();
142 ArUtil::sleep(1000);

```

Pour faire avancer le robot on définit une vitesse fixe que le robot va maintenir pendant un certain temps

On stoppe le robot pendant une seconde entre chaque déplacement pour éviter les conflits.

Pour faire tourner le robot on définit une vitesse de rotation fixe que le robot va maintenir pendant un certain temps

```

143
144     robot.lock();
145     ArLog::log(ArLog::Normal, "test1: Pose=(%.2f,%.2f,%.2f), Trans. Vel=%.2f, Rot. Vel=%.2f, Battery=%.2fV",
146         robot.getX(), robot.getY(), robot.getTh(), robot.getVel(), robot.getRotVel(), robot.getBatteryVoltage());
147     robot.unlock();
148
149
150     ArLog::log(ArLog::Normal, "test1: Ending robot thread...");
151     robot.stopRunning();
152
153     // wait for the thread to stop
154     robot.waitForRunExit();
155
156     // exit
157     ArLog::log(ArLog::Normal, "test1: Exiting.");
158     Aria::exit(0);
159     return 0;
160 }
161

```

Ce premier programme avait pour but de comprendre comment fonctionne les fonctions principales du robot que sont le déplacement en ligne droite et la rotation. Pour le déplacement en ligne droite, on peut également utiliser la fonction `move()` où on indique la distance à laquelle le robot doit se rendre. Cependant la distance limite de cette fonction est de 5 mètres ce qui est insuffisant dans la plupart des cas. C'est pourquoi on privilégie l'utilisation de la fonction `setVel()` avec laquelle on va définir une vitesse linéaire pour le robot. On va ensuite définir le temps pendant lequel le robot va garder cette vitesse. Ainsi on peut avoir définir la distance à laquelle le robot va se déplacer. Il est important de stopper le robot après chaque mouvement, déplacement ou rotation afin d'éviter au maximum les conflits entre les différentes commandes du robot. Dans ce premier programme on n'utilise le système d'actions car on ne cherche qu'à envoyer des commandes simples au robot.

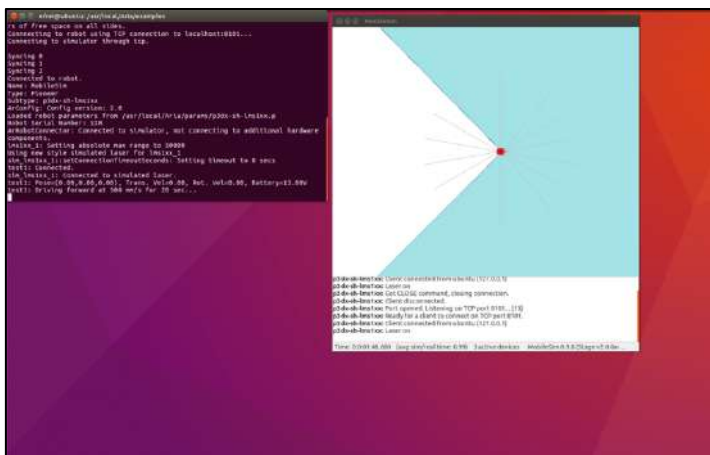


Figure 1 : Le robot avance en ligne droite

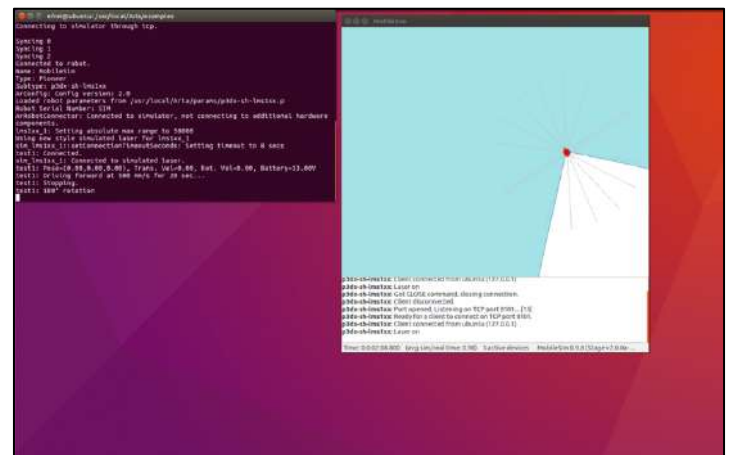


Figure 2 : Le robot tourne de 180°

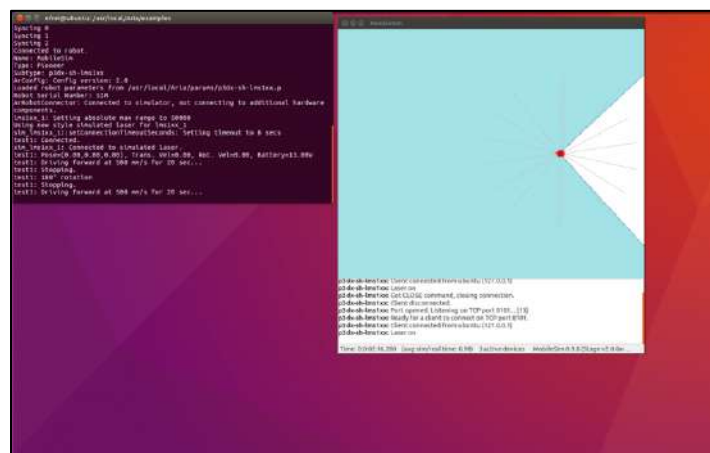


Figure 3 : Le robot avance en ligne droite

Le programme fonctionne bien sur le simulateur. On peut donc le tester sur le robot. Pour ça on va se placer dans le couloir et faire avancer le robot. Après quelques tests on constate que le robot n'avance pas en ligne droite. Il a tendance à dévier vers la gauche au bout de quelques mètres et à cogner le mur. Puisque sur le simulateur on ne constate pas ce problème il s'agit donc d'un problème matériel lié au robot en lui-même. Ainsi, l'une des premières étapes du stage va être de caractériser le problème et de trouver une solution à celui-ci.

## II. Problème de déviation du robot

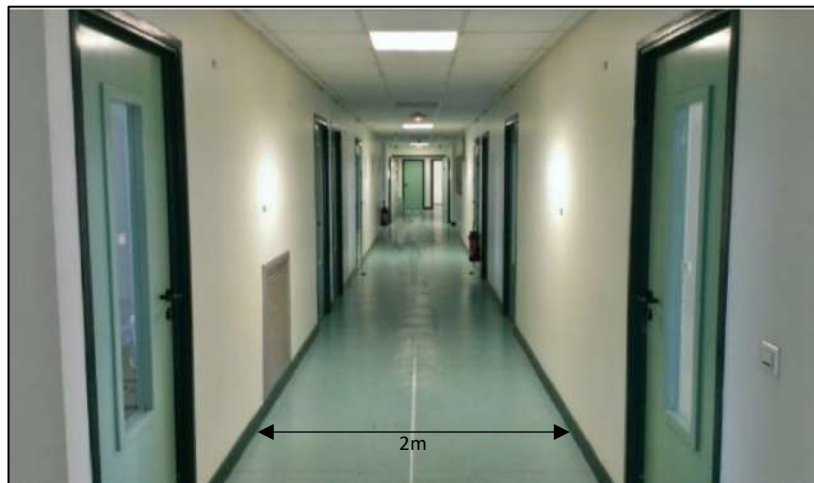
### A) Identification du problème

Après quelques tests simples de déplacement du robot, on constate une déviation vers la gauche sur des trajets en ligne droite. Les tests ont montré qu'il s'agissait d'un problème matériel et non logiciel. Ainsi la première étape du projet va être de corriger la déviation du robot. En effet, pour les prises de mesures pour la localisation, on a besoin d'une précision importante et cette déviation n'est donc pas tolérable.

Afin de corriger cette déviation il est important de commencer par l'identifier de façon précise.

#### 1) Méthode d'identification du problème : parcours en ligne droite

La première étape afin d'identifier le problème de déviation a consisté à essayer de faire parcourir au robot une grande distance en ligne droite. Pour cela j'ai utilisé le couloir du 4<sup>ème</sup> étage du bâtiment A qui fait 20 mètres de long.



Méthode de test :

- On place le robot à une extrémité du couloir à une distance de 80 cm du mur gauche.
- On lance un programme sur le robot qui doit le faire avancer de 20 m en ligne droite.
- Quand le robot est sur le point de toucher le mur de gauche à cause de la déviation, on mesure la distance entre la position actuelle du robot et sa position initiale.
- On effectue ce test 3 fois et on calcule la déviation sur 1 mètre pour chacun de ces 3 tests.
- On calcule la moyenne des déviations.

Distance parcourue (m)	Déviation (cm)
11,75	75
11,9	80
11,8	79

Distance parcourue (m)	Déviation (cm)
1	6,383
1	6,723
1	6,695
Moyenne	6,600

On constate donc une déviation d'environ 6,6 cm sur 1 m.

Cependant dans ce test on prend l'hypothèse que l'erreur est linéaire et intervient dès le début du parcours. Or, dans la pratique, on constate à l'œil nu que la déviation intervient après une certaine distance.

Pour confirmer cette observation on essaye de faire la même expérience mais en faisant des pauses tous les mètres. On veut vérifier la valeur de la moyenne de la déviation sur un mètre. En effectuant le test 3 fois et en calculant la moyenne on trouve les résultats suivants :

Distance parcourue (m)	Déviation (cm)
11,8	80
11,4	77
11,6	78

Distance parcourue (m)	Déviation (cm)
1	6,780
1	6,754
1	6,724
Moyenne	6,753

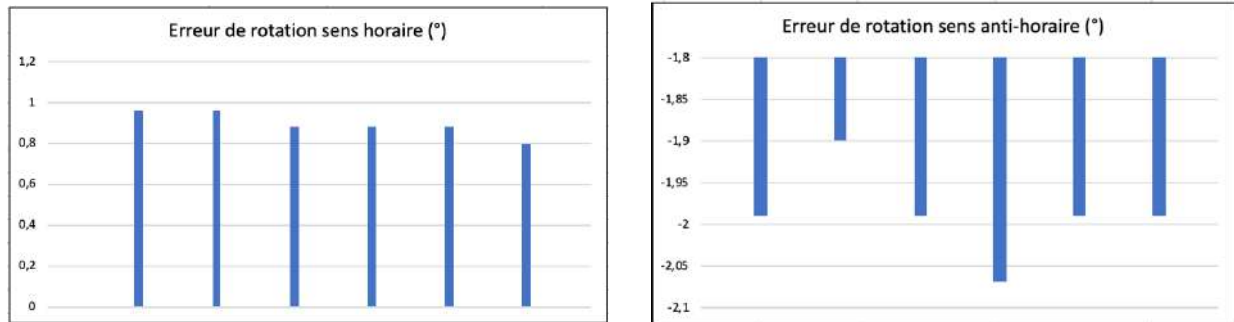
On constate donc ici aussi une déviation d'environ 6,7 cm sur 1 m.

On constate ainsi que les différentes pauses n'ont pas d'impact sur la déviation du robot. Si l'erreur était linéaire on aurait pu faire en sorte de compenser le problème de déviation en augmentant la vitesse de rotation de la roue de gauche afin de maintenir une trajectoire en ligne droite. Cependant la déviation n'intervient pas au début du trajet mais apparaît au bout de quelques mètres, ce qui complique la correction. Ainsi, afin de corriger l'erreur, il va falloir tout d'abord identifier d'où elle provient.

## 2) Méthode d'identification du problème : rotation

Pour identifier s'il y a une différence de vitesse de rotation entre les deux roues, on va faire tourner le robot sur lui-même sur 360° dans le sens horaire et dans le sens antihoraire à la même vitesse et on va mesurer l'angle réellement effectué dans les deux sens. Dans le tableau ci-dessous on répertorie les écarts d'angles par rapport à 360° qui correspondent aux erreurs lors du test de rotation. On calcule aussi la moyenne de ces erreurs.

Rotation (mesure capteur robot)	
- (horaire)	+ (Anti horaire)
0,96°	-1,99°
0,96°	-1,9°
0,88°	-1,99°
0,88°	-2,07°
0,88°	-1,99°
0,79°	-1,99°
Moyenne	
0,891666667°	-1,988333333°



D'après ces mesures on constate que, quand le robot tourne dans le sens horaire, il arrive presque à faire 360° puisqu'il tourne de 359,1° en moyenne. Mais en prenant en compte les incertitudes de mesures, on peut dire que, dans le sens horaire, le robot fait bien un tour complet. Cependant, dans le sens antihoraire, le robot a tendance à tourner de plus que 360°. En moyenne il tourne d'environ 362°.

### 3) Méthode d'identification du problème : Mesure des vitesses de rotation des roues

Après quelques recherches dans la documentation d'ARIA j'ai trouvé des fonctions qui permettent d'obtenir les vitesses linéaires de chacune des roues. La fonction `getLeftVel()` permet d'obtenir la vitesse de la roue gauche tandis que la fonction `getRightVel()` permet d'obtenir la vitesse de la roue de droite.

```
54 while(robot.isMoveDone() != true)
55 {
56     double leftSpeed = 0.0;
57     double rightSpeed = 0.0;
58     leftSpeed = robot.getLeftVel(); //Get the velocity of the left wheel
59     rightSpeed = robot.getRightVel(); //Get the velocity of the right wheel
60     cout << "Left velocity = " << leftSpeed << " mm/s" << endl;
61     cout << "Right velocity = " << rightSpeed << " mm/s" << endl;
62     sleep(1);
63 }
```

On fait donc un programme dans lequel on affiche la vitesse de chacune des roues toutes les secondes.

On va donc faire un test en faisant avancer le robot en ligne droite sur 5 m en affichant la vitesse de chacune des roues toutes les secondes ce qui permettra de voir si une des roues tourne plus vite que l'autre. On répète l'expérience trois fois afin de pouvoir comparer les résultats.

Test 1 (parcours direct sur 5 m)		
LEFT VELOCITY (mm/s)	RIGHT VELOCITY (mm/s)	DIFFERENCE
248	249	1
546	547	1
747	752	5
747	751	4
747	745	-2
750	749	-1
699	715	16
404	411	7
MOYENNE		3,875

Test 2 (parcours direct sur 5 m)		
LEFT VELOCITY (mm/s)	RIGHT VELOCITY (mm/s)	DIFFERENCE
249	256	7
549	548	-1
749	751	2
748	749	1
751	753	2
753	750	-3
699	709	10
413	393	-20
MOYENNE		-0,25



Test 3 (parcours direct sur 5 m)		
LEFT VELOCITY (mm/s)	RIGHT VELOCITY (mm/s)	DIFFERENCE
252	248	-4
548	547	-1
749	747	-2
750	749	-1
754	752	-2
749	743	-6
699	707	8
411	397	-14
	MOYENNE	-2,75

On va convertir les vitesses linéaires (mm/s) en vitesse de rotation (rad/s). En effet on sait que :

$$v_L = R_L \cdot \omega_L \quad v_R = R_R \cdot \omega_R$$

Avec :

$v_L$  : vitesse linéaire de la roue gauche

$v_R$  : vitesse linéaire de la roue droite

$\omega_L$  : vitesse de rotation de la roue gauche

$\omega_R$  : vitesse de rotation de la roue droite

R : Rayon des roues

Sur le Pioneer 3-DX on a  $D = 185$  mm donc  $R = D/2 = 92,5$  mm

Ainsi en convertissant les vitesses linéaires en vitesses de rotation on obtient les tableaux suivants :

Test 1 (parcours direct sur 5 m)		
LEFT VELOCITY (rad/s)	RIGHT VELOCITY (rad/s)	DIFFERENCE (right - left)
2,681081081	2,691891892	0,010810811
5,902702703	5,913513514	0,010810811
8,075675676	8,12972973	0,054054054
8,075675676	8,118918919	0,043243243
8,075675676	8,054054054	-0,021621622
8,108108108	8,097297297	-0,010810811
7,556756757	7,72972973	0,172972973
4,367567568	4,443243243	0,075675676
	MOYENNE	0,041891892



Test 2 (parcours direct sur 5 m)		
LEFT VELOCITY (rad/s)	RIGHT VELOCITY (rad/s)	DIFFERENCE (right - left)
2,691891892	2,767567568	0,075675676
5,935135135	5,924324324	-0,010810811
8,097297297	8,118918919	0,021621622
8,086486486	8,097297297	0,010810811
8,118918919	8,140540541	0,021621622
8,140540541	8,108108108	-0,032432432
7,556756757	7,664864865	0,108108108
4,464864865	4,248648649	-0,216216216
	MOYENNE	-0,002702703

Test 3 (parcours direct sur 5 m)		
LEFT VELOCITY (rad/s)	RIGHT VELOCITY (rad/s)	DIFFERENCE (right - left)
2,724324324	2,681081081	-0,043243243
5,924324324	5,913513514	-0,010810811
8,097297297	8,075675676	-0,021621622
8,108108108	8,097297297	-0,010810811
8,151351351	8,12972973	-0,021621622
8,097297297	8,032432432	-0,064864865
7,556756757	7,643243243	0,086486486
4,443243243	4,291891892	-0,151351351
	MOYENNE	-0,02972973

La colonne différence des tableaux correspond à la différence de vitesse entre la roue droite et la roue gauche (RIGHT VELOCITY – LEFT VELOCITY). On constate que sur les trois essais, la moyenne est quasiment nulle à chaque fois, ce qui voudrait dire qu'il n'y a pas de différences de vitesses entre les deux roues. Cependant il faut faire attention à ces résultats car ces mesures proviennent des capteurs internes du robot et on ne connaît pas l'efficacité et la précision de ces capteurs. Pour obtenir de meilleures mesures et vérifier celles prises par le robot il aurait fallu mesurer la vitesse de rotation des roues manuellement à l'aide d'un tachymètre. Cependant je ne disposais pas de cet outil de mesure. Ainsi, en conclusion ces tests de mesures de vitesses des roues n'ont pas permis d'identifier de façon précise la cause des déviations du robot.

#### 4) Identification du profil de la déviation

Quand on regarde le robot se déplacer en ligne droite, on aurait tendance à dire que la déviation vers la gauche intervient seulement après quelques mètres parcourus. Afin de vérifier cette hypothèse, on va procéder à un autre test. On va essayer de faire parcourir au robot une distance de 20 m en ligne droite en faisant des pauses tous les mètres, puisqu'on a vu précédemment que les pauses n'avaient pas d'impact sur la déviation. A chaque pause du robot, tous les mètres, on va mesurer la distance parcourue par rapport au point de départ du robot mais aussi la distance par rapport aux deux murs du couloirs. Pour ce faire j'ai placé au sol, tout le long du couloir, un mètre ruban d'une longueur de 30 m qui permettra de voir la distance parcourue par le robot. De plus, à chaque arrêt du robot, j'ai mesuré la distance par rapport au mur de droite du couloir à l'aide d'un télémètre laser. Comme je connais la largeur du couloir ainsi que la distance initiale du robot par rapport au mur de droite je peux calculer la déviation en distance du robot tout au long de son parcours. Après trois essais on obtient les tableaux et les courbes suivantes :

Test parcours couloir avec arrêt tous les mètres (test 1)			
Distance parcourue (m)	Distance par rapport au mur de droite (cm)	Distance par rapport au mur de gauche (cm)	Déviation
0 m	70 cm	130 cm	0 cm
1,006 m	70 cm	130 cm	0 cm
2,012 m	70,5 cm	129,5 cm	0,5 cm
3,019 m	72,7 cm	127,3 cm	2,7 cm
4,025 m	76 cm	124 cm	6 cm
5,031 m	83,3 cm	116,7 cm	13,3 cm
6,037 m	86,2 cm	113,8 cm	16,2 cm
7,044 m	94,5 cm	105,5 cm	24,5 cm
8,051 m	103,9 cm	96,1 cm	33,9 cm
9,057 m	113,9 cm	86,1 cm	43,9 cm
10,064 m	126 cm	74 cm	56 cm
11,07 m	138,5 cm	61,5 cm	68,5 cm
11,9 m	154,6 cm	45,4 cm	84,6 cm

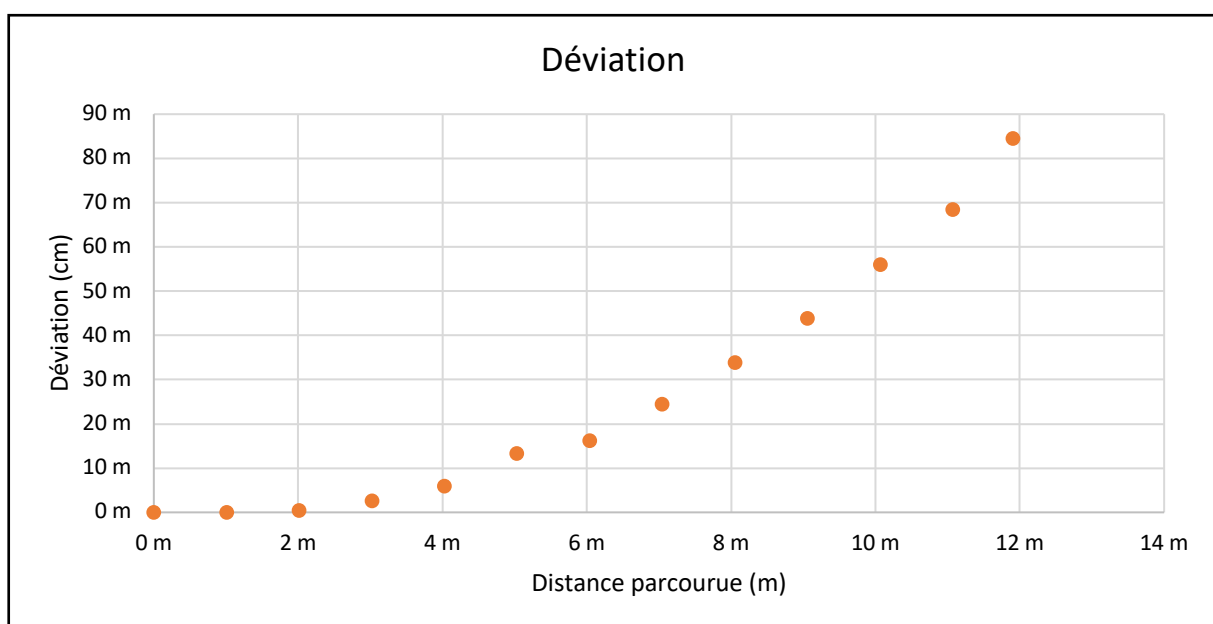


Figure 1 : Courbe de la déviation obtenue au test 1

Test parcours couloir avec arrêt tous les mètres (test 2)			
Distance parcourue (m)	Distance par rapport au mur de droite (cm)	Distance par rapport au mur de gauche (cm)	Déviation
0 m	70 cm	130 cm	0 cm
1,006 m	67,7 cm	132,3 cm	-2,3 cm
2,012 m	66,1 cm	133,9 cm	-3,9 cm
3,018 m	65,3 cm	134,7 cm	-4,7 cm
4,024 m	66,2 cm	133,8 cm	-3,8 cm
5,03 m	71 cm	129 cm	1 cm
6,036 m	70,3 cm	129,7 cm	0,3 cm
7,042 m	74,5 cm	125,5 cm	4,5 cm
8,048 m	78,4 cm	121,6 cm	8,4 cm
9,054 m	84,4 cm	115,6 cm	14,4 cm
10,061 m	95,1 cm	104,9 cm	25,1 cm
11,067 m	104 cm	96 cm	34 cm
12,073 m	114,8 cm	85,2 cm	44,8 cm
13,079 m	126,4 cm	73,6 cm	56,4 cm
14,085 m	137,9 cm	62,1 cm	67,9 cm
15,076 m	151,6 cm	48,4 cm	81,6 cm

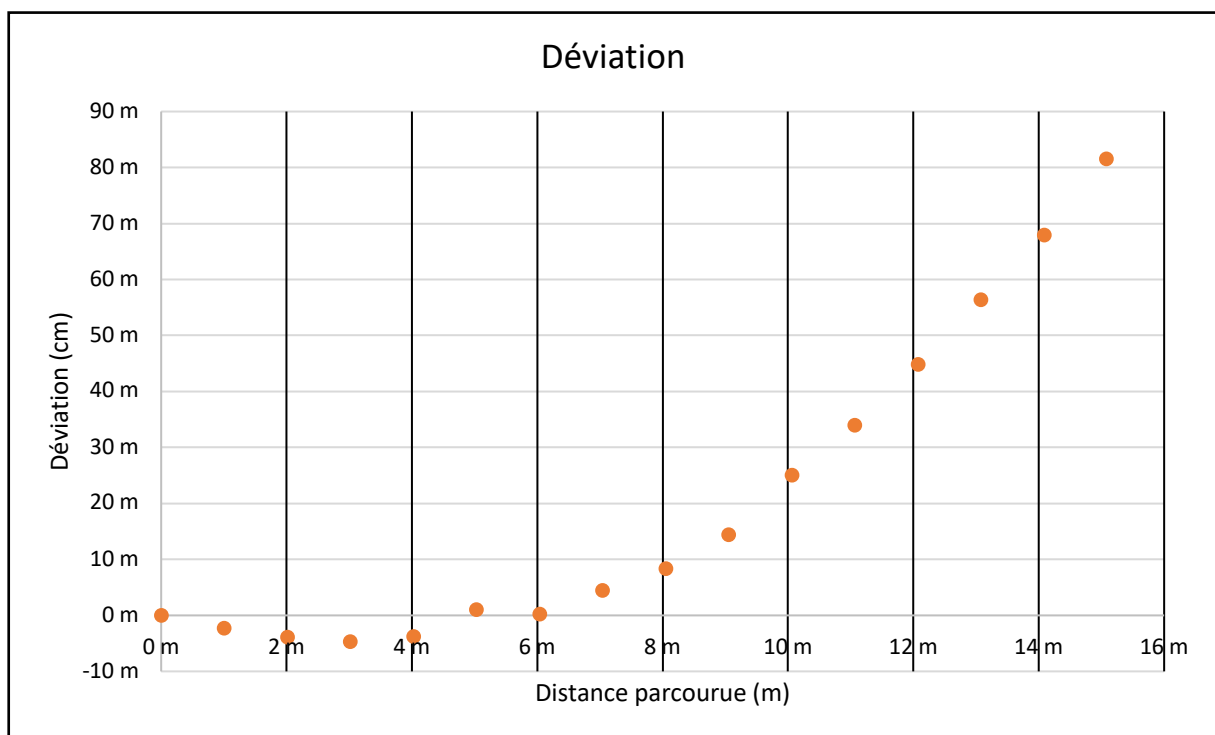


Figure 2 : Courbe de la déviation obtenue au test 2

Test parcours couloir avec arrêt tous les mètres (test 3)			
Distance parcourue (m)	Distance par rapport au mur de droite (cm)	Distance par rapport au mur de gauche (cm)	Déviaton
0 m	70 cm	130 cm	0 cm
1,005 m	70 cm	130 cm	0 cm
2,011 m	70 cm	130 cm	0 cm
3,016 m	71,5 cm	128,5 cm	1,5 cm
4,023 m	73,4 cm	126,6 cm	3,4 cm
5,03 m	80,6 cm	119,4 cm	10,6 cm
6,036 m	82,9 cm	117,1 cm	12,9 cm
7,042 m	89,3 cm	110,7 cm	19,3 cm
8,048 m	92,3 cm	107,7 cm	22,3 cm
9,054 m	104,3 cm	95,7 cm	34,3 cm
10,06 m	120 cm	80 cm	50 cm
11,066 m	134 cm	66 cm	64 cm
12,072 m	148 cm	52 cm	78 cm
12,5 m	155,3 cm	44,7 cm	85,3 cm

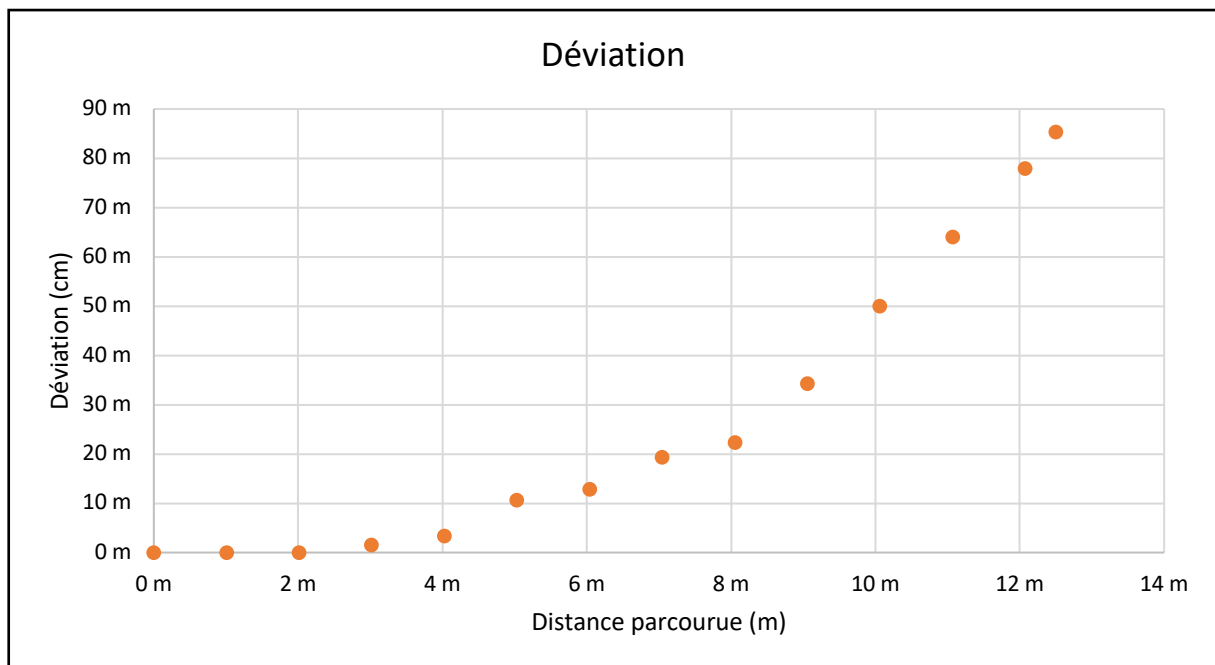


Figure 3 : Courbe de la déviaton obtenue au test 3

On constate dans les tableaux que la distance parcourue à chaque essai n'atteint pas les 20 mètres prévus. En effet, le robot a, à chaque fois, touché le mur avant 20 m. On arrête donc les relevés au moment où le robot touche le mur. Par la suite, on calcule ensuite la moyenne de la distance maximale parcourue et de la déviaton.

Moyenne Distance max	Moyenne Déviaton
13,15866667 m	83,83333333 cm

En observant les courbes, on constate qu'entre 0 et 2 m, la déviation est quasi-nulle. Mais à partir de 3 m, la déviation augmente. Dans les trois essais, on peut voir qu'à partir d'une certaine distance, l'augmentation de la déviation suit un profil linéaire.

### 5) Tests parcours carré

Après la lecture de plusieurs articles scientifiques sur les erreurs d'odométrie en robotique, j'ai été amené à réaliser un test qui consistait à faire parcourir au robot un parcours carré. Dans ce test, le robot part d'un coin du carré et doit avancer en essayant de rester le plus possible sur le carré. On mesure au cours du parcours l'angle de rotation du robot à chaque fois qu'il arrive dans un coin du carré. On regarde si le robot tourne bien de  $90^\circ$ . De plus, à la fin du parcours on compare la position finale du robot avec sa position initiale et on note la différence. On réalise ce test dans deux situations différentes. Dans un premier temps, le robot va parcourir le carré dans le sens horaire et dans un deuxième temps, il va parcourir le carré dans le sens antihoraire.

Pour réaliser ce test, j'ai utilisé une salle de classe complètement vidée de tous les meubles et j'ai tracé au sol un carré de 4 m de côté avec du ruban adhésif.



*Carré de 4,0 m de côté*

On place le robot dans un coin du carré et on va lancer un programme qui va le faire avancer en carré. Il doit avancer de 4 m puis tourner de  $90^\circ$ . On répète ces commandes 4 fois pour qu'à la fin du parcours le robot revienne à sa position de départ.

Pour mesurer l'angle réel de la rotation du robot dans chacun des coins, j'ai placé mon téléphone sur le robot avec l'application Boussole. J'ai noté la valeur que me donne la boussole quand le robot est à sa position initiale, puis je regarde la valeur de la boussole après que la rotation du robot. En soustrayant la nouvelle valeur à l'ancienne je peux savoir de quel angle le robot a tourné. On répète l'opération sur chaque arête du carré. On regarde la valeur

avant que le robot tourne et on la soustrait à la valeur obtenue après la rotation. Ainsi on peut avoir l'angle réel pour chaque coin.



Robot avec le téléphone et l'application boussole pour mesurer l'angle

### a) Parcours sens horaire

Dans ce scénario, le robot doit parcourir le carré dans le sens horaire. On effectue ce parcours 10 fois en notant à chaque fois les angles à chaque rotation. A la fin on calcule la moyenne de chacun des angles. On obtient le tableau suivant :

Rotation	Tour 1	Tour 2	Tour 3	Tour 4	Tour 5	Tour 6	Tour 7	Tour 8	Tour 9	Tour 10	Moyenne	Ecart Type	Variance
1	61	84	89	89	91	91	92	92	92	91	90,11111111	9,51957049	90,62222222
2	95	92	92	94	96	93	94	93	94	93	93,6	1,26491106	1,6
3	93	93	93	92	92	93	91	93	92	92	92,4	0,6992059	0,48888889
4	88	89	89	87	89	88	89	88	88	89	88,4	0,6992059	0,48888889

En se basant sur ces données on peut dessiner le schéma suivant :

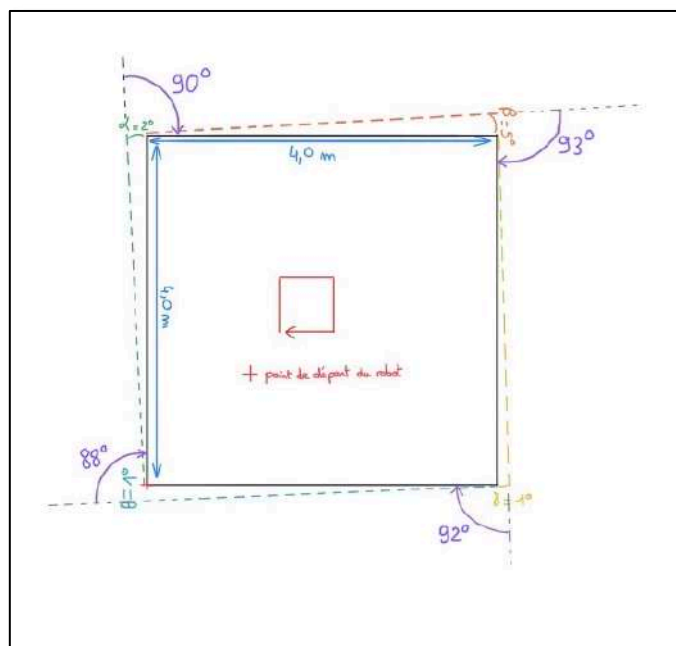


Schéma du parcours dans le sens horaire

Sur ce schéma, les traits en pointillés correspondent au parcours réel du robot. On voit qu'il ne suit pas parfaitement le carré. Il a toujours tendance à dévier un peu vers la gauche.

### b) Parcours sens antihoraire

On va maintenant effectuer le parcours du carré dans le sens antihoraire. Comme pour le sens horaire, on va effectuer le parcours 10 fois et on va calculer la moyenne des quatre angles. On obtient le tableau suivant :

Rotation	Tour 1	Tour 2	Tour 3	Tour 4	Tour 5	Tour 6	Tour 7	Tour 8	Tour 9	Tour 10	Moyenne	Différence	Ecart type	Variance
1	89	88	87	88	87	89	89	88	88	88	88,11111111	1,88888889	0,73785479	0,54444444
2	81	88	89	91	90	89	90	90	90	89	88,66666667	1,33333333	2,83039063	8,01111111
3	94	97	98	100	99	100	99	100	98	99	98,33333333	-8,33333333	1,83787317	3,37777778
4	82	82	80	81	81	82	81	80	81	81	81,11111111	8,88888889	0,73785479	0,54444444

A partir de ces données on peut donc dessiner le schéma suivant :

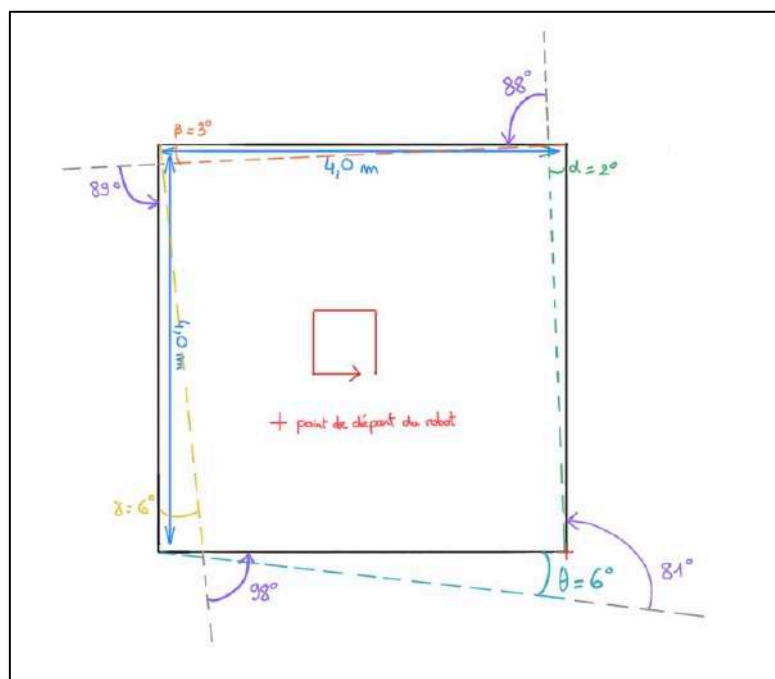


Schéma du parcours dans le sens antihoraire

On peut voir que dans le sens antihoraire le robot a plus de mal à effectuer les rotations de 90°. Ainsi on peut voir que le robot a plus de problème à effectuer des rotations vers la gauche. Lors de ces tests on a vu que le robot a toujours une tendance à dévier vers la gauche. Cependant dans les faits, l'angle de la déviation est relativement petit. Donc sur des petites distances, et au vu de la dimension assez importante du robot, la déviation n'impacte pas de façon trop importante la précision du robot. Dans notre cas les mesures qu'on va effectuer vont se faire dans une seule salle de classe. Ainsi la légère déviation du robot ne va pas impacter de façon significative les mesures.



### III. Mesures pour la localisation indoor

La prise des mesures pour la localisation indoor est la deuxième partie majeure de mon stage. Pour pouvoir faire le fingerprinting il faut mesurer à différents points de la pièce le RSSI (Received Signal Strength Indication), c'est-à-dire la puissance du signal émis. Dans notre cas, on va mesurer la puissance des signaux émis par des balises RFID. Avant d'effectuer les mesures, il faut d'abord définir comment on va les faire. Pour cela il nous faut définir les outils à utiliser ainsi que les différents scénarios de mesures. Concernant les outils, on va utiliser des balises RFID que l'on va coller sur les murs de la salle de classe. On va ensuite placer le lecteur RFID qui va mesurer le RSSI sur le robot en le reliant à l'ordinateur qui est aussi sur le robot. Le lecteur a dû être surélevé avec des cartons afin que les antennes du lecteur soient à la même hauteur que les balises RFID qui sont collées au mur.



Lecteur RFID



Balise RFID

*Lecteur RFID monté sur le robot*

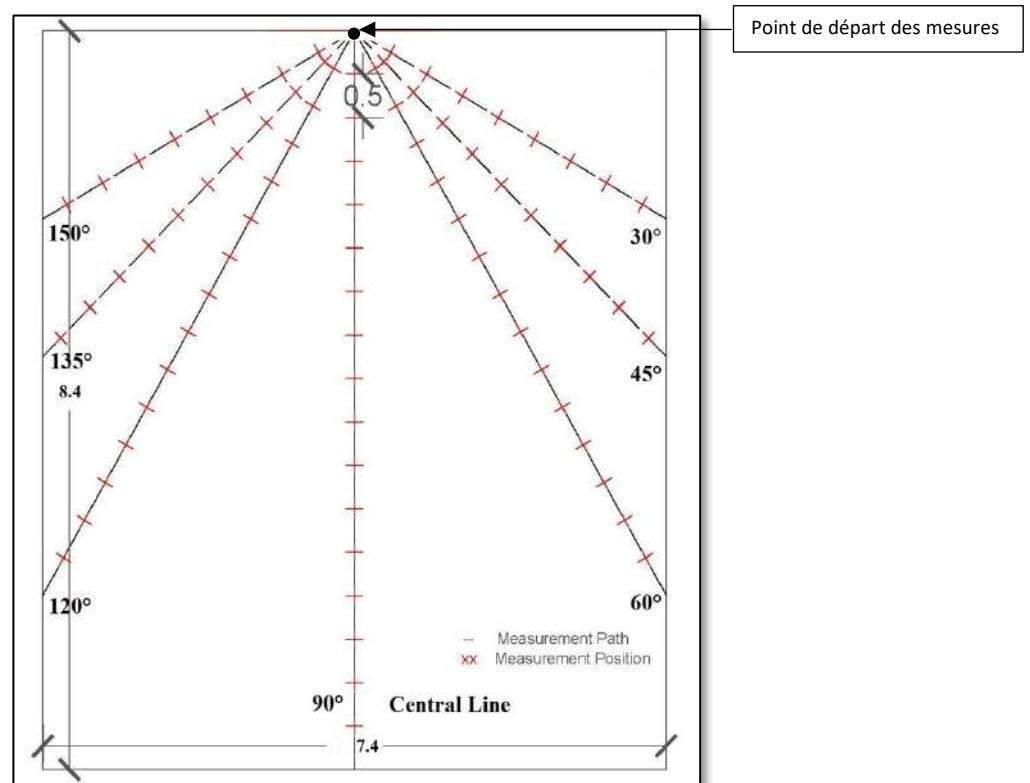
La connexion avec l'ordinateur m'a posé un problème pendant mon stage. En effet, le logiciel qui permet de mesurer le RSSI avec le lecteur ne fonctionne que sur Windows, cependant les programmes qui permettent de contrôler les déplacements du robot tournent sur Linux. Ainsi il a fallu que je lance les deux systèmes en simultané sur mon ordinateur. Cela avait pour conséquence de consommer beaucoup de batterie. Ainsi, il fallait souvent faire des pauses dans les mesures pour que je puisse recharger mon ordinateur.



### A) Les différents scénarios de mesures

Pour effectuer les mesures, j'avais besoin d'une salle entière complètement vide. Cependant la salle ne pouvant être réservée que pendant un mois, il a fallu se presser pour effectuer les différentes mesures. Ainsi, pour les différents scénarios, ils ont été décidés avec Mme. COLIN mais aussi avec M. Elias HATEM qui est un doctorant qui prépare sa thèse à Efrei Paris sur le thème de la localisation indoor. Dans le cadre de sa thèse, il a déjà été amené à effectuer des mesures pour faire du fingerprinting. Cependant, les mesures qu'il a effectuées étaient des mesures sans le robot. Ainsi, dans mon stage, je vais effectuer les mêmes scénarios de mesures que lui mais avec le robot. Ainsi on va pouvoir comparer les valeurs obtenues avec et sans le robot pour voir si le robot a un impact sur les mesures.

Pour les mesures, on va définir plusieurs trajectoires en lignes droites à différents angles par rapport au mur au fond de la salle. Sur chacune de ces trajectoires, le robot va avancer et faire des pauses à des distances données pour effectuer la mesure du RSSI à la position correspondante.

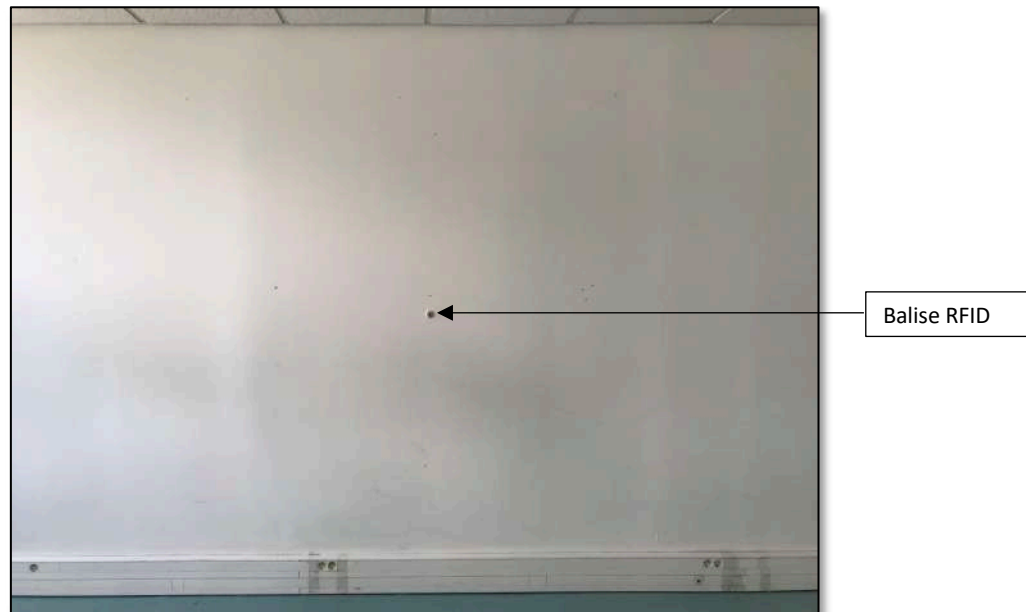


*Trajectoires utilisées par Elias HATEM pour sa thèse*

Le schéma ci-dessus représente les 7 trajectoires choisies et utilisées par Elias HATEM pour ses mesures sans le robot. Ce sont donc ces trajectoires que l'on va choisir en premier pour mes mesures avec le robot. Sur ces 7 trajectoires le robot va s'arrêter tous les 0,5 m pour effectuer les mesures du RSSI.

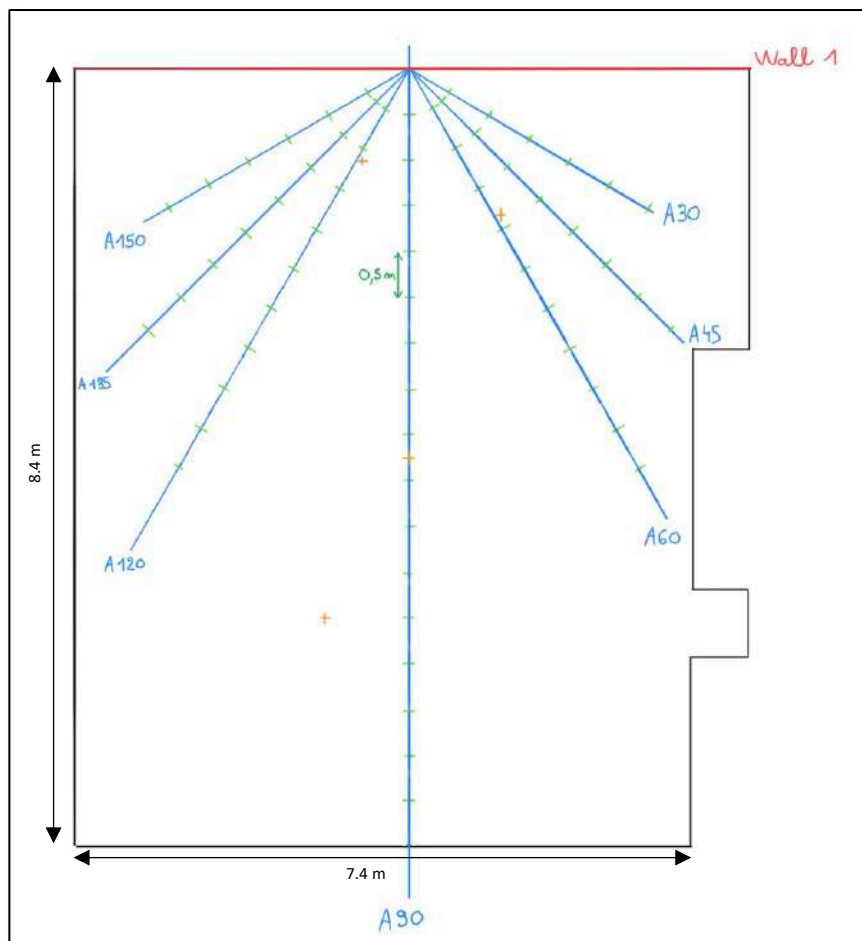
### 1) Scénario 1 : Single Tag Center

Le premier scénario consiste à placer une seule balise RFID au centre du mur au fond de la salle. Le robot va avancer sur les différentes trajectoires et, tous les 0,5 m, il va faire les mesures du RSSI. Afin de s'assurer de la valeur du RSSI, le lecteur RFID va effectuer 200 acquisitions du RSSI à chaque position.



Une seule balise RFID au centre du mur

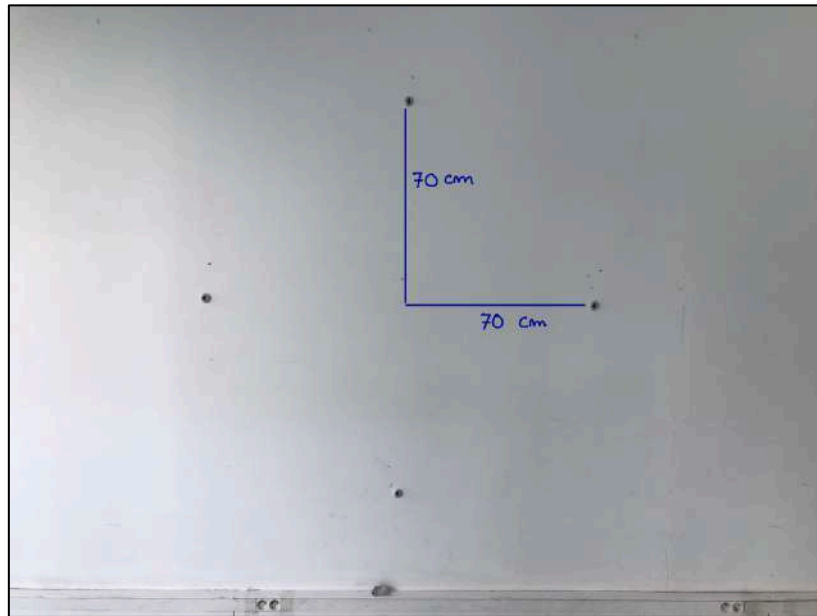
En plus des 7 trajectoires définies précédemment, on va également choisir 4 positions dans la salle pour lesquelles on va également mesurer le RSSI.



Sur ce schéma les croix orange représentent les 4 positions supplémentaires.

## 2) Scénario 2 : Constellation mur 1

Dans ce scénario on remplace la balise seule par 4 balises RFID. Les 4 balises sont disposées sous la forme d'une constellation de rayon  $R = 70$  cm. On refait les mesures sur les 7 mêmes trajectoires ainsi que sur les 4 positions choisies précédemment.



*Constellation de 4 balises de rayon  $R = 70$  cm*

## 3) Scénario 3 : Localisation

Dans ce scénario, on va une balise au centre de chacun des 4 murs de la salle. On va également définir de nouvelles trajectoires. On ne prend plus les trajectoires à  $30^\circ$ ,  $45^\circ$ ,  $135^\circ$  et  $150^\circ$ . A la place on choisit 3 nouvelles trajectoires. Pour ces trajectoires, le point de départ du robot est maintenant au niveau du mur à l'avant de la salle et le robot avance en direction du mur du fond. Ces 3 trajectoires correspondent aux angles  $90^\circ$ ,  $60^\circ$  et  $120^\circ$ . On appelle ces trajectoires A90', A60' et A120' pour les différencier des trajectoires originales A90, A60 et A120. En plus de ces nouvelles trajectoires, il a été décidé que le robot ne s'arrêterait plus tous les 0,5 m mais tous les 0,7 m maintenant. Enfin, en plus des 4 positions choisies lors des scénarios précédents, on ajoute 4 nouvelles positions supplémentaires.

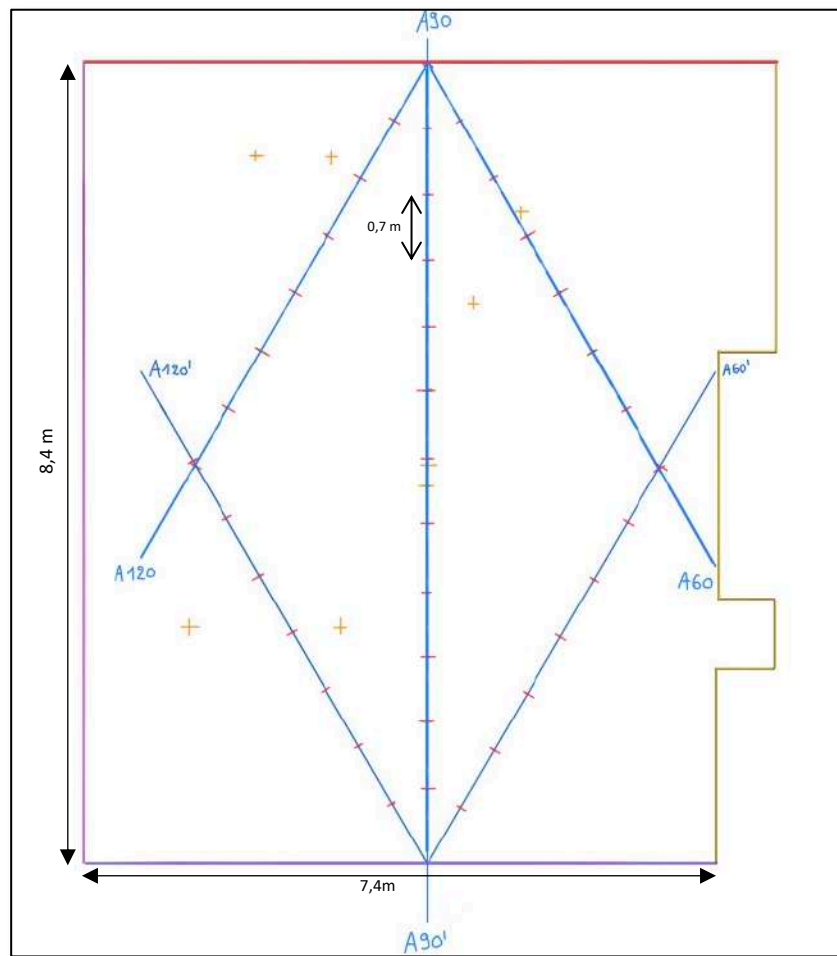
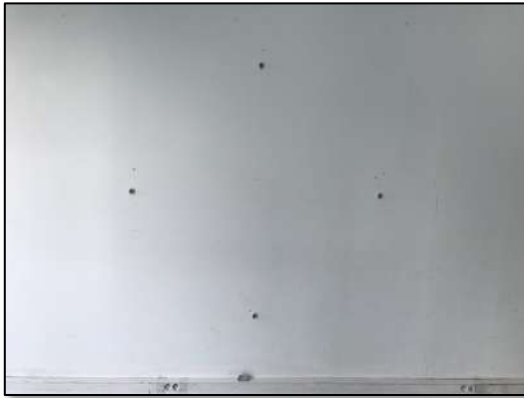
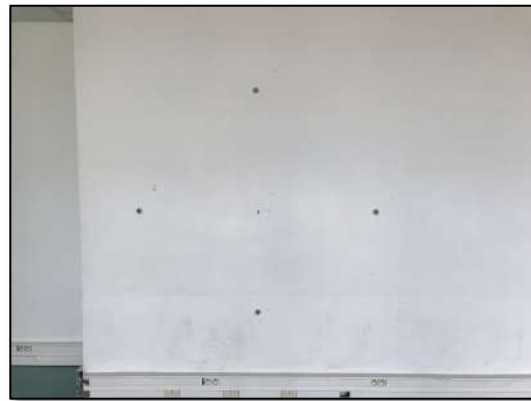
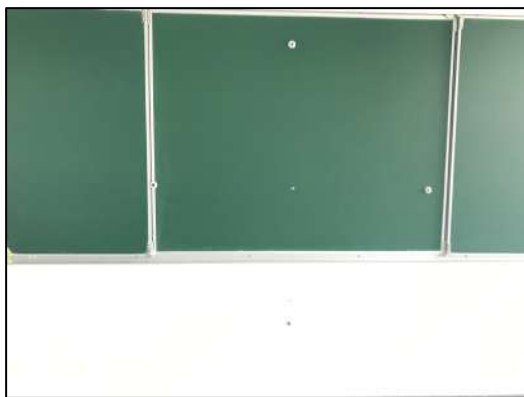
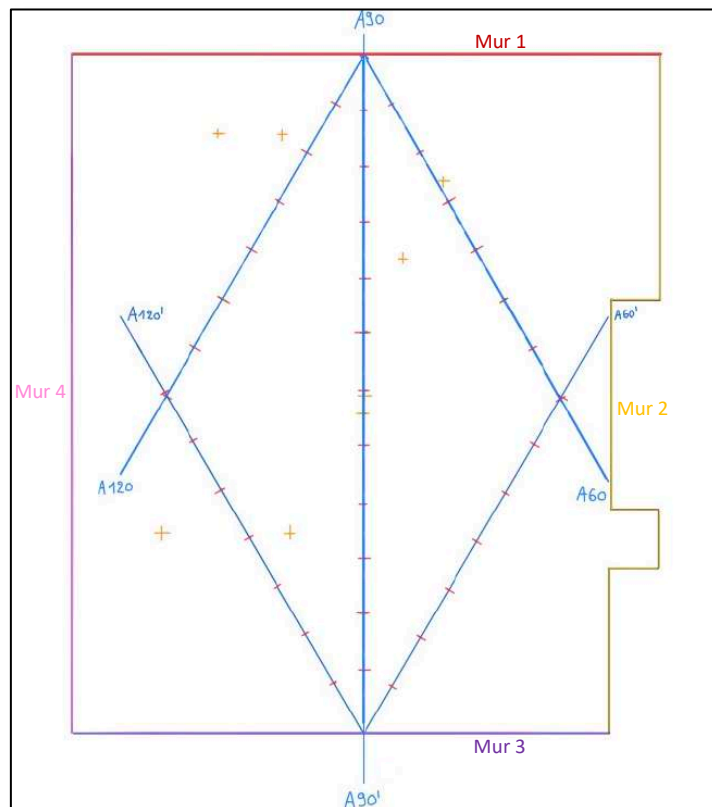


Schéma des nouvelles trajectoires choisies

Les croix orange représentent les 8 positions choisies.

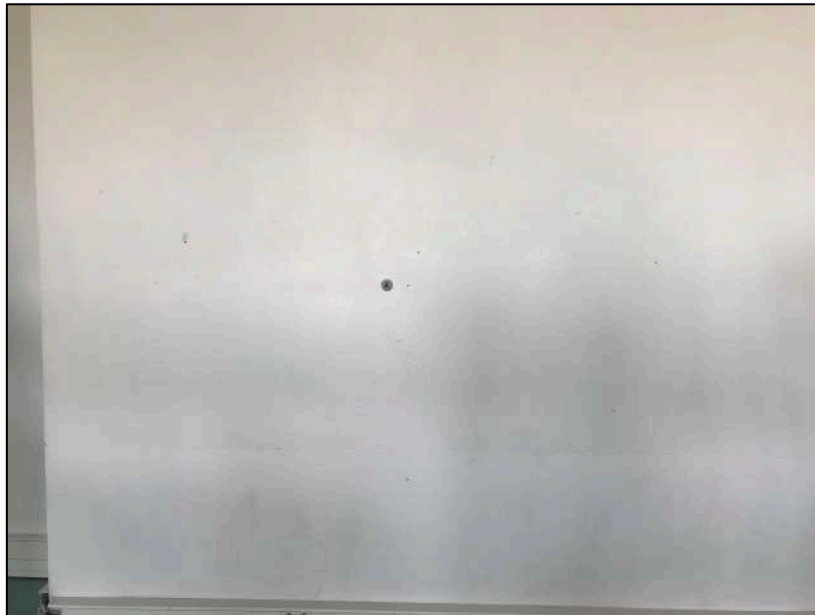
#### 4) Scénario 4 : Constellation sur chaque mur

Dans ce scénario on va placer une constellation de 4 balises RFID sur chacun des 4 murs de la salle. Cependant, on place la constellation sur un mur à la fois. La constellation a toujours un rayon de 70 cm comme dans le scénario 2. Tout d'abord on va placer la constellation sur le mur 1, on va faire les mesures sur toutes les trajectoires et à tous les points puis on va déplacer la constellation sur le mur 2 et on va faire les mesures. On va répéter l'opération pour les 4 murs. On garde ici les mêmes trajectoires que le scénario précédent. On garde aussi les 8 mêmes positions.

*Constellation sur le mur 1**Constellation sur le mur 2**Constellation sur le mur 3**Constellation sur le mur 4*

### 5) Scénario 5 : Single Tag Wall 2

Dans ce scénario on va placer une seule balise RFID au centre du mur 2. A la différence du scénario 1, on va utiliser les trajectoires A90, A60, A120, A90', A60' et A120' des scénarios 3 et 4. Le robot va donc faire les mesures tous les 0,7 m. On va également utiliser les 8 positions choisies.



*Une balise RFID au centre du mur 2*

### 6) Scénario 6 : Single Tag Wall 4

Ce scénario est identique au scénario 5 sauf que la balise est placée au centre du mur 4 au lieu du mur 2.



*Une balise RFID au centre du mur 4*

### 7) Scénario 7 : Constellation mur 4

Ce scénario est identique au scénario 6 mais on met une constellation de 4 balises de rayon 70 cm à la place d'une seule balise.



*Constellation de 4 balises sur le mur 4*



## Conclusion générale

Ce stage s'est déroulé en trois grandes parties. La première concernait la prise en main du robot. Dans cette partie j'ai dû appréhender de nouvelles technologies tel que ROS afin de comprendre comment fonctionne le robot qui m'a été confié. La première prise en main du robot s'est avérée assez compliquée du fait que l'entreprise qui fabrique le robot, MobileRobots, n'existe plus. Ainsi, il a été difficile de trouver les outils nécessaires à la programmation du robot, tel que ARIA qui est la base de tous les programmes du Pioneer 3-DX. Quand j'ai fini par trouver une archive du site internet du constructeur qui contenait encore les différents outils à télécharger, ça m'a permis d'avancer et j'ai pu contrôler le robot. Le stage m'a également permis de m'améliorer en C++. En effet, le C++ était un langage de programmation que j'utilisais très peu avant le stage. Ainsi j'ai pu, avec le stage, gagner en aisance et aujourd'hui mon niveau en C++ est bien meilleur qu'avant.

La deuxième grande partie du stage concernait le problème de déviation du robot. Ce problème a vraiment été un élément inattendu durant le stage. La recherche de la cause de la déviation a pris vraiment beaucoup de temps pendant le stage, environ 2 mois, et il a été compliqué de définir les tests à effectuer pour identifier le problème. En effet, je ne savais pas si les tests que je faisais allaient me donner des résultats utiles. J'ai, pendant ces 2 mois, lu pas mal d'articles scientifiques afin d'essayer de comprendre les raisons de cette déviation. Cela m'a permis d'acquérir des connaissances sur des sujets qui m'étaient inconnus. Pendant cette période, j'ai également pu réaliser le côté négatif de la recherche. En effet, quand on fait de la recherche, quand on a un problème qui nous bloque, on prend beaucoup de temps pour trouver une solution car peu de personnes ont travaillé sur le sujet et parfois on ne la trouve tout simplement pas. De plus, pendant ce temps, le projet n'avance pas car toutes les étapes dépendent les unes des autres. Ce problème de déviation du robot m'a toutefois permis d'acquérir une grande capacité d'adaptation qui me sera utile dans les futurs projets que j'aurai à mener.

Enfin, la dernière partie de ce stage concernait la prise de mesures avec le robot. Cette étape a dû se faire un peu dans la précipitation du fait que la salle dans laquelle je devais faire les mesures ne pouvait être disponible que pendant un mois. Cela m'a contraint à faire quelques compromis afin que les mesures puissent se faire dans les temps. Premièrement l'utilisation de deux systèmes, Windows et Linux, pour la prise de mesures et la commande du robot m'a forcé à trouver un moyen d'effectuer les deux tâches en simultané. Ainsi, je contrôlais la prise de mesure avec ma tablette et la commande du robot avec mon téléphone. Si j'avais eu plus de temps j'aurais certainement cherché une solution afin que les tâches s'exécutent automatiquement de façon synchronisée. Enfin, à cause de la contrainte de temps, on a dû limiter le nombre de scénarios à effectuer pour se concentrer sur ceux déjà effectués par Elias HATEM sans le robot. Il aurait été intéressant d'essayer d'autres scénarios dans lesquels le robot se déplace plus librement et pas seulement en ligne droite.

En conclusion, ce stage m'a permis de découvrir le monde de la recherche avec ses bons côtés et ses limites. Le projet qui m'a été confié a exigé que je fasse preuve de beaucoup d'autonomie et m'a permis d'acquérir de nombreuses connaissances et compétences qui me seront très utiles pour ma future vie professionnelle.