



TER

---

## Flottille de Robots

---

*Membres du groupe :*

Victor JUNG  
Steve MALALEL  
Michel MARMONE-MARINI  
Aymeric VALDENAIRE

*Encadrants :*

Marie PELLEAU  
Enrico FORMENTI

17 juin 2019

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>État de l'art</b>	<b>2</b>
2.1	Matériel . . . . .	2
2.2	Algorithmes . . . . .	5
<b>3</b>	<b>Travail effectué</b>	<b>6</b>
3.1	Recherche . . . . .	6
3.1.1	Comment construire le robot ? . . . . .	6
3.1.2	Comment cartographier un espace ? . . . . .	6
3.1.3	Comment faire communiquer les composants ? . . . . .	7
3.2	Conception et fabrication du robot . . . . .	7
3.2.1	Modélisation et impression 3D . . . . .	8
3.2.2	Matériel et assemblage . . . . .	10
3.3	Méthode d'exploration . . . . .	11
3.3.1	Calibrage du robot . . . . .	11
3.3.2	Algorithme . . . . .	11
3.4	Cartographie et communication . . . . .	13
3.4.1	Cartographie . . . . .	13
3.4.2	Communication des données . . . . .	14
<b>4</b>	<b>Gestion de projet</b>	<b>15</b>
4.1	Logistique . . . . .	15
4.2	Répartition des tâches . . . . .	16
<b>5</b>	<b>Difficultés rencontrées</b>	<b>16</b>
5.1	Mesure des capteurs . . . . .	16
5.2	Positionnement . . . . .	16
5.3	Mémoire . . . . .	17
5.4	Communication WiFi . . . . .	17
<b>6</b>	<b>Résultats</b>	<b>18</b>
6.1	Déplacement et comportement du robot . . . . .	18
6.2	Cartographie obtenue . . . . .	18
<b>7</b>	<b>Conclusion</b>	<b>19</b>
<b>8</b>	<b>Perspectives et réflexions personnelles</b>	<b>19</b>
8.1	Algorithme d'exploration . . . . .	20
8.2	Système de positionnement . . . . .	20
<b>9</b>	<b>Annexes</b>	<b>20</b>

# 1 Introduction

Ce TER a pour but de créer une flottille de robot capable d'explorer et de cartographier un espace en 2D. Nous nous basons sur une première version d'un robot capable de détecter des obstacles et de calculer la distance qui les sépare du robot à l'aide de six capteurs à ultrason. Notre objectif est donc d'améliorer ce robot afin qu'il soit capable de longer des obstacles, d'enregistrer les données des capteurs et de les communiquer afin de pouvoir réaliser une carte de l'espace exploré. Notre cahier des charges est donc le suivant :

- Ajout d'un dispositif de communication à distance
- Mise en place d'un protocole d'échange d'information
- Connaître la position des robots dans un espace 2D
- Conception d'un outil de cartographie
- Ajout d'autres robots pour arriver à avoir une flottille

Nous avons décidé de donner la priorité à la confection du robot principal de la flottille et d'essayer de cartographier un espace avec celui-ci, puis d'utiliser plusieurs robots qui auraient pu l'assister dans sa cartographie. Par manque de temps nous avons donc décidé d'utiliser seulement le robot principal de la flottille. Cependant, l'architecture du serveur pour les communications a été conçu de manière à pouvoir gérer plusieurs robots à la fois.

Notre groupe est composé de JUNG Victor, MALALEL Steve, MARMONE-MARINI Michel, VALDENAIRe Aymeric, encadré par Madame PELLEAU et Monsieur FORMENTI.

Dans ce rapport, nous commencerons par présenter notre point de départ, puis nous aborderons les différents choix que nous avons été amenés à faire concernant la conception du robot, la méthode d'exploration, la communication entre les différents composants et enfin la cartographie d'un espace. Pour finir, nous parlerons des perspectives d'amélioration concernant notre robot.

## 2 État de l'art

Pour ce projet, nous sommes partis du TER "Robot avec détection d'obstacles" réalisé l'année dernière (2018) par BENSOUSSAN Chloé, BOUKOU Grace, GRÉAU Alexandre et WILHELM Andreina (Les images contenues dans cette partie sont toutes récupérées de leur rapport).

### 2.1 Matériel

Le matériel mis à disposition comprenait deux robots, un maître et un suiveur, ainsi que plusieurs pièces de rechange. Nous nous sommes seulement intéressés au

robot maître (car seul celui-là était fonctionnel), celui-ci était équipé de six capteurs placés en demi-cercle, d'une carte Arduino équipée d'un shield, de deux moteurs et d'une roue libre ainsi que d'une alimentation.

**Capteurs** Les capteurs à ultrasons ont chacun un émetteur et un récepteur permettant de détecter un objet devant eux (cependant moins l'objet est perpendiculaire à la direction du signal émis par un capteur, moins ce dernier sera apte à le détecter). Ces capteurs peuvent aussi calculer automatiquement la distance les séparant de l'objet qu'ils détectent, ce qui nous sera utile pour dessiner une carte.

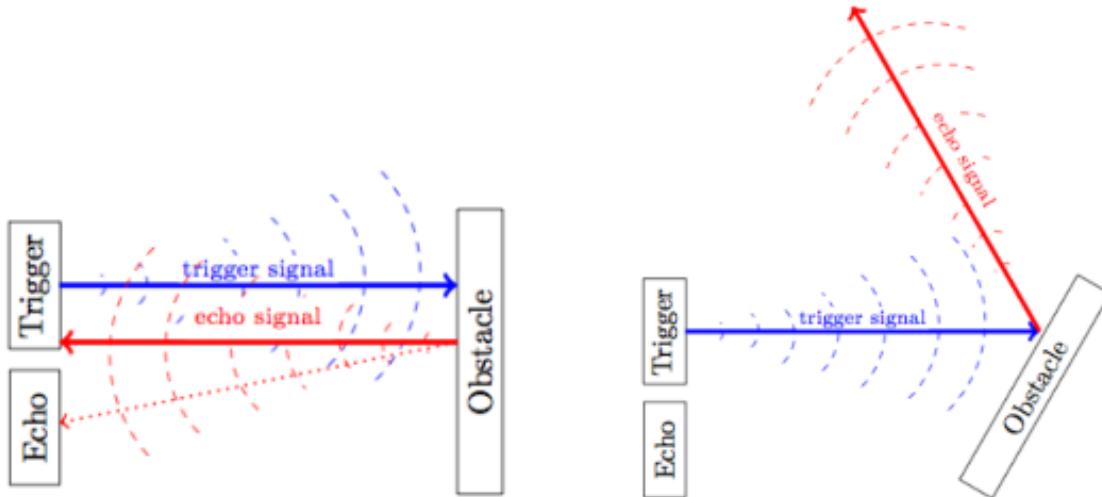


FIGURE 1 - Perte du signal selon l'orientation de l'obstacle

**Moteurs** Les moteurs avec roues ont la possibilité de tourner dans les deux sens et d'avoir une vitesse variable. Ce dernier point nous sera utile pour permettre au robot d'avancer tout droit correctement (dû à certaines différences de puissance entre les moteurs, les roues tournent à des vitesses différentes alors que la valeur représentant la vitesse dans le code est la même pour toutes les roues).

**Carte Arduino** Afin de récupérer les informations des capteurs et de donner des instructions aux moteurs, le robot était équipé d'une carte Arduino Uno et d'un shield. Le tout était connecté de la manière suivante :

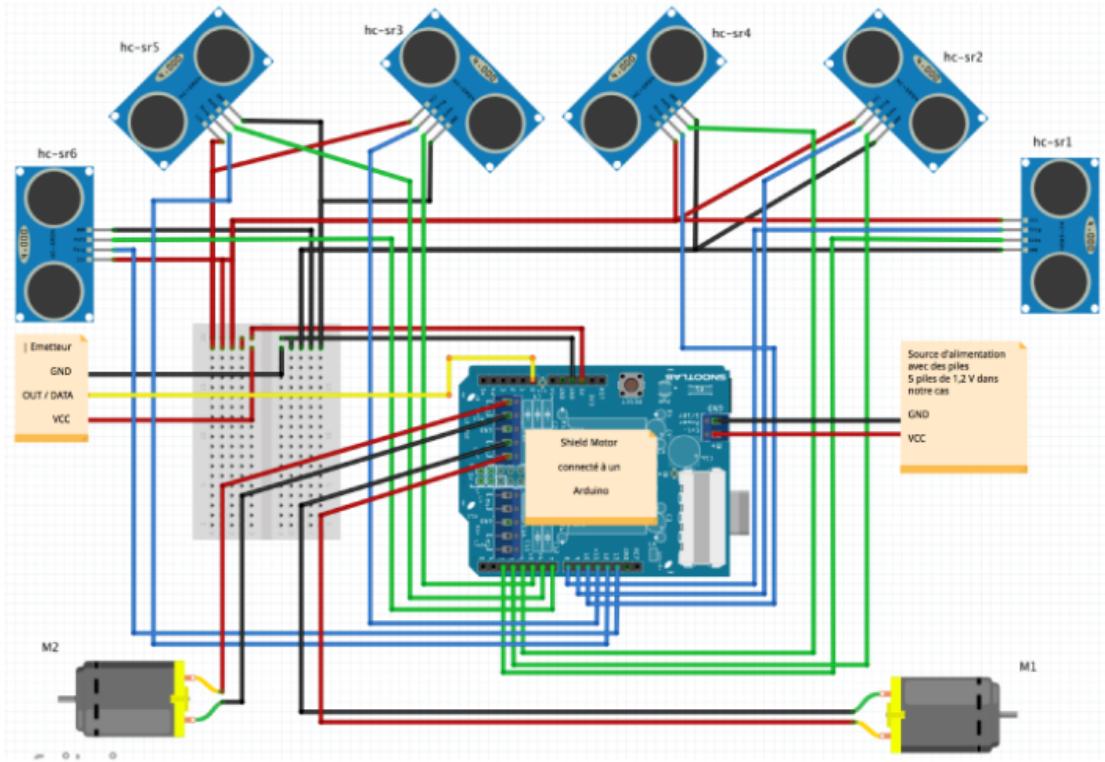


FIGURE 2 - Branchements

**Alimentation** Pour alimenter tout le matériel, le robot était également équipé d'un support imprimé en 3D pouvant accueillir 5 piles. Le courant passait dès que toutes les piles étaient insérées dedans et le seul moyen de l'arrêter était d'en enlever une ou de débrancher le support du robot.



FIGURE 3 - Support pour piles

**Châssis** Le châssis permettait de relier tout les précédents éléments et de placer les capteurs d'une certaine manière, leur disposition couvrait une surface optimale selon les recherches effectuées par les concepteurs de ce robot. S'appuyant sur leurs résultats, peu de modifications ont été apportées sur ce placement.

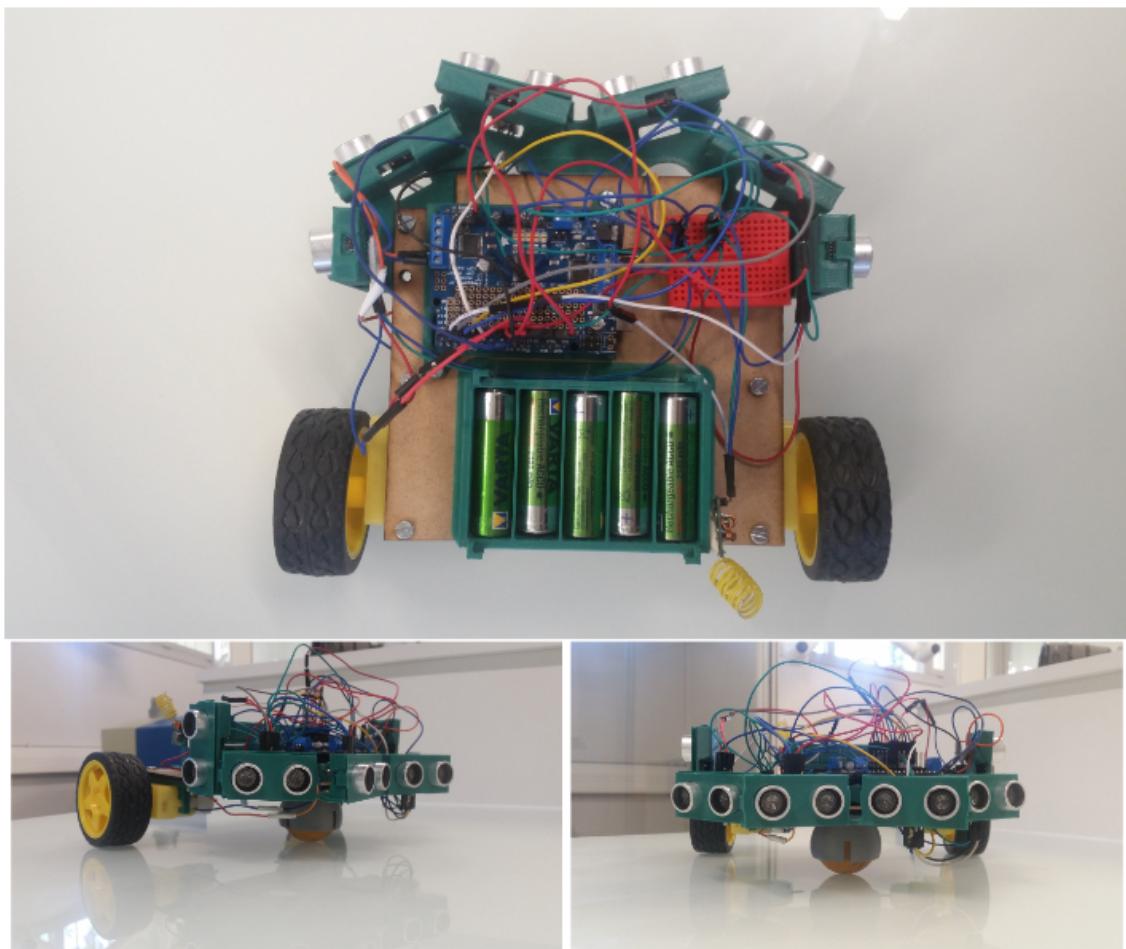


FIGURE 4 - Le robot totalement monté

## 2.2 Algorithmes

Avec tout ce matériel étaient fournis quatre algorithmes permettant :

- d'avancer en évitant tous les obstacles
- de longer un obstacle
- de suivre le robot maître pour le robot suiveur
- de communiquer entre les deux robots

Mais seulement un seul va nous intéresser :

**longe obstacle** De son nom complet "scrappy\_algo2\_longe\_obstacle.ino", cet algorithme permet de longer et donc de faire le tour d'un obstacle ce qui sera intéressant pour numériser son contour dans l'environnement à cartographier. Cependant il permet de longer uniquement les coins ou les contours circulaires, il était donc nécessaire de le changer pour avoir un plus large panel d'objet à cartographier.

### 3 Travail effectué

#### 3.1 Recherche

Avant de commencer à travailler sur les différents aspects du TER (robot, carte, communication), nous sommes passés par une phase de recherche sur comment faire, avec les moyens à notre disposition, pour répondre au mieux aux attentes du sujet. Nous allons donc présenter dans cette partie la phase de réflexion pour chaque composant.

##### 3.1.1 Comment construire le robot ?

Afin de répondre à cette question, nous avons dû d'abord établir les pré-requis ou caractéristiques de notre robot. Il nous fallait un robot qui puisse respecter les contraintes suivantes :

- Un déplacement stable, c'est-à-dire qu'il ne dérape pas ou que les roues ne se bloquent pas
- Être capable de traquer sa position ( $x,y$ ) et sa direction (angle)
- Être capable de détecter les obstacles devant lui et sur chaque côté
- Pouvoir envoyer des informations sur un serveur à distance
- Pouvoir longer un obstacle de forme quelconque (rectangle, ovale, etc...)

Par rapport au robot qui nous a été fourni, seule la contrainte de détection d'obstacle était pleinement satisfaite. Le robot était effectivement incapable de longer un obstacle quelconque (l'algorithme ne fonctionnait que dans certains cas précis), de traquer sa position (aucune notion de position) et pas de dispositif d'échange d'information à distance. De plus, le robot dérapait assez souvent et les rotations n'étaient pas précises à cause de la roue du milieu. Nous avons donc dû construire un nouveau robot permettant de satisfaire toutes les contraintes que nous nous sommes imposées, dont les étapes seront détaillées plus tard dans le rapport.

##### 3.1.2 Comment cartographier un espace ?

L'idée pour cartographier un espace est de simplement ajouter des points sur un plan 2D. Cependant, lors des mesures effectuées par le robot, nous disposons seulement de la distance de l'obstacle par rapport au robot. Il nous fallait donc un outil capable d'ajouter un point à un plan 2D en disposant d'un point de référence (position du robot), de la distance par rapport à ce point (la mesure) ainsi que l'angle (angle du robot par rapport aux axes des abscisses et ordonnées). Ceci n'est pas vraiment difficile, étant donné qu'il s'agit d'une simple opération trigonométrique.

Un des plus grands problèmes est de savoir à quel moment la cartographie est terminée, ou même à plus petite échelle : quand finit-on de longer un objet ? Dans un premier temps, nous avions pensé à faire un modèle de graphe, où un objet serait entièrement contourné si nous pouvions obtenir une composante fortement connexe

à partir d'un point de celui-ci. Deux soucis majeurs sont alors apparus, dont chacun est suffisant seul pour abandonner cette idée. Premièrement, il faut impérativement que deux points côté à côté soient connectés entre eux, ce qui est difficile étant donné que notre matériel n'est pas assez précis et que beaucoup de facteurs environnementaux peuvent influencer la précision, comme par exemple le type de sol sur lequel le robot se déplace (et cela dépend également de la fréquence d'échantillonnage des distances). Ainsi, nous pouvions ne jamais arriver à une composante fortement connexe. Deuxièmement, la mémoire est insuffisante dans le cas d'un graphe avec énormément de noeuds.

Nous nous sommes ensuite tournés vers une solution plus simple : lors du début du parcours d'un objet, on place des "points d'ancrage" qui vont nous servir de repère. Si on retombe sur ces points après avoir parcouru une distance minimum, on sait que l'on vient de faire le tour de l'objet, que l'on peut ainsi quitter.

### 3.1.3 Comment faire communiquer les composants ?

Notre idée de base était d'utiliser un module WiFi par robot pour que chacun d'eux soient connectés au serveur et qu'ils envoient les informations nécessaires au traçage de la carte. De plus le robot principal aurait donné des ordres de déplacement aux petits robots pour qu'ils visitent d'autres parties de l'espace à explorer, ce qui aurait permis de cartographier plus rapidement. Pour certaines raisons expliquées précédemment nous avons finalement décidé d'utiliser seulement un robot (le robot principal de la flottille). Par conséquent, un seul module WiFi connecté avec la carte Arduino du robot suffit pour permettre l'échange d'informations entre le robot (client) et le serveur. Plusieurs envois d'informations lors de l'exploration du robot sont donc nécessaires. Tout d'abord l'envoi des distances calculées par les capteurs ultrason ainsi que la position du robot de la carte Arduino au module WiFi en communication série. Ensuite le module WiFi envoie au serveur ces informations pour permettre le traçage de la carte (le module WiFi et le serveur doivent être connectés sur le même réseau).

## 3.2 Conception et fabrication du robot

Le robot est passé par plusieurs stades. Nous sommes partis du robot principal fourni, puis nous l'avons modifié dans l'optique d'améliorer sa précision de déplacement pour la cartographie.

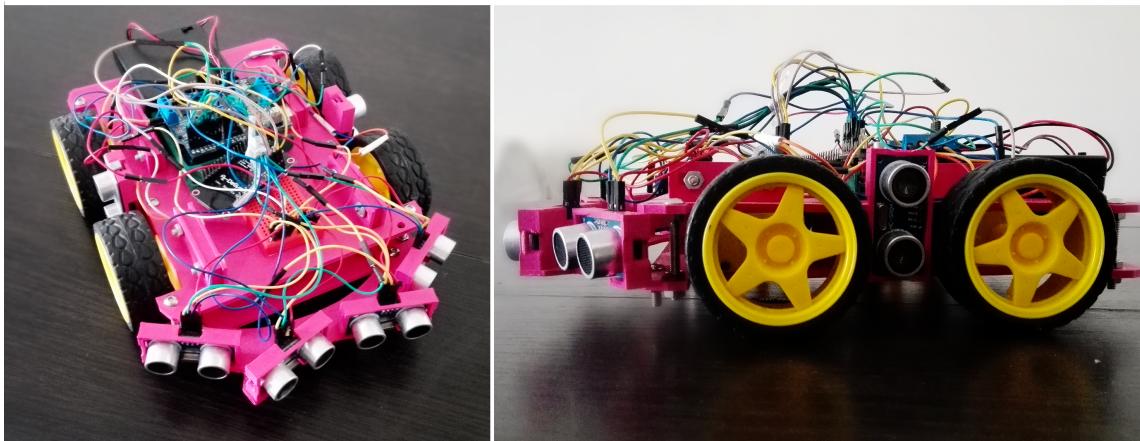


FIGURE 5 - Le robot monté

### 3.2.1 Modélisation et impression 3D

L'objectif était d'obtenir un robot le plus stable possible pour minimiser les erreurs de précision pouvant se créer lors du déplacement ou de la rotation du robot. Nous avons donc décidé d'ajouter deux roues supplémentaires et de fixer le tout correctement afin qu'il n'y ait aucun jeu. La totalité du châssis a ainsi été recréée à l'aide de pièces imprimées en 3D.

Les différentes pièces imprimées sont les suivantes :

**- Plaque supérieure** C'est une simple plaque d'environ 20 cm de longueur par 16 cm de largeur, elle possède quatre encoches permettant de visser les supports des moteurs. Elle permet également de supporter une des deux batteries et la carte Arduino Uno avec son shield.

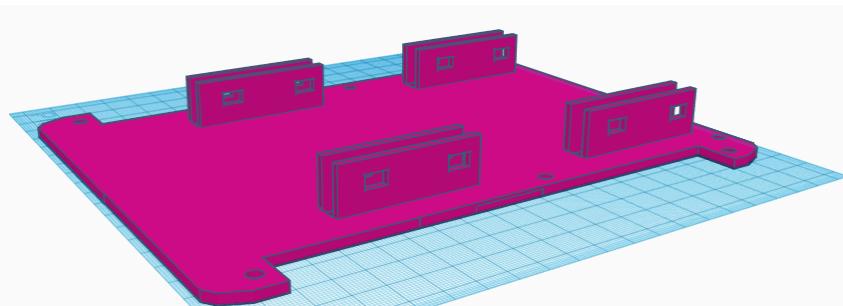


FIGURE 6 - Modèle 3D de la plaque supérieure

**- Plaque inférieure** Cette plaque de 20 cm de longueur par 16 cm de largeur a deux supports pour capteur ultrason (un de chaque côté), elle permet de fixer le support pour les capteurs et de supporter une batterie.

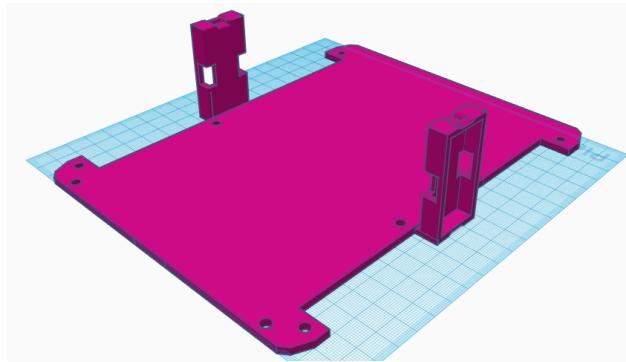


FIGURE 7 - Modèle 3D de la plaque inférieure

**- Support moteur** C'est une petite pièce de 4,5 cm de large par 3.5 cm sur laquelle on peut fixer un moteur. Les mesures pour les dimensions de cette pièce ont été prises de sorte à ce qu'il n'y ait pas de jeu entre le moteur et le support afin de limiter les erreurs d'imprécision.

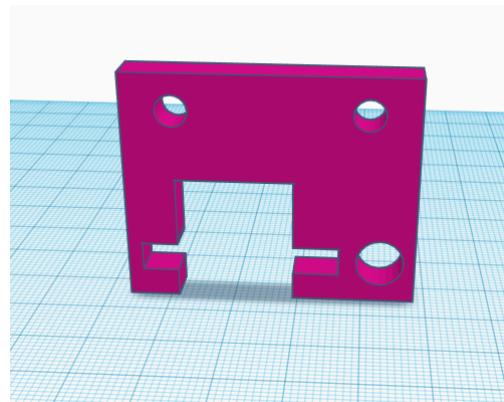


FIGURE 8 - Modèle 3D support de moteur

**- Support capteurs** Constitué de quatre supports de capteur ultrason, cette pièce se fixe à l'avant du robot et maintient les capteurs ultrason.

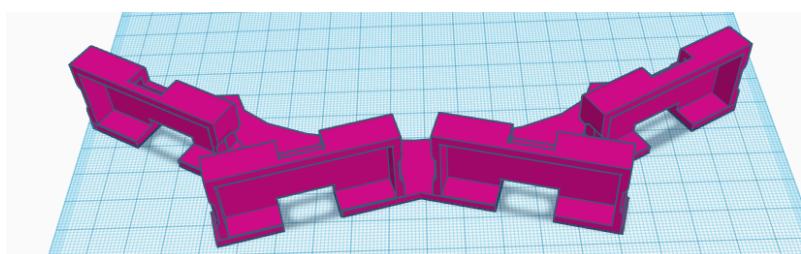


FIGURE 9 - Modèle 3D du support de capteurs

### 3.2.2 Matériel et assemblage

#### Matériel nécessaire au montage :

- Une carte Arduino Uno
- Un shield Arduino
- Une carte WiFi ESP8266
- Quatre moteurs avec roues
- Des câbles Arduino
- Six capteurs à ultrason
- Des piles ou une batterie

#### Matériel imprimé en 3D :

- Une plaque supérieure
- Une plaque inférieure
- Quatre fixations moteurs
- Un support capteurs

L'assemblage du robot se fait de la manière suivante : il suffit de fixer un moteur sur chaque support moteur puis de visser les supports sur la plaque supérieure. Ensuite, il faut fixer le support des capteurs sur la plaque inférieure puis attacher cette dernière sous la plaque supérieure. Une fois le châssis monté avec les roues, on branche correctement les différents composants comme dans le schéma du circuit ci-dessous. L'ensemble du circuit est supporté par le châssis.

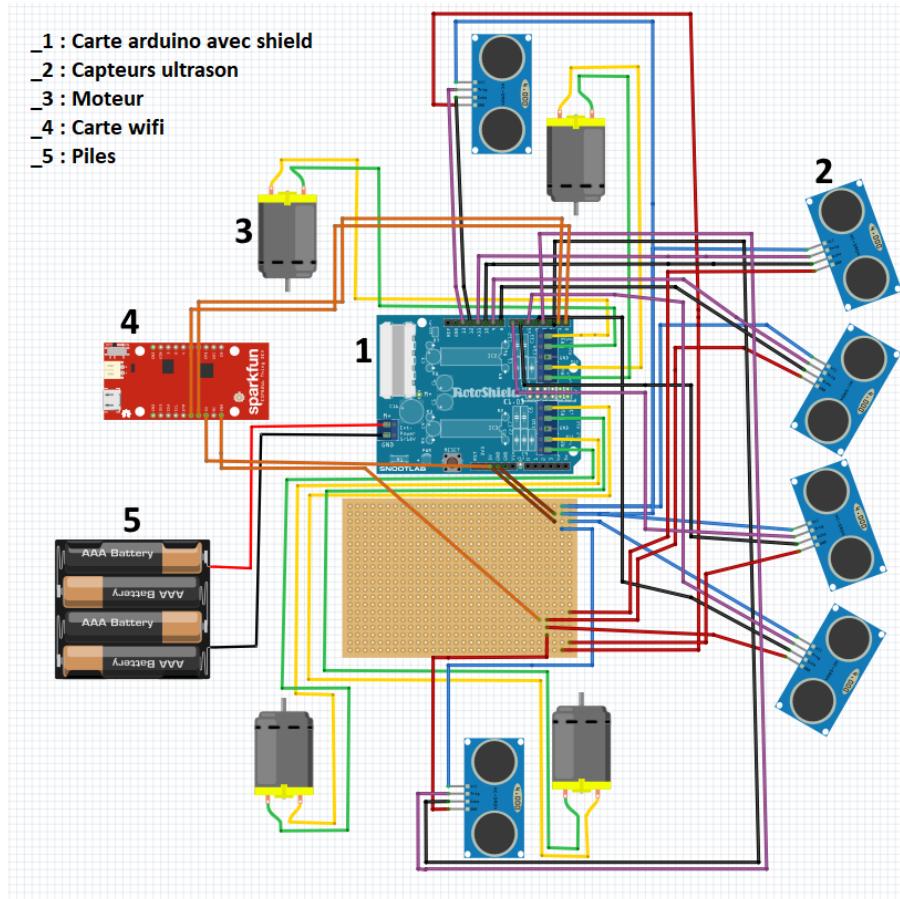


FIGURE 10 - Schéma circuit

### 3.3 Méthode d'exploration

#### 3.3.1 Calibrage du robot

Afin de pouvoir savoir où se trouve le robot dans l'espace, nous avons décidé de le faire se déplacer pas à pas, avec à chaque pas la même distance parcourue (ou rotation effectuée). Plus la distance effectuée est petite, plus l'exploration est longue, plus la carte devrait être précise. Nous avons choisi d'utiliser une distance de 3 cm et un angle de 10°. On essaie ensuite de calibrer le robot pour qu'il parcourt et effectue précisément les distances et les rotations voulues. Cela nous permet donc de pouvoir récupérer la position et l'angle du robot sur un plan 2D.

#### 3.3.2 Algorithme

L'algorithme principal du robot va être l'algorithme qui va lui permettre de longer un obstacle. Cet algorithme est divisé en deux grandes phases : une phase d'exploration, où le robot va chercher un obstacle à longer, et une phase de suivi dans laquelle le robot va longer un obstacle jusqu'à le perdre ou en avoir fait le tour.

**Exploration** La phase d'exploration est plutôt simple : le robot continue d'avancer en ligne droite jusqu'à ce qu'il rencontre un obstacle. Une fois l'obstacle rencontré, il aligne un des deux capteurs situé sur les côtés par rapport à l'obstacle. Pour faire simple, le robot se tourne jusqu'à ce qu'un capteur de côté capte l'obstacle en question.

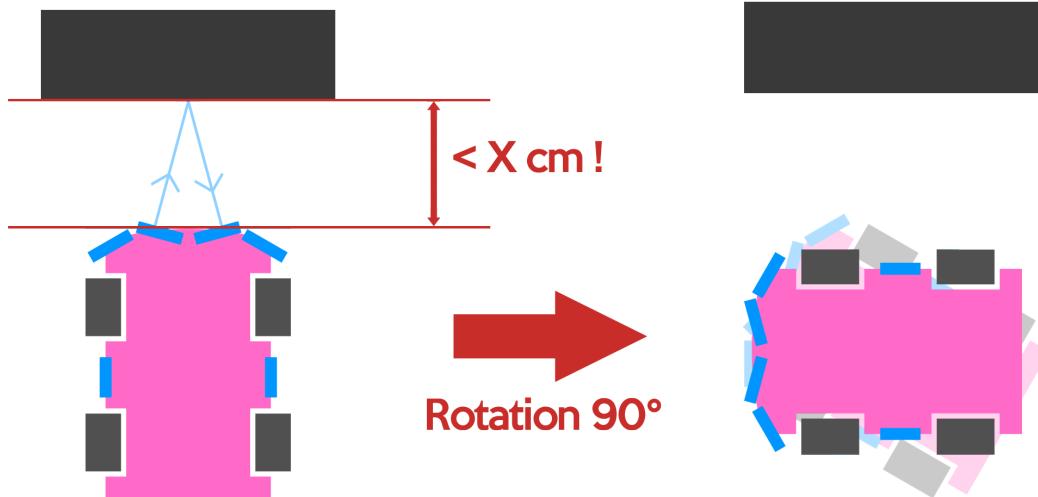


FIGURE 11 - Schéma de la détection d'un obstacle

**Longer** Pour longer un obstacle, le robot essaie de maintenir une certaine distance. Dans les faits, il essaie de rester à  $9 \text{ cm} \pm 3 \text{ cm}$ . Lorsqu'il détecte qu'il est en train de trop s'éloigner, il effectue une rotation de 10° en direction de l'obstacle. De même, s'il s'approche trop, il effectue une rotation de 10° dans la direction opposée à l'obstacle.

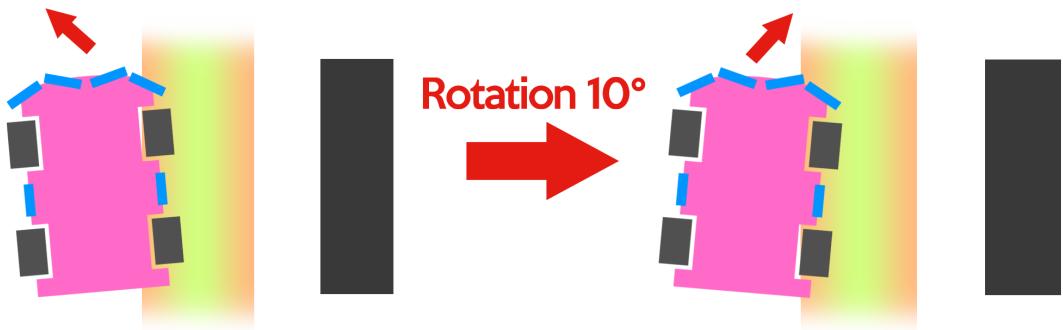


FIGURE 12 - Schéma montrant comment le robot se recentre lors de la phase de suivi d'un obstacle. Ici, le robot détecte qu'il s'éloigne trop de l'objet et décide de tourner sur la droite pour se recentrer.

**Perte d'un obstacle** Dans le cas où le robot est en train de longer un obstacle et qu'il ne le détecte plus, cela signifie qu'on vient de "dépasser" l'obstacle. Dans le cas par exemple d'une boîte en carton, cela signifie que l'on arrive à un coin. Dans ce cas, le robot va effectuer une rotation de  $10^\circ$  en direction de l'obstacle et avancer de 3 cm, jusqu'à retomber sur un obstacle (et ceci pour un maximum de  $120^\circ$  de rotation). Si après cela il n'a toujours pas trouvé d'obstacle, il repasse en mode exploration.

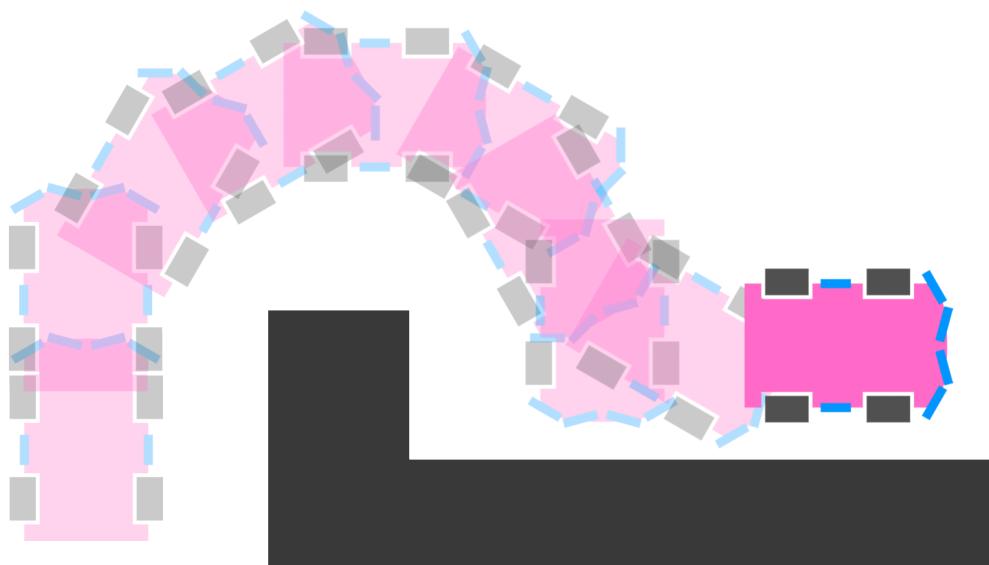


FIGURE 13 - Schéma du comportement du robot lors de la perte d'un obstacle.

**Condition de fin** Nous n'avons pas réussi à trouver une condition globale de fin satisfaisante. Cependant, nous avons pu résoudre ce problème à plus petite échelle : savoir quand on a terminé de longer un objet. Lorsque le robot rentre dans la phase

de suivi, il "place" des points qu'il garde en mémoire. Quand il retombe sur ces points, si la distance parcourue depuis le moment où le point a été ajouté est assez élevée, alors il considère avoir fait le tour de l'objet, et se tourne dans la direction opposée à l'obstacle pour continuer son exploration. Sinon, il continue de longer.

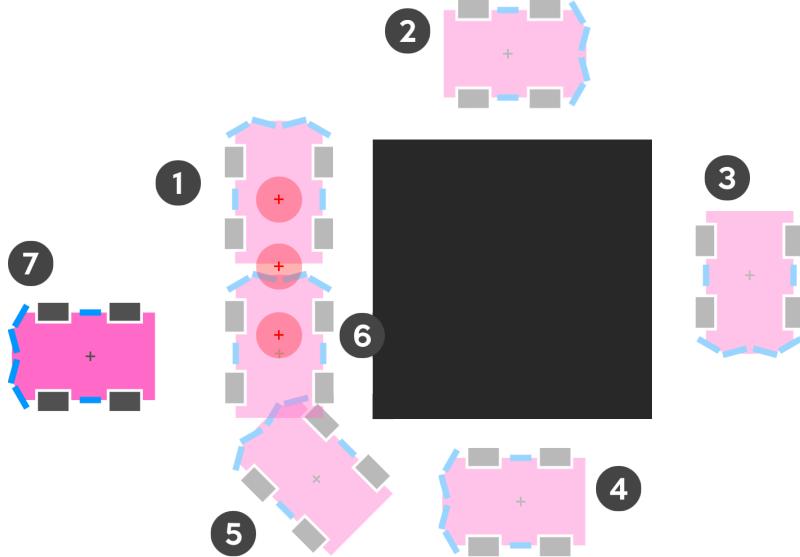


FIGURE 14 - Schéma de la condition de fin d'arrêt. À l'étape 6, on remarque que le centre du robot se trouve dans le rayon d'un des repères. Il quitte alors l'objet (étape 7).

## 3.4 Cartographie et communication

### 3.4.1 Cartographie

Une fois notre robot explorateur terminé, il nous reste à mettre en place un outil permettant de dessiner une carte à partir des points mesurés par le robot. Nous avons décidé de réaliser cet outil en Java.

**Modèle** Pour modéliser notre carte, nous avons choisi d'utiliser un graphe, c'est-à-dire des sommets reliés entre eux par des arcs. Nous avons également pris la décision que deux sommets soient reliés entre eux s'ils se trouvent à une certaine distance l'un de l'autre. Afin d'optimiser la complexité de l'ajout d'un point dans un grand graphe, nous avons décidé de diviser le graphe en zones. Un point ne pourra se lier avec un autre point que s'il se situe dans la même zone que lui, à l'exception des sommets situés sur le bord d'une zone (qui pourront se lier avec les sommets de la zone ou des zones voisines).

**Représentation graphique** La carte est un simple canvas sur lequel on dessine les sommets. Le plus compliqué est de passer de la représentation graphe à la représentation sur un plan 2D (car les coordonnées des sommets peuvent être négatives), de telle sorte que les proportions de distances soient respectées. Pour cela on regarde quels sont les points situés sur les extrémités (gauche, droite, haut

et bas), puis on calcule le ratio (ou "zoom") permettant l'affichage de tous les points sur la zone de dessin. À partir de ce ratio, il est possible d'attribuer une position ( $x$ ,  $y$ ) sur le canvas à chaque point du graphe tout en respectant le rapport des distances. Les points blancs représentent le déplacement du robot et les points des autres couleurs sont les points détecter par les capteurs (à chaque capteur correspond une couleur).

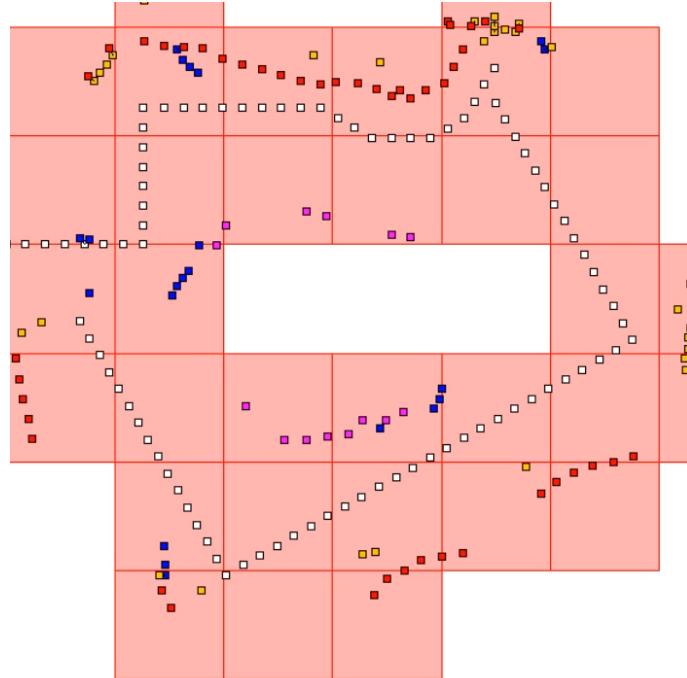


FIGURE 15 - Cartographie obtenue

**Ajout d'un sommet** Pour pouvoir ajouter un sommet à la carte, nous devons disposer de plusieurs informations : la position depuis laquelle est ajoutée ce sommet (la position du robot), l'angle auquel on a détecté le sommet (l'angle du robot + l'angle du capteur) ainsi que la distance du sommet par rapport à la position du robot. À partir de là, une simple opération de trigonométrie nous donne les coordonnées ( $x$ ,  $y$ ) du nouveau sommet ajouté.

$$x = dist \times \cos(a)$$

$$y = dist \times \sin(a)$$

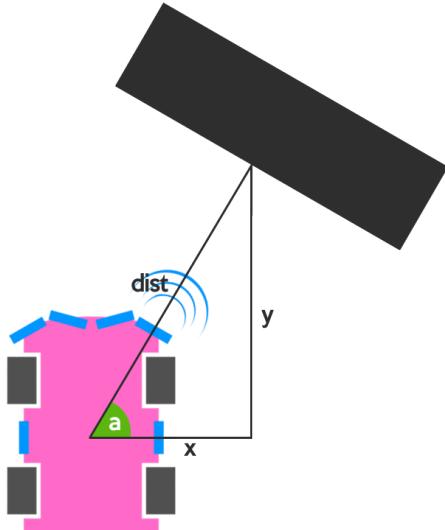


FIGURE 16 - Schéma de trigonométrie

### 3.4.2 Communication des données

Pour assurer l'aspect communication, nous avons opté pour une architecture X-Robots → Serveur. Cela permet tout d'abord de centraliser toutes les informations,

mais également de simplifier la mise en place de plusieurs robots, car un robot doit seulement connaître le serveur et non plus tous les autres robots (l'architecture est donc plus modulable). Les communications s'effectuent par des Sockets sur un serveur Java.

**Échantillonage** Lors de son exploration, le robot effectue différentes mesures grâce aux capteurs à ultrasons. Il effectue un échantillonnage après chaque déplacement ou rotation. Lorsqu'il effectue une mesure, celui-ci enregistre les données sous forme de JSON, puis transmet les informations depuis la carte Arduino jusqu'au module WiFi.

---

```
{  
    "x": position x du robot,  
    "y": position y du robot,  
    "angle": angle du robot,  
    "distances": {  
        "0": mesure du capteur 0 (gauche)  
        "1": mesure du capteur 1 (haut/gauche)  
        ...  
        "5": mesure du capteur 5 (droite)  
    }  
};
```

---

#### JSON CRÉÉ ET ENVOYÉ PAR LE ROBOT

**Envoy et traitement des données** Lorsque le module WiFi reçoit les informations provenant de la carte Arduino, celui-ci les envoie directement au serveur. Lors de la réception d'un message, le serveur vérifie qu'il est valide ou qu'il s'agit de plusieurs informations en un message (JSON séparés par un ";" ), puis interprète le ou les JSON et récupère les données relevées par le robot dans une pile qu'il traitera ensuite. Il ajoute finalement les points situés en dessous d'une certaine distance par rapport au robot à la carte, et met à jour la carte.

## 4 Gestion de projet

### 4.1 Logistique

Tout d'abord nous tenons à remercier le Fablab de l'UNS ainsi que les "Fablab managers" pour leur aide, les conseils et le matériel mis à notre disposition. Au Fablab nous avons pu utiliser les imprimantes 3D et toute sorte d'outils afin de réaliser le châssis de notre robot. Nous avons utilisé Github pour gérer les différentes versions des sources. Pour la communication entre les membres de notre groupe nous avons utilisé principalement le logiciel "Discord", qui nous a permis de travailler quelquefois à distance, en conversation vocale et avec le partage d'écran. Pour le développement nous avons principalement utilisé l'IDE d'Arduino ainsi que IntelliJ

(pour le code java du serveur et du traçage de la carte). Pour la modélisation et l'impression des composants 3D nous avons respectivement utilisé Tinkercad et Cura.

## 4.2 Répartition des tâches

À propos de la répartition du travail effectué, tous les membres du groupes ont plus ou moins participé à chaque aspect du TER. Nous allons tout de même préciser sur quels aspects chaque membre du groupe a principalement travaillé.

**Modèle et construction** Steve, Michel.

**Algorithme d'exploration** Dans un premier temps Michel et Aymeric, et pour la version finale principalement Victor.

**Algorithme de cartographie** Victor et Steve.

**Communication** Aymeric, Victor et Michel.

## 5 Difficultés rencontrées

Durant ce TER, nous nous sommes confrontés à beaucoup de problèmes. La plupart, comme nous allons le voir, sont surtout reliés au matériel et à l'imprécision de celui-ci.

### 5.1 Mesure des capteurs

Dans un premier temps, nous avons constaté que les mesures des capteurs étaient imprécises. En effet, certains capteurs semblaient être capricieux de temps en temps. Il arrivait que le robot "détecte" quelque chose devant lui alors qu'il n'y avait strictement rien, lui faisant ainsi changer d'état dans la plupart des cas. De plus, quelques interférences ont été observées vis à vis des deux capteurs de devant.

### 5.2 Positionnement

Savoir où se trouve le robot dans l'espace a été notre plus gros problème durant ce TER. Étant donné qu'il nous était impossible de faire ceci côté matériel, nous avons dû nous retrancher sur le côté logiciel : le robot calculait lui-même sa position en fonction de ses déplacements. Bien qu'ayant effectué plusieurs fois le calibrage du robot, la position ainsi que l'angle de ce dernier semblaient à chaque fois dériver de la réalité, sans que nous puissions mesurer cet écart depuis le robot (nous pouvions seulement le constater au tracé de la carte). De plus, après plusieurs tests, la

distance à laquelle le robot ”dérive” semble chaotique et dépendre de l'environnement (surface irrégulière comme du carrelage par exemple), et toutes nos tentatives pour contrebalancer ce phénomène se sont avérées inutiles. Il s'agit d'ailleurs d'un problème récurrent concernant ce genre de robot cartographe.

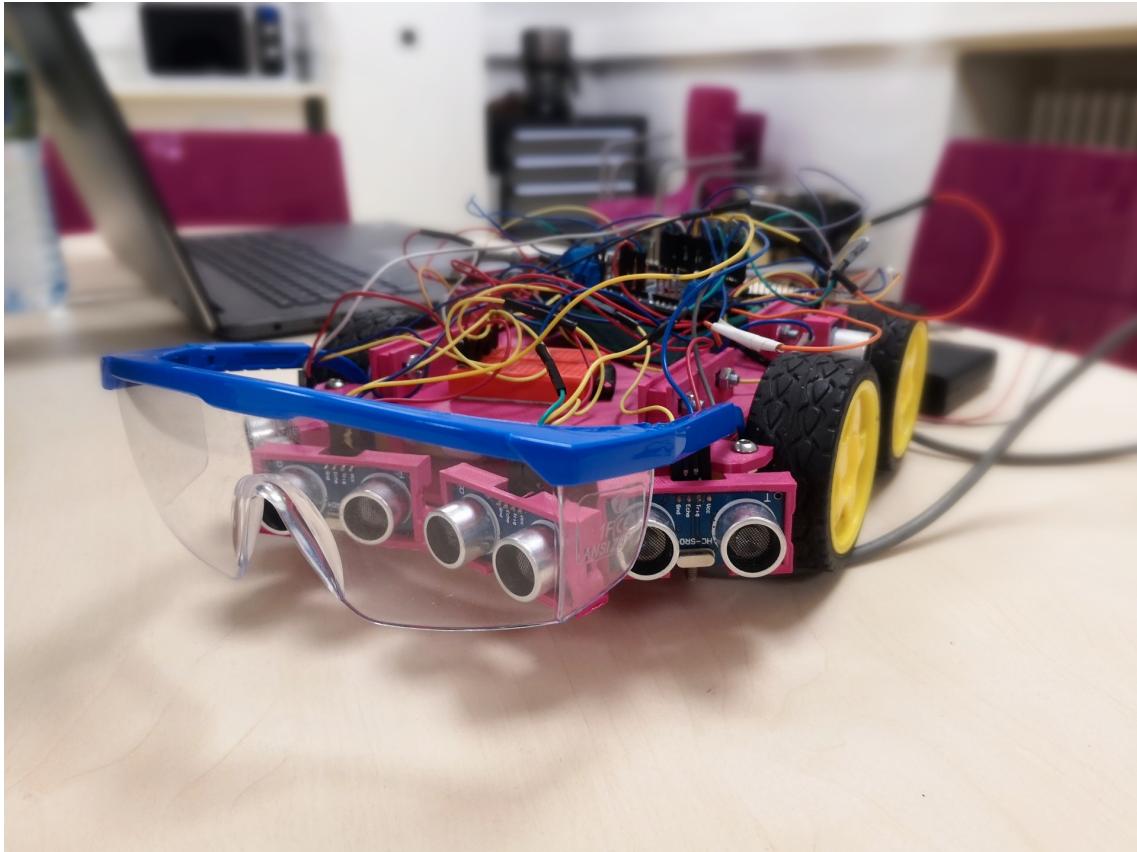
### 5.3 Mémoire

Au départ, le nombre de données à enregistrer était visiblement trop important pour la carte Arduino. Il a fallu ainsi réduire leur nombre, ce qui a affecté le comportement du robot lorsqu'il s'agissait de détecter si il avait fait le tour complet d'un obstacle. De plus cela nous a empêché d'implémenter un algorithme de pathfinding dont on parlera plus tard.

### 5.4 Communication WiFi

Nous avons aussi eu des problèmes pour transmettre des informations (position, distances captées) de la carte Arduino à un serveur hébergé sur une de nos machines. En effet nous avons tout d'abord essayé de faire communiquer la carte Arduino en WiFi (sans succès) à l'aide d'une carte de type ”ESP32”, qui nous avait été fournie avec le reste du matériel (ce qui nous a fait perdre beaucoup de temps). Nous avions plusieurs cartes de ce même modèle, cependant malgré de nombreuses tentatives nous n'avons jamais réussi à connecter la carte en WiFi. Nous avons finalement décidé d'utiliser un autre modèle de carte WiFi de type ”ESP8266”, que nous avons réussi à configurer sans trop de problème. Peut-être que les cartes WiFi ”ESP32” fournies étaient défectueuses.

## 6 Résultats



### 6.1 Déplacement et comportement du robot

Les résultats obtenus correspondent à nos attentes : le robot est capable de longer et de faire le tour d'une grande variété d'obstacles, puis de les quitter une fois ce tour effectué. De même, les capteurs, bien que parfois capricieux, repèrent les obstacles à longer et renvoient une distance correcte pour ajouter les points sur la carte. Cependant le manque de précision dû aux déplacements du robot empêche parfois ce dernier de partir une fois le tour d'un obstacle réalisé. Il est également important de noter que lorsque les capteurs de devant produisent une erreur de mesure, le robot peut alors entrer dans le cas où il pense avoir détecté un obstacle devant lui et se tourne alors de manière "spontanée" de 90°.

### 6.2 Cartographie obtenue

Les résultats de cette partie sont moins satisfaisants : bien que la communication du robot vers le serveur fonctionne et que l'ajout des points sur un plan 2D s'effectue de manière correcte, un décalage de la position du robot entre l'environnement réel et le numérique est présent. Le nombre de déplacement entraîne une imprécision de plus en plus importante ce qui nous donne une carte non fidèle à la réalité comme on peut le constater sur la comparaison suivante :

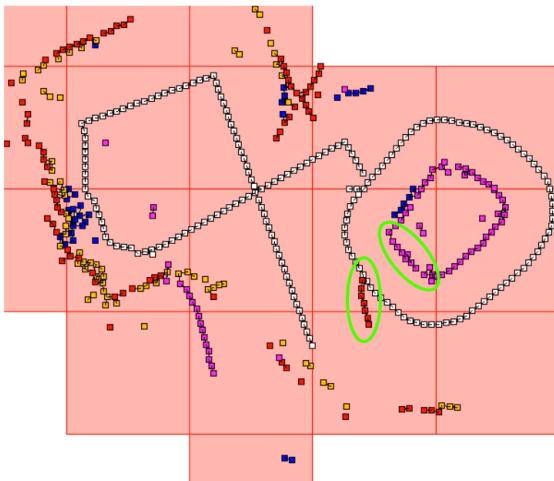


FIGURE 17 - Carte obtenue par le robot

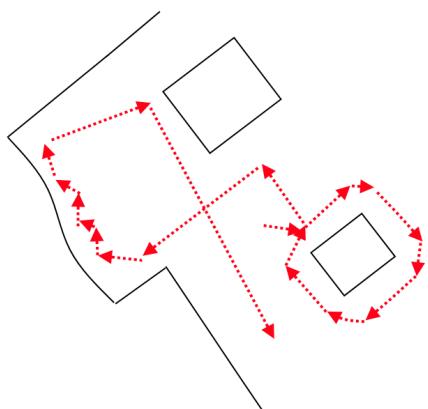


FIGURE 18 - Carte réelle de la zone explorable

Sur la carte obtenue ci-dessus, les deux suites de points obtenues entourées en vert correspondent à la même paroi, seulement celle en magenta a été dessinée au tout début de l'exploration, tandis que celle en rouge l'a été bien après.

Notre manière de procéder pour déterminer sa position doit être la cause de ce décalage : en effet la présence d'un sol irrégulier (bosses ou trous), son adhésion au sol ou les éventuels dysfonctionnements d'un des moteurs peut entraîner un léger décalage par rapport au déplacement attendu du robot. Cependant, si on regarde obstacle par obstacle, là où le décalage n'a pas le temps de devenir trop important, on est plutôt satisfait (le carton ressemble à un rectangle avec plus ou moins les bonnes dimensions).

## 7 Conclusion

Notre projet était de réaliser une flottille de robot capable d'explorer un espace. Au final, nous avons préféré nous focaliser sur un seul robot, tout en gardant en tête l'idée d'une flottille au niveau de l'architecture du code. Le robot est bien capable de suivre un obstacle correctement, et de le quitter une fois celui-ci visité. Il est également capable d'envoyer les mesures des capteurs à un serveur situé à distance, serveur qui s'occupe de la centralisation d'information et du dessin de la carte. Cependant, la manière dont on traque la position du robot est beaucoup trop imprécise, si bien que les écarts de mesures sur la carte deviennent de plus en plus importants au fil de l'exploration.

## 8 Perspectives et réflexions personnelles

Dans cette partie nous allons parler des idées que nous avons eu afin de pouvoir améliorer le projet, mais qui ont finalement été abandonnées car cela représentait

une charge de travail trop importante (à la limite d'un sujet de TER à part entière) ou car nos résultats ne le permettaient pas.

## 8.1 Algorithme d'exploration

Afin de résoudre le problème d'une exploration intelligente, nous avions en tête de mettre en place un algorithme de *pathfinding* (plus précisément un A\*) afin de pouvoir diriger le robot vers des zones pas encore explorées lorsqu'il ne sait plus où aller, c'est-à-dire dans les cas suivants :

- au début de l'exploration
- quand il a fini de longer un obstacle
- quand il a atteint une zone indiquée par l'algorithme de pathfinding sans rencontrer d'obstacle

Cet algorithme permettrait aussi de détecter les zones inaccessibles (car entourées d'obstacles). Nous nous sommes cependant heurtés assez rapidement à un problème de mémoire du côté de la carte Arduino. Pour résoudre ce problème, nous avions pensé à utiliser la puissance du serveur afin de réaliser le calcul, puis de transmettre les résultats au robot. Cela suppose également que l'on mette en place un système de communication du serveur vers le robot. Cela n'a finalement pas été fait par manque de temps et par manque de précision du déplacement du robot.

## 8.2 Système de positionnement

En vue des résultats concernant le positionnement du robot, nous avons cherché un moyen de résoudre ce problème. Nous nous sommes donc intéressés à la création d'un système de positionnement plus précis, en utilisant la triangulation ou la trilateration, avec par exemple trois antennes placées dans la pièce à explorer. Nous avions également pensé à avoir un capteur puissant et précis qui permettrait de calculer la distance que l'on vient de parcourir (un laser par exemple), ou bien encore à un accéléromètre. Concernant l'angle dans lequel se trouve le robot, nous avons également pensé à l'utilisation d'un gyroscope. En couplant la partie matériel à la partie logiciel, nous pensons qu'il est possible d'obtenir une très bonne approximation de la position et de l'angle du robot.

## 9 Annexes

**Github :** <https://github.com/MMarmone/Flotilles-de-robots>

**Vidéo de notre première version du robot :** <https://www.youtube.com/watch?v=SoIFKPasbq4>

**Vidéo de démonstration finale :** <https://youtu.be/eTb0e0Awots>