

ROBOT EXPLORATEUR M1 IFI UNIVERSITE DE NICE SOPHIA-ANTIPOLIS

Robot explorateur
Enrico Formenti et Marie Pelleau

https://github.com/ChloeMaccarinelli/TER_M1_S2

Yacine Lotfi
Chloé Maccarinelli
Pierre Marion



Membre de UNIVERSITÉ CÔTE D'AZUR 

Table des matières

| | | |
|----|---|----|
| 1- | Introduction..... | 2 |
| 2- | Etat de l'art..... | 3 |
| 3- | Outils et matériels | 4 |
| 4- | Travail effectué..... | 5 |
| 1- | Création/Adaptation des pièces..... | 5 |
| 2- | Programmation et tests des différents composants..... | 6 |
| | Les capteurs Ultrasons HC-SR04..... | 6 |
| | Les moteurs | 7 |
| 3- | Assemblage des composants | 7 |
| | Le circuit | 7 |
| | Le robot | 9 |
| 4- | Le déplacement du robot | 9 |
| | 1 ^{ère} version..... | 9 |
| | 2 ^{ème} version..... | 13 |
| 5- | La réception des données | 15 |
| | Le module WIFI ESP8266..... | 15 |
| | Réception des données | 16 |
| 5- | Gestion du projet..... | 18 |
| 6- | Conclusion | 19 |
| 7- | Perspectives et réflexions..... | 20 |
| 1- | Réflexions | 20 |
| | Principe d'odométrie, Filtre de Kalman et algorithme de SLAM..... | 20 |
| 2- | Perspectives..... | 22 |
| | ANNEXE 1 : POSITION DES CAPTEURS | 23 |
| | ANNEXE 2 : PIECES INKSCAPE | 25 |
| | ANNEXE 3 : TEST 1 CAPTEUR | 26 |
| | ANNEXE 4 : TEST 3 CAPTEURS | 26 |
| | ANNEXE 5 : MOTEURS ET CAPTEURS..... | 26 |
| | ANNEXE 6 : DEPLACEMENTS V1 | 26 |
| | ANNEXE 7 : DEPLACEMENTS V2 | 26 |

1- Introduction

La nature fortement expérimentale de la recherche en robotique conditionne les progrès de l'état de l'art à la disponibilité de plates-formes ayant de bonnes capacités fonctionnelles et largement accessibles, en coût et en complexité d'utilisation. De ce fait la robotique parvient à une certaine maturité avec une accélération importante du domaine du fait de la présence sur le marché de plates-formes de recherche stables, de qualité industrielle, avec des environnements de programmation relativement matures.

Ce TER s'inscrit dans un projet de robot explorateur capable de détecter et éviter des obstacles.

L'objectif principal était d'implémenter une version de l'algorithme A* sur Arduino. Or une redéfinition du sujet a été effectuée. En effet, au vu du temps imparti, de la pertinence et faisabilité du sujet, nous avons été amenés à créer un robot explorateur et commencer un travail sur la localisation du robot et la cartographie de l'espace.

Notre groupe est composé de Yacine Lotfi, Chloé Maccarinelli et Pierre Marion et a été encadré par Madame Pelleau.

Dans ce rapport nous commencerons par détailler les contraintes de départ ainsi que les outils utilisés puis nous décrirons les travaux effectués avant de conclure et d'évoquer les possibles améliorations.

2- Etat de l'art

La robotique est un domaine pluridisciplinaire qui a toujours fasciné l'homme avec la possibilité de créer un mécanisme capable de réaliser nos tâches à notre place et parfois mieux que nous.

Ses réalisations conjuguent l'automatique, l'informatique, la mécanique et l'électronique. Elle est très présente en intelligence artificielle, laquelle est au cœur de la boucle perception – décision – action, en particulier pour les problèmes suivants :

- Interprétation de l'environnement et sémantique des données sensorielles
- Action délibérée
 - Planification de mouvements et de tâches
 - Exécution de mouvements et de tâches

La robotique d'exploration permet à des robots mobiles dans différents environnements de cartographier, d'analyser le sol ou sous-sol, d'extraire, d'intervenir sur des sites variés par le déploiement d'instruments divers (exploration maritime, terrestre ou spatiale).

Un robot est une machine composée d'organes – actionneurs, capteurs, calculateurs, radio-transmetteurs – capable d'accomplir un ensemble de tâches avec un certain degré d'autonomie en tenant compte de son environnement. Une difficulté essentielle en robotique est la variabilité des environnements et des tâches auxquelles un robot peut être confronté.

Il présente en outre les capacités suivantes :

- Déplacement : par roues, pattes, ailes, nageoires, chenilles
- Préhension : par bras mécaniques, pinces, main, ventouses, outils spécialisés
- Perception par capteurs :
 - Proprioceptifs : qui estiment l'état interne de la machine (odomètre, inclinomètre, GPS) et par capteurs
 - Extéroceptifs : qui informent sur l'environnement extérieur : caméra, laser, radar, spectromètre, télémètre à infrarouge ou à ultrason
- Communication : wifi, Bluetooth
- Prise de décision

L'idée principale du sujet est de créer un robot explorateur capable de détecter et d'éviter les obstacles. Nous avons à notre disposition la référence d'un robot réalisé l'an dernier. <https://github.com/master1-ifi-semester2/TER>. Leur robot utilise 6 capteurs ultrasons, une carte Arduino UNO et à un châssis sur mesure.

Nos contraintes :

- 1 carte Arduino Leonardo
- 3 capteurs ultrasons
- Un kit imposé pour le châssis

Le sujet nous amène également à réfléchir sur la notion de cartographie et localisation simultanée. En d'autres termes, le robot doit être capable de construire une carte de son

environnement et de s'y localiser sans connaissances préalables. De nombreux travaux de recherches sont disponibles et nous en avons retenu un type d'algorithme en particulier :

- L'algorithme de type SLAM

Les travaux de recherche réalisés sur cet algorithme sont très nombreux et offre diverses versions du SLAM mais l'idée principale reste la même et se découpe en 4 étapes :

Perception : Fournir des données issues de capteurs (ultrasons, lasers, etc.) un ensemble d'observations de l'environnement qui constituent des amers (les obstacles par exemples).

Association de données : Étant donné une observation, on doit décider si cette observation correspond à un amer déjà présent dans la carte, et de quel amer il s'agit, ou bien alors si c'est un nouvel amer à ajouter à la carte.

Proprioception : Des données internes au système qui renseignent l'évolution d'un état (position du robot dont l'évolution est typiquement fournie par des capteurs odométriques).

Estimation : Intègre les données issues de la perception ainsi que de la proprioception afin de fournir une estimation de la position du robot et des positions des amers, en prenant en compte les incertitudes (dû à l'environnement par exemple).

Nous nous focaliserons sur ces recherches pour comprendre au mieux comment il pourrait être adapté à notre robot.

3- Outils et matériels

Logiciels utilisés pour la réalisation du projet :

- Inkscape
- Arduino IDE
- GitHub

Matériels utilisés :

- 2x carte Arduino Leonardo
- 1x Adafruit MotorShield
- 1x breadboard
- 3x capteurs ultrasons HC-SR04
- 1x kit Seed Tricycle Bot
- 1x pile 9V
- 1x câble USB-micro USB
- 1x module WIFI ESP826
- Des câbles

Matériaux :

- Plexiglas

4- Travail effectué

1- Création/Adaptation des pièces

La première étape imposée est la modélisation des pièces du kit figure 1 : kit imposé, afin de pouvoir les réutiliser. Pour cela, nous utilisons le logiciel Inkscape, retranscrivons les mesures et en redimensionnons certaines afin de pouvoir les découper dans du plexiglass à la découpeuse laser.

Les mesures :

Les mesures des pièces du kit ne sont pas forcément adaptées aux capteurs ultrasons que nous utilisons. De plus l'angle prédéfini ne permet pas aux capteurs de couvrir la zone de détection correctement. Nous effectuons un travail de réflexion sur la position des capteurs ([annexe 1 : positions des capteurs](#)).

Les pièces du robot nous étant imposées, nous devons nous adapter pour la position des capteurs. Au lieu de les positionner selon la réflexion vue en [annexe 1 : positions des capteurs](#), nous les plaçons avec un angle d'environ 50 degrés d'écart chacun.



Figure 1 : kit imposé

Au niveau des mesures des pièces nous réalisons différentes versions des pièces pour s'adapter à l'épaisseur du plexiglass utilisé lors de la découpe. (Version 3mm et 4mm)

En [annexe 2 : pièces inkscape](#) vous pouvez retrouver les pièces reproduites avec Inkscape et utilisées pour notre robot. Ainsi qu'un lien vers toutes les pièces réalisées.

2- Programmation et tests des différents composants

Les capteurs Ultrasons HC-SR04

1^{ère} étape : test de 1 capteur :

On commence par tester le fonctionnement d'un capteur à l'aide du code que vous pouvez trouver en [annexe 3 : test 1 capteur](#). On cherche à calculer la distance entre un capteur et un obstacle.

$$\text{Distance} = ([\text{Durée du niveau haut}] * [\text{vitesse du son :340m/s}]) / 2$$

Représente la distance entre le capteur et l'obstacle où **Durée du niveau haut** représente la durée de l'impulsion durant au moins 10µs pour que le module démarre sa lecture.

On peut simplifier cette formule grâce aux informations du constructeur :

$$\text{Distance (cm)} = \text{impulsion (}\mu\text{s)} / 58$$

Les données concernant le capteur HC-SR04 (voir [annexe 1 : positions des capteurs](#)) se valident :

- La distance minimale de détection optimale est d'environ 2 cm.
- La distance maximale de détection optimale se situe entre 2 à 4m en fonction du matériau.

2^{ème} étape : test de 3 capteurs :

En suivant la même logique que précédemment on test le fonctionnement de 3 capteurs à l'aide du code que vous pouvez trouver en [annexe 4 : test 3 capteurs](#) et on obtient les résultats suivants (d'après l'environnement de l' image 1: test capteurs):

Ordre de détection : Capteur avant |capteur gauche |
capteur droit

Distance: 200.77mm
Distance: 26.69mm
Distance: 65.96mm

Distance: 201.11mm
Distance: 25.67mm
Distance: 65.62mm

Distance: 201.11mm
Distance: 26.69mm
Distance: 65.11mm

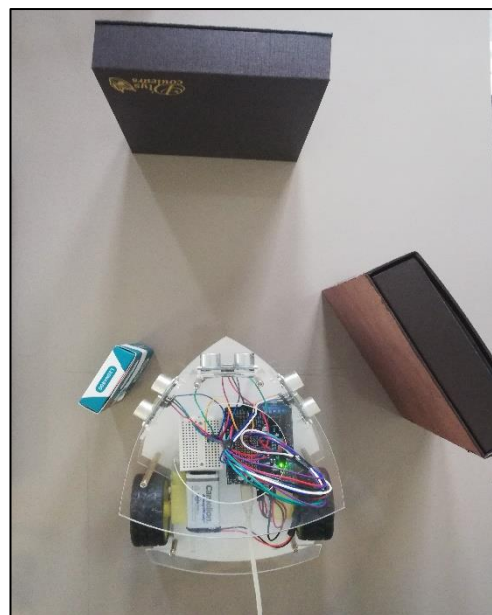


Image 1: test capteurs

Les moteurs

En ce qui concerne les moteurs, nous utilisons la librairie **Adafruit_MotorShield** ce qui facilite leur définition comme on le voit ci-dessous dans l'extrait de code :

```
#include <Adafruit_MotorShield.h>

*****

// Create the motor shield object

Adafruit_MotorShield AFMS = Adafruit_MotorShield();

Adafruit_DCMotor *myMotor = AFMS.getMotor(1) ; // init motor left
Adafruit_DCMotor *myMotor2 = AFMS.getMotor(2) ; // init motor right

*****
```

On les initialise également très simplement comme ci-dessous :

```
void setupMotors() {
  // Left wheel
  motorsG->setSpeed(motorSpeed);
  motorsG->run(FORWARD);
  motorsG->run(RELEASE);

  // Right wheel
  motorsD->setSpeed(motorSpeed);
  motorsD->run(FORWARD);
  motorsD->run(RELEASE);
}
```

3- Assemblage des composants

Le circuit

On commence par brancher les capteurs sur la carte Arduino suivant la figure 2 : circuit page suivante.



Image 2 : branchements

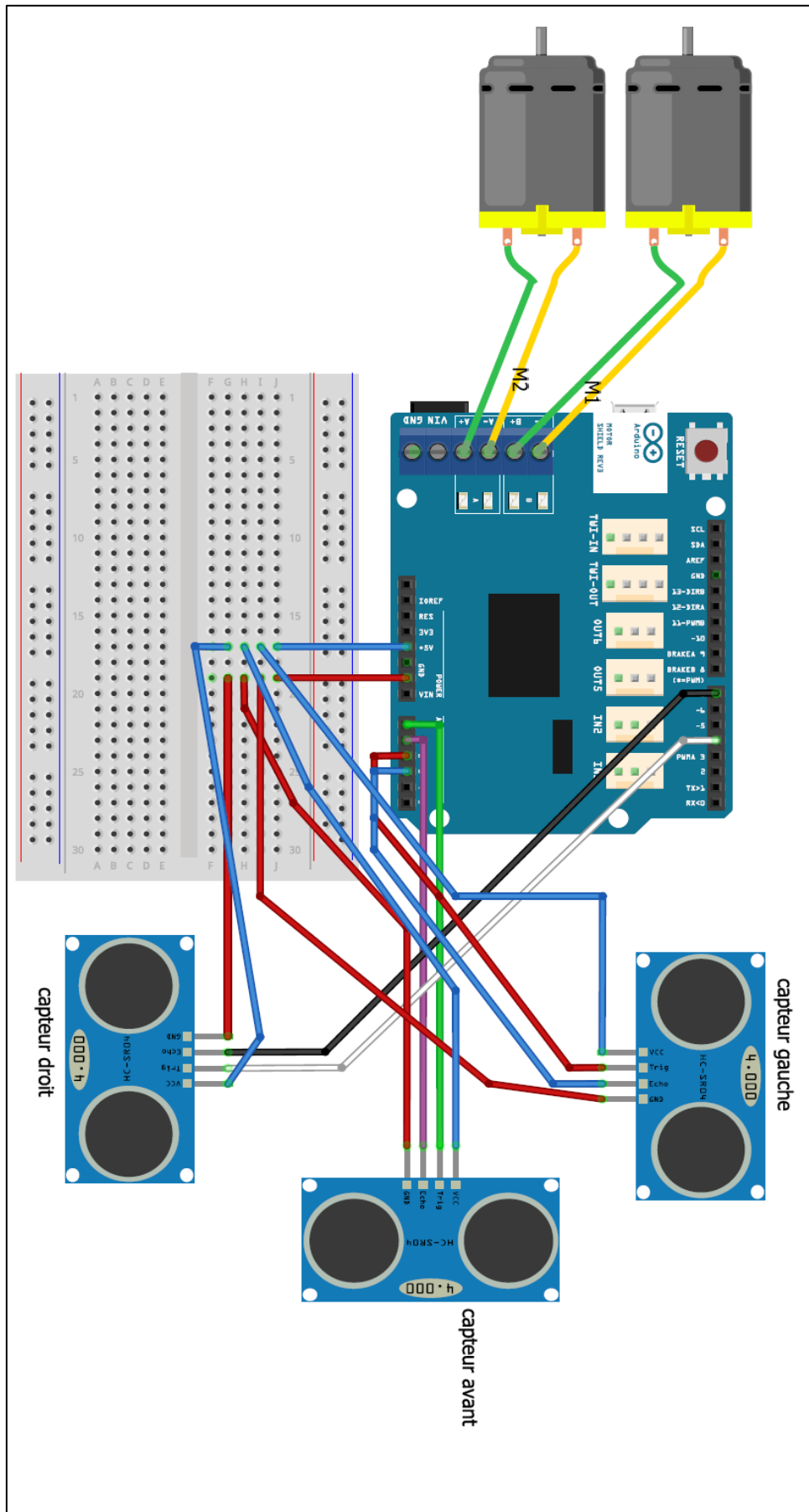


Figure 2: circuit

Le robot

On monte ensuite le châssis et les composants restant et obtenons le robot ci-dessous :

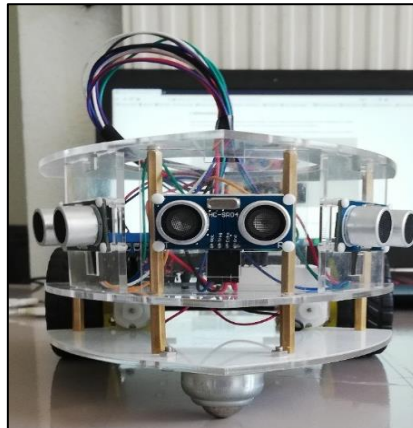


Image 3 : aperçu final

4- Le déplacement du robot

1ère version

Le code

1ère étape :

On met en commun le code de calcul de distance testé précédemment et on le combine au code des moteurs. On peut ainsi vérifier que le branchement des capteurs et des moteurs en simultané ne pose aucun problème.

| | |
|--|--|
| <pre> void setup() { Serial.begin(115200); // set up Serial library at 9600 bps while(!Serial); Serial.println("Adafruit Motorshield v2 - DC Motor test!"); AFMS.begin(); // create with the default frequency 1.6KHz //AFMS.begin(1000); // OR with a different frequency, say 1KHz // Set the speed to start, from 0 (off) to 255 (max speed) myMotor->setSpeed(150); myMotor->run(FORWARD); myMotor2->setSpeed(150); myMotor2->run(FORWARD); // turn on motor myMotor->run(RELEASE); myMotor2->run(RELEASE); } void loop() { </pre> | <pre> uint8_t i; myMotor->setSpeed(50); myMotor2->setSpeed(50); Serial.println("AVANT"); while(tick<3){ myMotor->run(FORWARD); myMotor2->run(FORWARD); tick++; Serial.println("CAPTEUR AVANT 1"); calculDistance(TRIGGER_PIN_AVANT, ECHO_PIN_AVANT); Serial.println("CAPTEUR GAUCHE 1"); calculDistance(TRIGGER_PIN_GAUCHE, ECHO_PIN_GAUCHE); Serial.println("CAPTEUR DROITE 1"); calculDistance(TRIGGER_PIN_DROITE, ECHO_PIN_DROITE); } </pre> |
|--|--|

Avec l'extrait de code ci-dessus (le code complet en [annexe 5 : moteurs et capteurs](#)), on spécifie les actions à effectuer et l'affichage pour prouver que tout fonctionne correctement.

Adafruit Motorshield v2 - DC Motor test!

AVANT

CAPTEUR AVANT 1
Distance: 73.78mm
CAPTEUR GAUCHE 1
Distance: 1969.45mm
CAPTEUR DROITE 1
Distance : 111.35mm
CAPTEUR AVANT 1
Distance: 73.27mm
CAPTEUR GAUCHE 1
Distance: 1969.45mm
CAPTEUR DROITE 1
Distance: 110.50mm
CAPTEUR AVANT 1
Distance: 72.93mm
CAPTEUR GAUCHE 1
Distance: 1969.45mm
CAPTEUR DROITE 1
Distance: 110.33mm

ARRIERE

CAPTEUR AVANT 2
Distance: 72.76mm
CAPTEUR GAUCHE 2
Distance: 1969.45mm
CAPTEUR DROITE 2
Distance: 109.99mm
CAPTEUR AVANT 2
Distance: 37.91mm
CAPTEUR GAUCHE 2
Distance: 1969.45mm
CAPTEUR DROITE 2
Distance: 109.99mm
CAPTEUR AVANT 2
Distance: 30.94mm
CAPTEUR GAUCHE 2
Distance: 1969.45mm
CAPTEUR DROITE 2
Distance: 111.35mm

AVANT

Nous observons que les instructions du moteur n'interfèrent pas sur les instructions des capteurs.

2^{ème} étape :

On passe aux fonctions de déplacements (le code en [annexe 6 : déplacements V1](#)). Le déplacement du robot se base sur la version proposée l'an dernier qui consiste à suivre l'obstacle pour l'éviter et le longer.

Le raisonnement est le suivant (figure 3: algorithme v1):

Cherche à être parallèle à l'objet (cherche l'objet)

- Si obstacle à gauche
 - Si obstacle assez loin
 - Arrête de tourner et passe à la suite
 - Sinon tourne à droite
- Sinon si obstacle à droite
 - Si obstacle assez loin
 - Arrête de tourner et passe à la suite
 - Sinon tourne à gauche

Avance parallèle à l'objet (ne le cherche plus)

- Si obstacle à gauche
 - Si obstacle assez loin
 - Tourne à gauche
 - Si objet hors de la marge tolérée (distance inférieure à distance safe – marge de mouvement) ou objet devant trop près
 - Tourne à droite
 - Sinon avancer
- Sinon si obstacle à droite
 - Si obstacle assez loin
 - Tourne à droite
 - Si objet hors de la marge tolérée ou objet devant trop près
 - Tourne à gauche
 - Sinon avancer
- Sinon si robot viens d'avancer (tick<4 nombre magique)
 - Avancer
 - Si obstacle devant trop près
 - Si obstacle gauche et obstacle droite trop près
 - Reculer
 - Sinon si gauche trop près et droite safe
 - Tourner à droite
 - Sinon si gauche safe et droite trop près
 - Tourner à gauche
 - Sinon reculer pendant un moment (jusqu'à tick≥200 encore nombre magique)
 - Sinon si gauche trop près
 - On indique qu'on a un obstacle à gauche
 - Sinon si droite trop près
 - On indique qu'on a un obstacle à droite
- Avancer

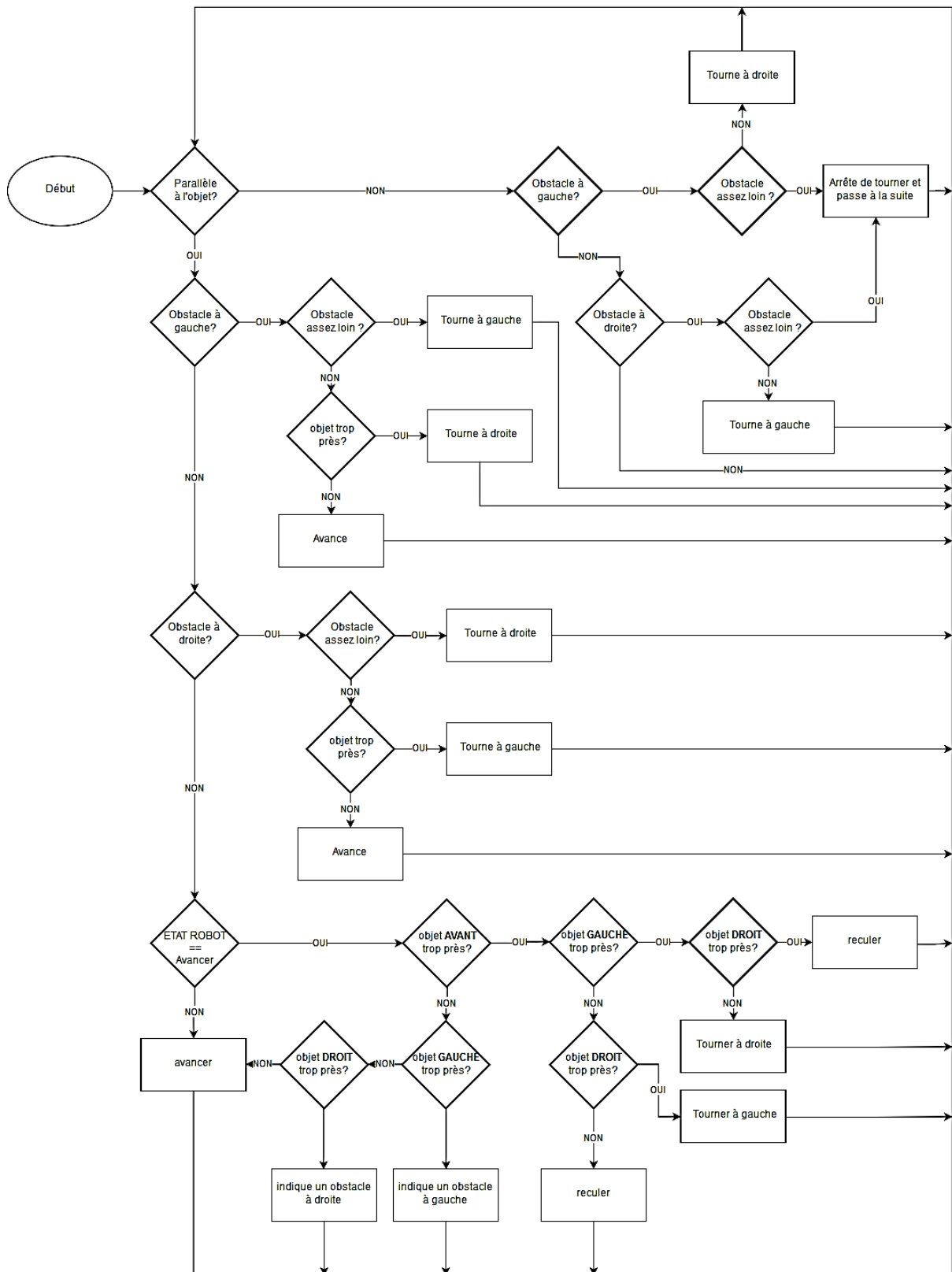


Figure 3 : algorithme V1

Les limites

Cette version, après adaptation à notre robot n'est pas optimale.

En effet, avec notre implémentation on note trois principaux problèmes :

- Pas de marche arrière fonctionnelle (dans un cul de sac par exemple)
- Pas de rotations différentes des roues en simultané pour tourner à gauche ou à droite sur place (doit toujours tenir compte de la taille du robot et prévoir une marge de manœuvre)
- Cherche toujours à retrouver un obstacle afin de le longer (un vrai problème dans des culs de sacs)

LIEN VERS LES VIDEOS :

https://drive.google.com/open?id=1KaYvbsnJH1nLz_G5G_DL3mjdzW1SO3

2^{ème} version

Le code

On décide de reprendre le raisonnement de déplacement du début et de ne pas se focaliser sur l'action de longer l'obstacle. On cherche juste à l'éviter et toujours garder une distance de sécurité comme définie ci-dessous :

```
***  
  
const float safetyDistanceFront = 20 ; // toujours être à plus de 20 cm d'un obstacle avant  
const float safetyDistanceRight = 40 ; // toujours être à plus de 40 cm d'un obstacle droit  
const float safetyDistanceLeft = 40 ; // toujours être à plus de 40 cm d'un obstacle gauche  
  
***
```

Le raisonnement est le suivant (figure 4: algorithme v2):

S'il y a un obstacle à l'avant :

- S'arrête
- S'il y a un obstacle à droite
 - S'il y a un obstacle à gauche
 - Fait demi-tour
 - Sinon
 - On tourne à droite
- Sinon
 - On tourne à gauche

Dans ce bloc on stoppe les moteurs à chaque détection d'obstacles afin de lui laisser le temps de bien analyser son environnement avant de prendre une décision.

S'il n'y a pas un obstacle à l'avant :

- Avance
- Regarde s'il y a un obstacle à droite avec une distance < Distance de sécurité
 - Ajuste la trajectoire
- Regarde s'il y a un obstacle à gauche avec une distance < Distance de sécurité
 - Ajuste la trajectoire

Ici on n'arrête pas les moteurs, l'ajustement se fait en continu (le code en [annexe 7: déplacements V2](#))

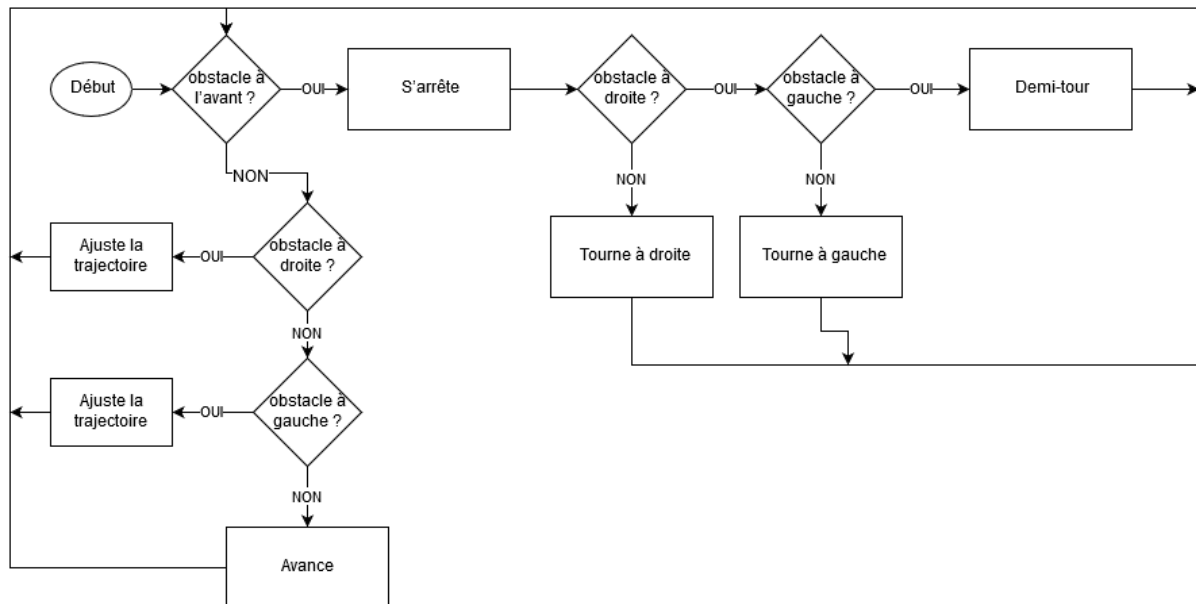


Figure 4: algorithme V2

Ci-dessous figure exemple d'affichage des actions sur le moniteur série :

```

***** ATTENTION DEVANT *****
collision possible à : 6 cm correction de la trajectoire en cours

***** ATTENTION DROITE *****
collision possible à : 38 cm

***** ATTENTION GAUCHE *****
collision possible à : 5 cm faire demi tour

***** ATTENTION DEVANT *****
collision possible à : 9 cm correction de la trajectoire en cours

***** ATTENTION DROITE *****
collision possible à : 5 cm

***** ATTENTION GAUCHE *****
collision possible à : 11 cm faire demi tour
    
```

Les limites

Dans cette version, on note un problème majeur :

- La marche arrière n'est pas maîtrisée du fait qu'il n'y ait pas de capteur arrière pour prévoir ou non la présence d'éventuels obstacles

LIEN VERS LES VIDEOS :

<https://drive.google.com/open?id=1zaU3x7IIWiptBALn8Vb3FeU2IyW26otV>

5- La réception des données

Le module WIFI ESP8266

Branchements

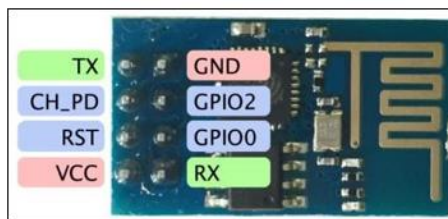


Figure 4 : module Wifi ESP8266

TX /RX : Liaison série mais peuvent également servir de GPIO (transmission / réception)

GPIO2/GPIO0 : Dans ce mode l'ESP8266 ne démarrera pas son programme interne mais entrera dans un mode appelé « UART download mode » où il écrira tout ce qu'il reçoit sur la liaison série, dans sa mémoire. Permet l'injection d'un programme

CH_PD : chip power-down doit être raccordé au 3,3V (fait fonctionner le module)

RST : doit être raccordé au 3.3v. Une pulse à GND sur cette broche génère un reset du composant

VCC : broche d'alimentation 5V

GND : broche d'alimentation 0V

Initialisation et connexion à Arduino

Les commandes AT principales à utiliser pour communiquer avec l'Arduino :

AT : "AT" est une commande de test qui renvoie "OK"

Initialisation du module :

Attribution d'une IP au module :

AT+CIPSTA_DEF=<IP>,<PASSERELLE>,<MASQUE>

Définit le mode de fonctionnement du module en client WiFi :

AT+CWMODE_CUR=1, ou 3 : (1=Client WiFi, 2=Point d'accès, 3= les deux)

Connexion à un point d'accès WIFI :

Connexion au point d'accès wifi :

AT+CWJAP=<NOM POINT ACCES>,<MOT DE PASSE>

Vérifie si la connexion a réussi :

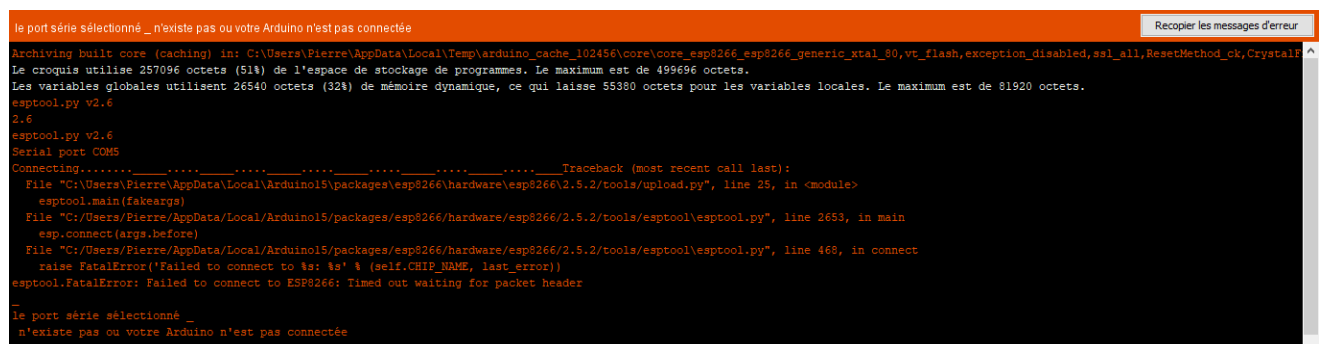
AT+CWJAP=?

Réception des données

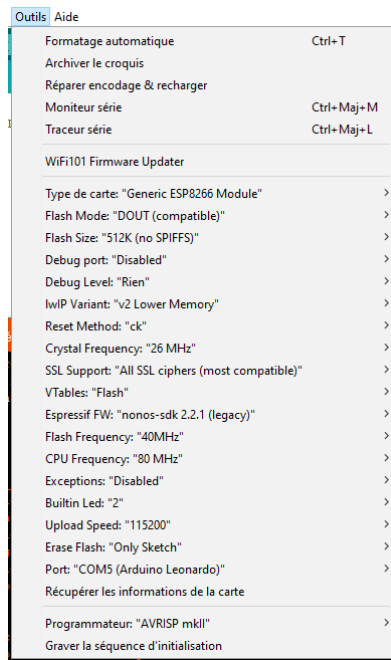
L'utilisation d'un module wifi est intéressante si nous souhaitons envoyer les données à un serveur afin de les traiter en direct comme dans notre cas pour la cartographie.

Les problèmes

L'avantage des cartes Leonardo est qu'elles permettent de faire intermédiaire entre le PC et le module wifi afin de paramétrer ce dernier. Cependant de notre côté malgré les différents tutoriels existant le résultat à ce niveau est toujours le même pour nous, l'IDE ne détecte pas le module wifi à travers la carte Arduino.



```
le port série sélectionné _ n'existe pas ou votre Arduino n'est pas connectée
Recopier les messages d'erreur
Archiving built core (caching) in: C:\Users\Pierre\AppData\Local\Temp\arduino_cache_102456\core\core_esp8266_generic_xtal_80_vt_flash_exception_disabled_ssl_all_ResetMethod_ck_Crystall
Le croquis utilise 257096 octets (51%) de l'espace de stockage de programmes. Le maximum est de 499696 octets.
Les variables globales utilisent 26540 octets (32%) de mémoire dynamique, ce qui laisse 55380 octets pour les variables locales. Le maximum est de 81920 octets.
esptool.py v2.6
2.6
esptool.py v2.6
Serial port COM5
Connecting.....Traceback (most recent call last):
  File "C:\Users\Pierre\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.5.2\tools\upload.py", line 25, in <module>
    esptool.main(fakeargs)
  File "C:\Users\Pierre\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.5.2\tools\esptool\esptool.py", line 2653, in main
    esp.connect(args.before)
  File "C:\Users\Pierre\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.5.2\tools\esptool\esptool.py", line 466, in connect
    raise FatalError("Failed to connect to %s: %s" % (self.CHIP_NAME, last_error))
esptool.FatalError: Failed to connect to ESP8266: Timed out waiting for packet header
_
le port série sélectionné _
n'existe pas ou votre Arduino n'est pas connectée
```



Il existe des solutions ne passant pas par cette étape mais nous nous rendons compte qu'au moment de l'utilisation des commandes AT sur le module ce dernier les reçoit mais ne les traite pas.

Les solutions

Au vu du temps restant nous n'avons pas pu implémenter de solutions mais nous avons déterminé certaines pistes :

- Changer le module WIFI






















En effet le problème pourrait venir du module WIFI car il semble recevoir les commandes AT(LED bleue) mais ne renvoi aucunes réponses ou semble ne pas réussir à les faire.

- Utiliser un module UART

Le problème pourrait sinon venir de la carte Leonardo qui n'arrive pas à faire l'intermédiaire entre l'IDE et le module WIFI. Avoir un module UART permettrait d'envoyer directement les firmwares souhaités sur le module WIFI.

5- Gestion du projet

Ci-dessous un tableau récapitulant les principales étapes du projet et l'implication de chacun :

| | Chloé | Pierre | Yacine |
|---|---|---|---|
| Création des pièces : ➤ Modélisations ➤ Découpage |  |  |  |
| Implémentations |  |  |  |
| Tests |  |  |  |
| Réflexions avant implémentation |  |  |  |
| Cartographie et localisation |  |  |  |
| Réception des données |  |  |  |
| Rédaction et documentation |  |  |  |



Forte implication



Moyenne implication



Aucune implication

6- Conclusion

Dans ce projet, nous avons réalisé un robot explorateur avec Arduino. L'objectif étant de lui permettre de se localiser dans une carte créée en simultanée. Pour ce faire, nous avons à disposition différents composants et des contraintes matérielles.

La réalisation de ce projet a permis d'appréhender le problème « théorie versus réalité » qui nous a freiné un bon nombre de fois. Cependant, être face à ces contraintes nous a incité à avoir un esprit beaucoup plus critique qu'au départ.

Les problèmes rencontrés :

- Débogage nécessaire pour la carte Arduino qui ne supporte pas la boucle de non interruption.
- Distinction entre les pins PWM (Pulse Width Modulation) et normaux sur la carte Arduino
- Connexion par port USB défaillante par moment
- Module wifi esp8266 (voir [Réception des données](#))

À l'issue de cette période de TER nous avons un robot explorateur fonctionnel, et un début de mise en œuvre de la notion de cartographie et de localisation simultanée. Plus précisément la partie perception et association de données évoquée plus haut.

7- Perspectives et réflexions

1- Réflexions

Dans ce projet, nous avons commencé à aborder la notion de cartographie et de localisation simultanée. Une notion assez complexe que nous n'avons pas eu le temps d'implémenter sur notre robot mais que nous simulons sous Matlab comme vous allez le découvrir plus bas.

Principe d'odométrie, Filtre de Kalman et algorithme de SLAM

SLAM (Simultaneous Localization and Mapping) permet, comme son nom l'indique, de construire une carte de l'environnement du robot et d'estimer en permanence la position de celui-ci au sein de la carte.

Pour cela l'algorithme de SLAM doit avoir les informations suivantes :

- L'odométrie
- Des repères

L'odométrie c'est l'estimation de la distance que le robot a parcourue, en mesurant le nombre de tours effectués par chacune de ses roues par exemple.

Avec ces informations nous sommes aptes à définir la trajectoire du robot et les emplacements des repères simultanément sur une carte.

Problèmes :

- L'environnement réel est sujet à des perturbations :
 - Sols glissants
 - Matériaux des obstacles
 - Qualité des capteurs
 - Erreurs de perceptions

Solutions :

- Filtrage pour fusion de capteurs (FFC) ou filtre de Kalman

L'idée est donc de récupérer les données brutes que les capteurs transmettent (par exemple via module wifi [partie 5- réception des données](#)) et ensuite de les utiliser pour prédire les positions futures possibles du robot en tenant compte des observations comme schématisé dans la [figure 3 : localisation et filtre de Kalman](#).

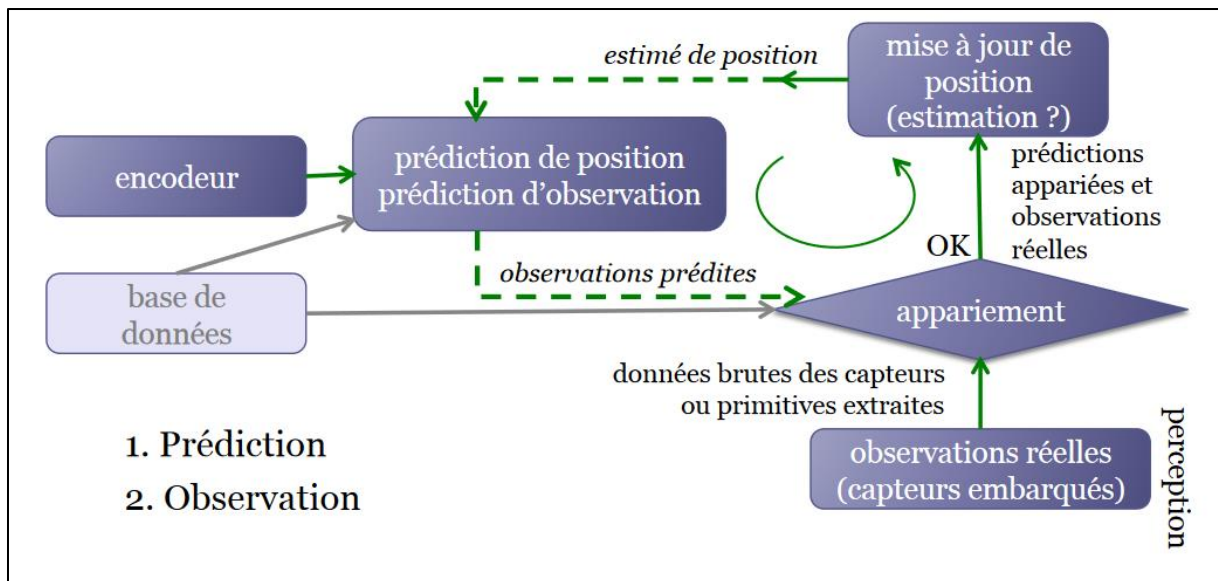


Figure 5: Localisation et Filtre de Kalman

Ci-dessous une modélisation Matlab d'implémentation du principe de localisation par filtre de Kalman.

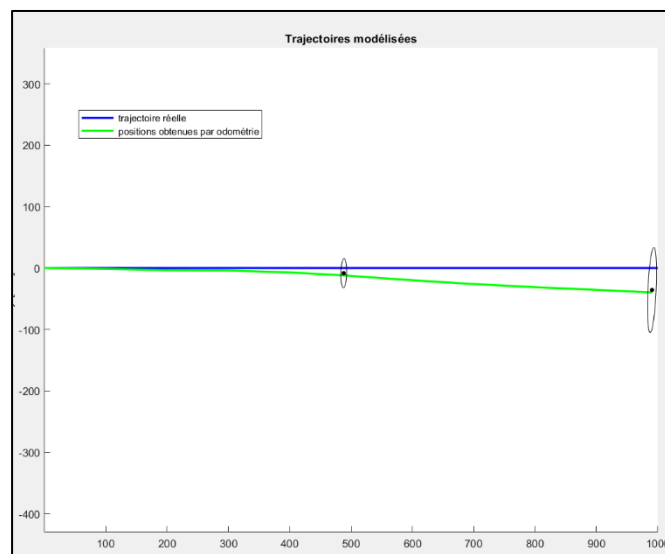


Figure 6: modélisation Matlab

On remarque que la trajectoire prédite est similaire à la trajectoire réelle à une marge d'erreur près (ellipses d'incertitudes sur la figure).

2- Perspectives

La notion vue précédemment est selon nous la suite logique du travail réalisé sur le robot explorateur. Ici nous nous focalisons sur des notions très souvent utilisées dans le domaine de la robotique et de l'Intelligence artificielle, comme par exemple pour la construction de voitures autonomes.

Les perspectives d'évolution seraient tout d'abord d'implémenter une version de l'algorithme de SLAM puis de perfectionner le déplacement et la prise de décision du robot en suivant les méthodes de localisation et mapping qui seront mises en place.

ANNEXE 1 : POSITION DES CAPTEURS

POSITION DES CAPTEURS

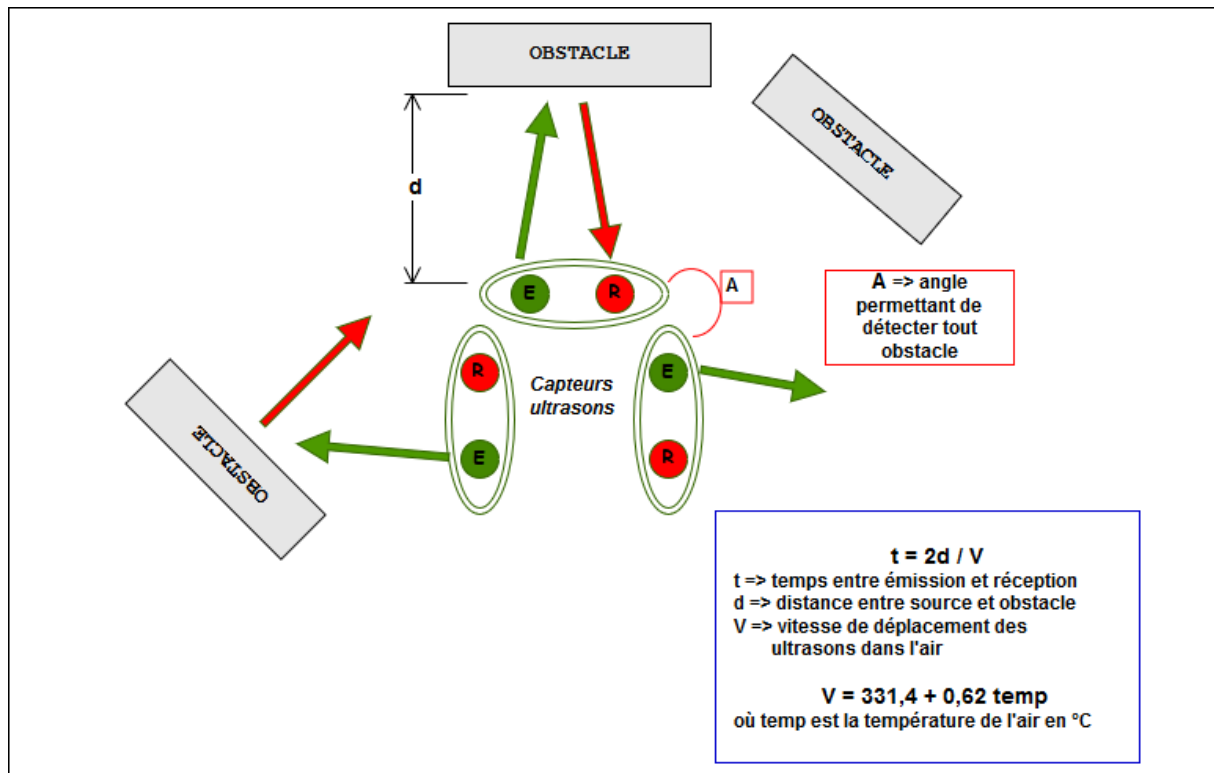
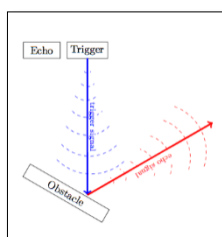


Figure de réflexion 1 : Position des trois capteurs ultrasons

La principale réflexion ici est de connaître exactement l'angle de détection des capteurs afin de connaître l'angle A, d'estimer la distance maximale entre l'obstacle et le robot pour être détecté et enfin de prévoir une distance minimale entre l'obstacle et le robot pour que celui-ci puisse se dévier sans avoir à reculer et/ou entrer en collision.

Dans cette configuration il n'y a pas d'interférence possible entre les capteurs ultrasons, mais la position des obstacles est à prendre en considération. En effet, si un obstacle se trouve à plus de 30° du capteur il émettra un signal mais ne recevra pas de réponse. Comme expliqué ci-dessous extrait de <https://macduino.blogspot.com/2013/11/HC-SR04-part1.html>.

Le son est une onde longitudinale (c'est-à-dire qu'il progresse le long d'une ligne horizontale). Par conséquent, lorsque l'obstacle n'est pas parfaitement devant le module, les sons sont déviés et le signal d'écho risque de ne pas atteindre le capteur ni de l'atteindre très atténué et donc de ne pas être détecté (voir la figure ci-dessous).



Lorsque l'obstacle est à un angle supérieur à 30° par rapport à la direction de propagation du signal de déclenchement, le capteur donne des résultats erratiques.

Par conséquent, notre premier prototype se référera aux chiffres issus des éléments fournis par le constructeur en tenant compte des tests réalisés par des chercheurs.

LE CAPTEUR HC-SR04

Caractéristiques :

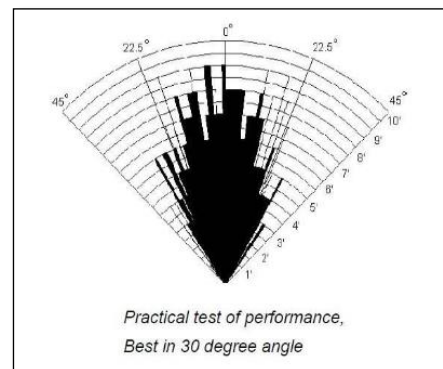
Dimensions : 45 mm x 20 mm x 15 mm

Plage de mesure : 2 cm à 400 cm

Résolution de mesure : 0.3 cm

Largeur d'impulsion sur l'entrée de déclenchement : 10 μ s (Trigger Input Pulse width)

Angle de mesure efficace : 15 ° (de chaque côté) donc 30°
entre chaque capteur



Broches de connexion :

Vcc= Alimentation +5 V DC

Trig= Entrée de déclenchement de la mesure (Trigger input)

Echo= Sortie de mesure donnée en écho (Echo output)

GND= Masse de l'alimentation

En utilisant ces données nous rejoignons la réflexion retrouvée ici :

<https://macduino.blogspot.com/2013/11/hc-sr04-using-multiple-ultrasonic.html> . Ainsi nous pouvons nous baser sur les calculs de Mr Formenti pour définir la position exacte de nos capteurs selon le schéma suivant :

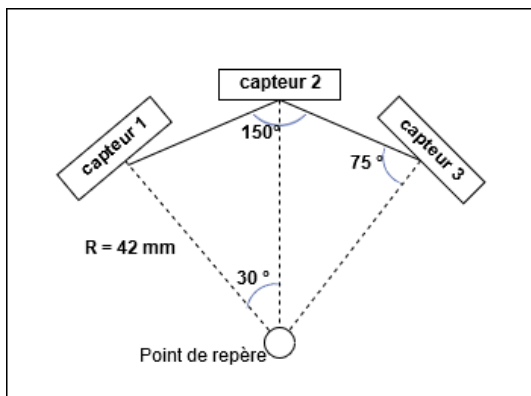
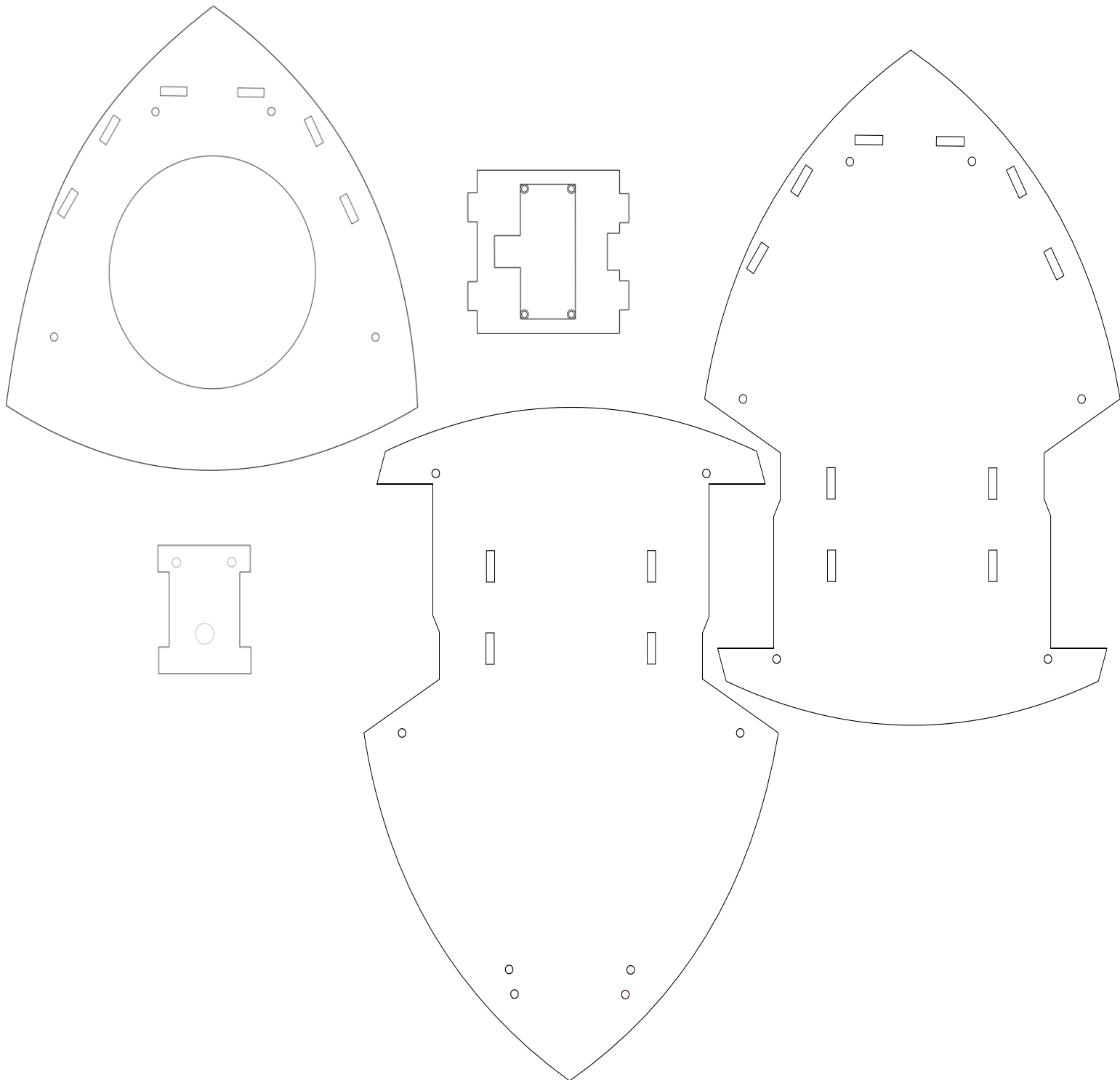


Figure de réflexion 2 : Position des trois capteurs ultrasons

ANNEXE 2 : PIECES INKSCAPE



https://github.com/ChloeMaccarinelli/TER_M1_S2/tree/master/maquettes_inkscape

ANNEXE 3 : TEST 1 CAPTEUR

https://github.com/ChloeMaccarinelli/TER_M1_S2/blob/master/ardunio/calculDistance/calculDistance.ino

ANNEXE 4 : TEST 3 CAPTEURS

https://github.com/ChloeMaccarinelli/TER_M1_S2/blob/master/ardunio/calculDistanceComplet/calculDistanceComplet.ino

ANNEXE 5 : MOTEURS ET CAPTEURS

https://github.com/ChloeMaccarinelli/TER_M1_S2/blob/master/ardunio/motor_capt/motor_capt.ino

ANNEXE 6 : DEPLACEMENTS V1

https://github.com/ChloeMaccarinelli/TER_M1_S2/blob/master/ardunio/deplacements/deplacements.ino

ANNEXE 7 : DEPLACEMENTS V2

https://github.com/ChloeMaccarinelli/TER_M1_S2/blob/master/ardunio/detecteurObstaclesVersion1.0/detecteurObstaclesVersion1.0.ino