

Rabire HABDECHE-HAKIM  
T2D1

# **MEDUSA**

## **DOSSIER TECHNIQUE**

### **BAC STI2D - SIN**



Sommaire :

I - Présentation générale

- 1 - Introduction
- 2 - Présentation du projet
- 3 - Ma partie du travail

II - Définition du besoin

- 1 - Mise en situation
- 2 - Analyse des contraintes

III - Analyse du système

- 1 - Diagramme de cas d'utilisation
- 2 - Schémas et étude des liaisons
- 3 – Communication

IV – Matériels utilisés

- 1 - Choix des technologies
- 2 - Choix des matériels
- 3 - Chaîne d'information et d'énergie

V - Expérimentation

- 1 – Programmation
  - 1.1 – Application
  - 1.2 – Serveur
  - 1.3 – Robot
- 2 - Problèmes rencontrés

## **I - Présentation générale**

### **1 - Introduction**

Lors du projet de spécialité en STI2D, nous devons proposer un projet en groupe, en rédigeant un cahier des charges répondant à diverses problématiques. Nous devons nous positionner dans le présent et dans le futur pour mener à bien ce projet technologique jusqu'au bout. Le travail est partagé dans le groupe. Et nous devons réaliser en pratique notre partie du travail.

### **2 - Présentation du projet**

Le projet Medusa consiste à créer un bateau-robot-aspirateur qui nettoie les déchets à la surface de l'eau des ports. Ce robot est capable de faire son travail de manière autonome.

Sur son passage, le robot ramasse les déchets à l'aide d'un tapis roulant et les accumule dans un bac. Les déchets sont analysés. Le robot prélève aussi une quantité d'eau qui est examinée. Un site web affiche la position en temps réel du robot ainsi que des informations sur les déchets et l'eau.

L'intervention humaine est nécessaire qu'au moment de vider les bacs et recharger les batteries. Le robot peut être piloté à l'aide d'une application mobile.



### **3 – Ma partie du travail**

Le robot Medusa propose une solution de nettoyage automatique. Pour cela, je me suis occupé des déplacements du robot.



Le robot se dirige à l'aide d'un propulseur monté sur un servomoteur. C'est un moyen inspiré des pod sur les bateaux. C'est un élément qui sert à la propulsion maritime. Il peut également contribuer au contrôle de la direction des bateaux.

Le robot analyse les distances autour de lui grâce à un capteur de distance.

Il se déplace sans fil d'abord grâce aux batteries embarquées, mais il est aussi contrôlé à distance en wifi à l'aide d'une application mobile sur smartphone.

## **II - Définition du besoin**

### **1 - Mise en situation**

Les ports souffrent d'une grande pollution partout dans le monde. Les déchets qui recouvrent les ports sont un grand problème esthétique et d'hygiène. La faune et la flore souffrent de cette pollution, car de nombreux produits toxiques sont rejetés.

La création du robot médusa est une solution à ce problème, en nettoyant les déchets encombrants à la surface de l'eau. La majorité des déchets flottants sont des déchets plastiques qui peuvent être recyclés. L'utilisation de ce robot représente un gain d'hygiène, de temps et d'énergie humaine.

### **2 - Analyse des contraintes**

Les contraintes représentent toutes les obligations qui s'opposent à la libre réalisation du projet. Elles doivent être prises en compte en trouvant des solutions ou en contournant le problème.

Dans le cadre de la réalisation de ma partie du projet, mes contraintes sont les suivantes :

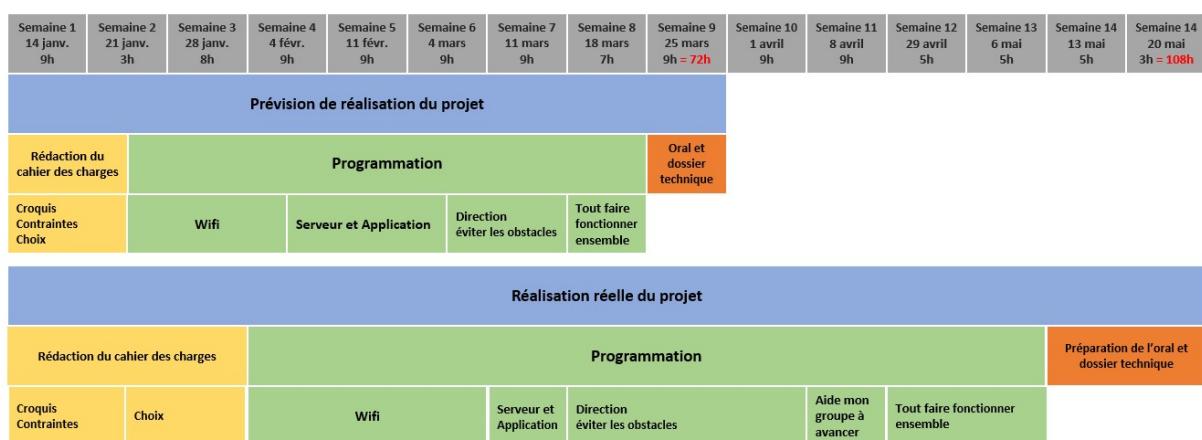
#### **L'étanchéité du robot :**

Dans un premier temps, le robot aspirateur doit être étanche, l'eau ne doit pas être un problème au bon fonctionnement des déplacements du robot. Tous les organes de déplacements du robot sont immersés sous l'eau. Lors de la réalisation de la maquette du projet, les matériels utilisés ne sont pas étanches. Cependant, lors de la réalisation finale du projet, cette contrainte sera prise en compte plus sérieusement.

#### **La contrainte de temps :**

Le projet devait être réalisé en 70 heures. Au final, je rédige ce dossier technique à plus de 80 heures de travail en classe sur ce projet, sans compter les heures de travail à la maison. Ma partie du projet fonctionne correctement en pratique.

Vous trouverez ci-après le diagramme de Gantt du projet :



On constate une grande différence entre les prévisions et la réalisation réelle. Je décrirais plus tard les problèmes rencontrés.

### **Contrainte liée à l'activité maritime :**

Le robot aspirateur a pour obligation de ne pas déranger les diverses activités du port. Il ne doit pas représenter une perturbation pour la vie marine et ne pas troubler les déplacements des bateaux dans le port. C'est ce qui a amener à la réalisation d'un automatisme qui évite les obstacles. C'est au robot de s'adapter à la situation, pas aux autres.

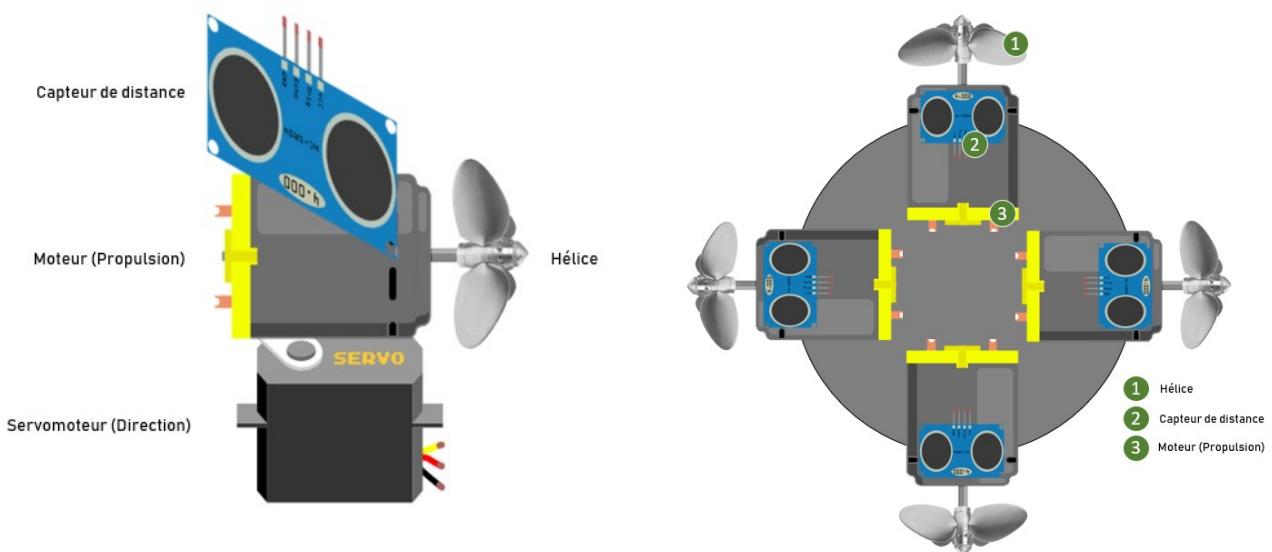
Cela entraîne un effort lié à la sécurité des personnes susceptibles de circuler dans le port.

### **Contraintes environnementale et énergétique :**

Au début de la planification du projet, je voulais créer un robot avec des moteurs et des capteurs de distance de chaque cotés. Les élèves de EE avec qui nous partageons ce projet m'ont contraint à utiliser le moins de moteurs et de composants possible. Cette contrainte est liée à des soucis de consommation et de stockage d'énergie.

Organe de déplacements final du robot :

Ce à quoi je pensais au début :



Le robot utilise 2 « Pod », soit 2 moteurs, 2 servomoteurs et 2 capteurs de distance. Au lieu de 4 moteurs et 4 capteurs de distance.

Un capteur consomme environ 15 mA soit 0.015 A.

Un servomoteur consomme environ 4 A en charge.

Un moteur consomme environ 11 A en charge.

Ici, pour le premier robot, on utilise 2 organes de déplacements en raison d'une course du servomoteur trop restreinte. Si on avait un servomoteur avec une rotation de 360°, un seul aurait suffi.

Consommation Robot 1 =  $2 \times 0.015 + 2 \times 4 + 2 \times 11 = 0.03 + 8 + 22 = 30.03$  A (ou 15.015 A si un seul)

Consommation Robot 2 =  $4 \times 0.015 + 4 \times 11 = 0.06 + 44 = 44.06$  A

### **Respect des normes :**

Le robot fonctionne à l'énergie électrique, c'est une énergie qui rejette très peu de déchets dans la nature. En fonctionnement aucun déchet n'est rejeté (pas de CO<sub>2</sub> par exemple) cela correspond aux normes suivantes :

- 2.1.4.0. - Épandage d'effluents ou de boues
- 2.2.1.0. - Rejet susceptible de modifier le régime des eaux
- 2.2.2.0. - Rejets en mer ou milieux naturels
- 2.2.3.0. - Rejet dans les eaux de surface
- 2.2.4.0. - Installations ou activités à l'origine de déchets toxiques
- 2.3.1.0. - Rejets d'émissions de vapeurs dans l'atmosphère
- 2.3.3.0. - Rejets de déchets composites

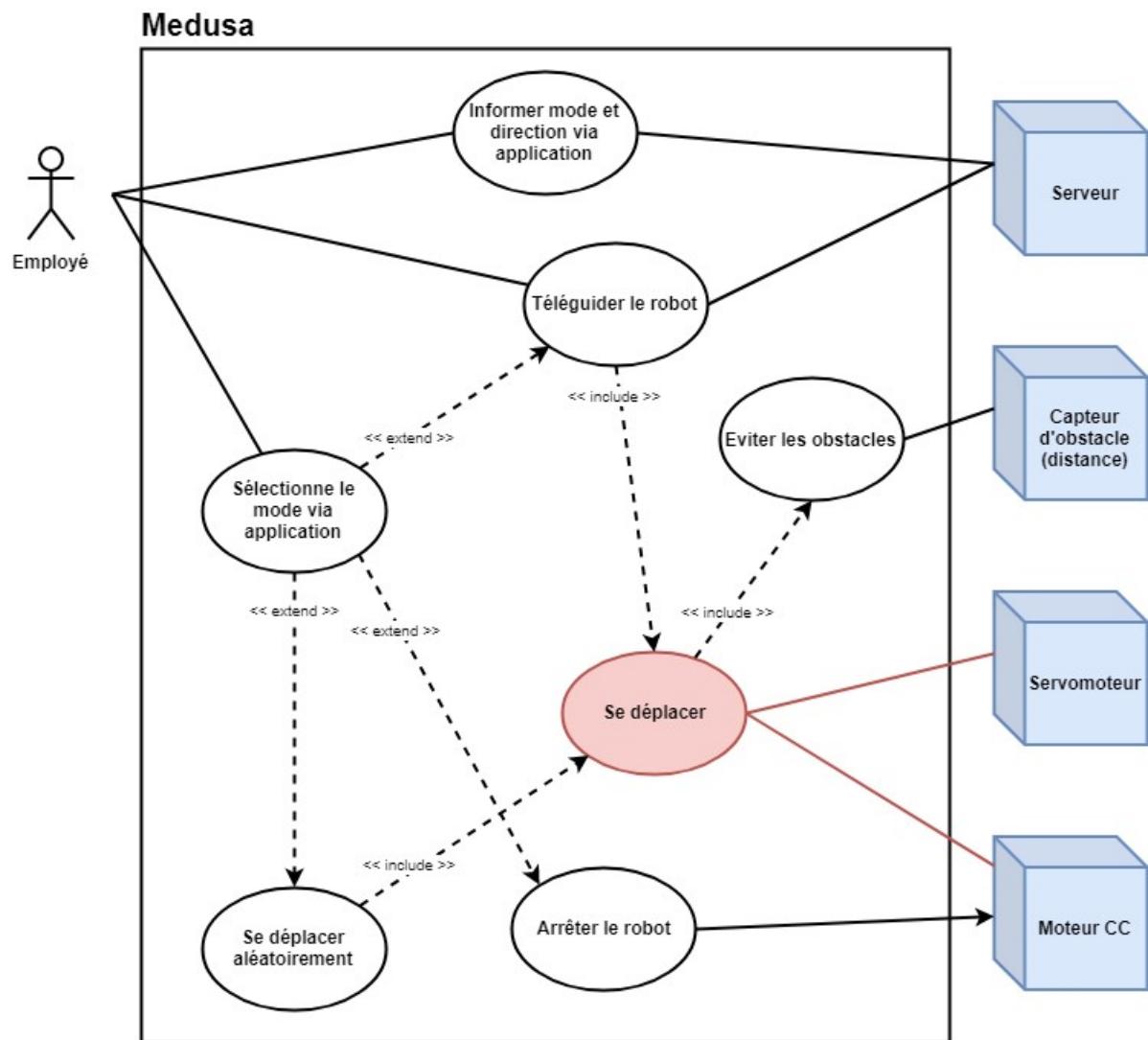


Le projet respecte aussi la directive européenne RoHS de 2002 qui vise à limiter l'utilisation de six substances dangereuses.



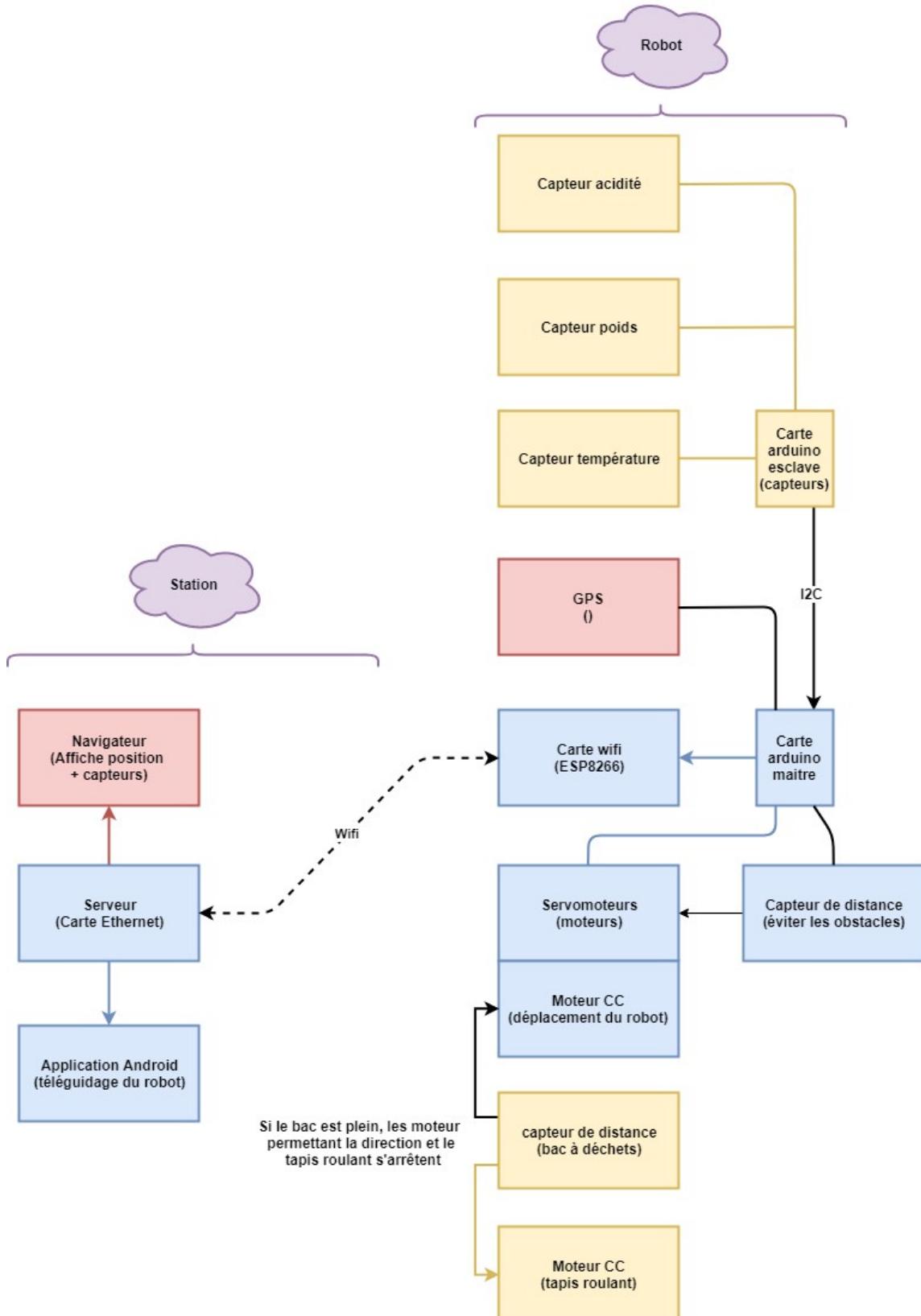
### III - Analyse du système

#### 1 - Diagramme de cas d'utilisation de ma partie

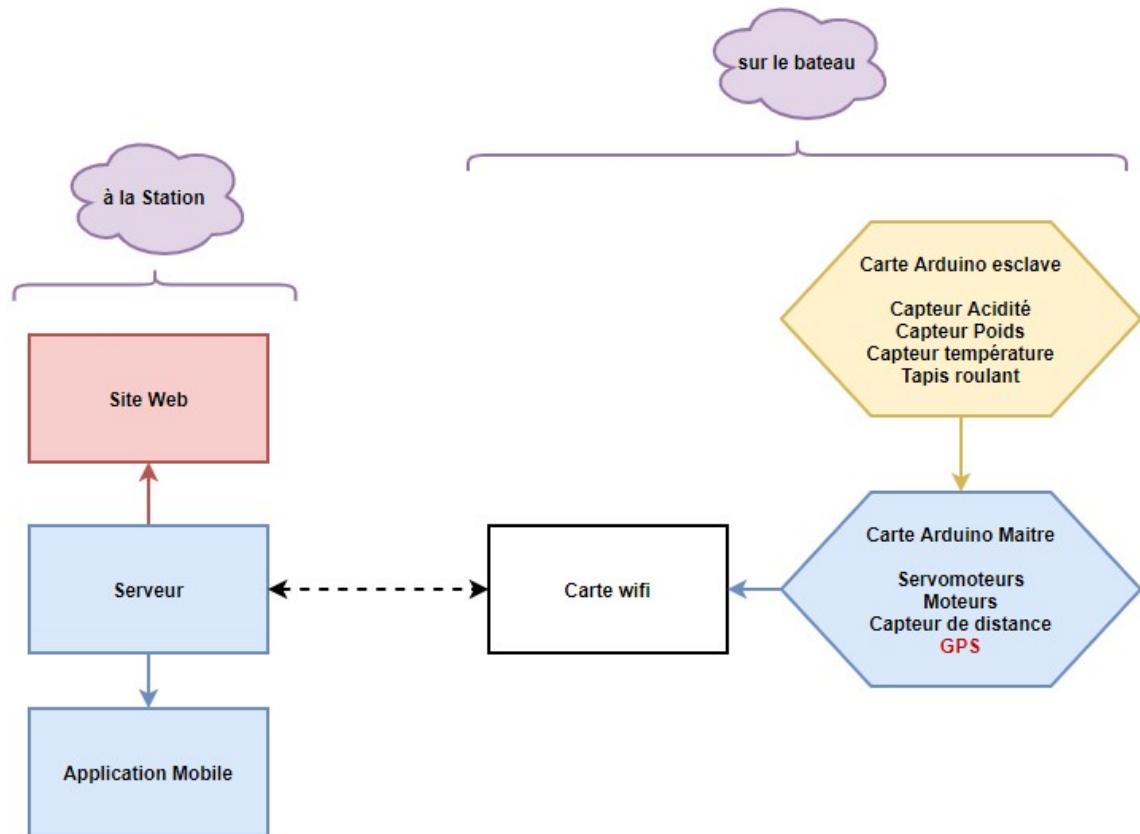


## 2 – Schéma fonctionnel des liaisons

Liaisons détaillées :



## Liaisons simplifiées :



J'ai représenté en bleu ce que j'ai fait.

Le robot, muni d'une carte wifi communique avec un serveur HTML, c'est sur ce dernier que l'application, le site web et même le robot reçoivent et envoient de requêtes.

Les messages qui circulent entre les différentes parties du projet sont les suivantes :

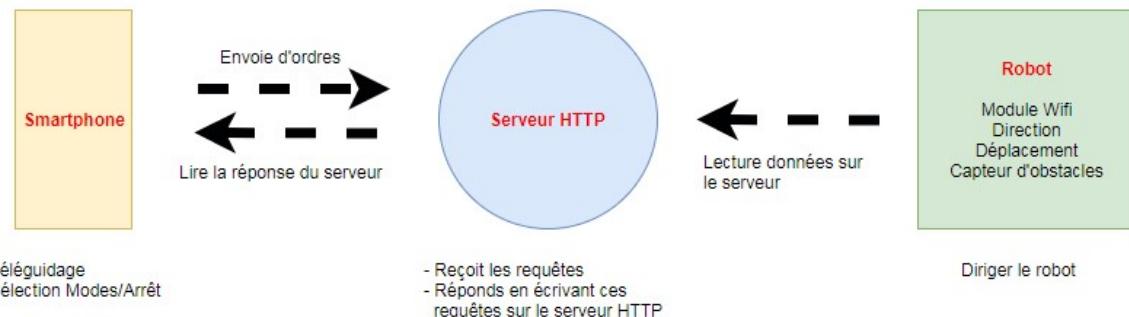
Requêtes envoyées par l'application	Réponses envoyées par le serveur	Interprétation par le robot
appS	direction0	Arrêter le robot
appU	direction1	Diriger le robot en avant
appL	direction2	Diriger le robot à gauche
appR	direction3	Diriger le robot à droite
appD	direction4	Diriger le robot en arrière
ModeAuto	mode5	Mode Automatique
ModeManu	mode6	Mode Manuel
ModeInterrupt	mode7	Mode Interrupteur

Ce tableau aura bien plus de sens quand vous lierez ma partie Expérimentation/Programmation.

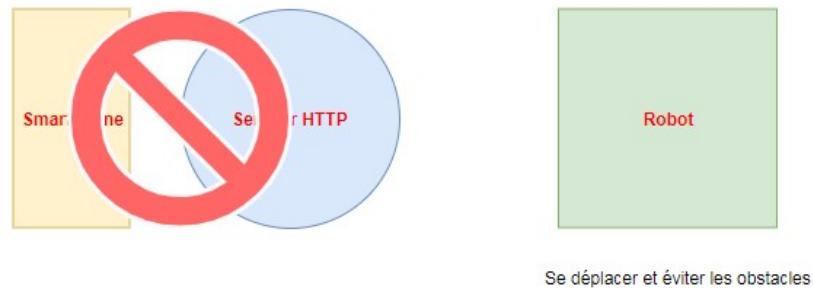
### 3 - Communication

Voici la description des communications entre les différents modes :

#### Mode Manuel :



#### Mode Automatique :



## **IV – Matériels utilisés**

### **1 - Choix des technologies**

#### **Application :**

Pour réaliser l'application mobile, j'ai fait le choix d'utiliser un site : Thunkable.



Thunkable permet de faire des applications simples en programmation graphique (assemblages d'éléments visuels).

Le choix de Thunkable plutôt qu'Android studio s'explique par le fait que Thunkable ne nécessite pas l'installation de logiciels sur l'ordinateur. Seulement une application est nécessaire sur le smartphone qui permet de modifier ou créer une application en direct sans l'installer. À l'inverse d'AppInventor.

Pour le prototypage du projet, seulement les appareils sous Android peuvent être utilisés pour piloter le robot.

#### **Microcontrôleur :**



Pour réaliser le robot, j'ai fait le choix d'utiliser un microcontrôleur Arduino.

Les Cartes Arduino contiennent des dizaines de pin (broches) sur lesquelles peuvent être branchées des entrées et des sorties numériques ou analogiques. Les cartes Arduino possèdent aussi un port vers USB permettant d'injecter les programmes dans la carte. Ce port USB, permet aussi d'écrire des informations dans un moniteur série (valeurs d'un capteur par exemple) intégré dans l'application Arduino sur ordinateur.

Les cartes Arduino peuvent alimenter elles-mêmes les modules branchés dessus. L'énergie électrique est fournie par l'ordinateur grâce au port USB. Cependant, il est préférable d'utiliser une alimentation externe.

Les programmes s'exportent à l'aide d'un logiciel Arduino dans lequel le programme peut être écrit, vérifiés et exportés. Ces programmes sont écrits en C++.



Carte Arduino UNO

## **2 - Choix des matériels**

### **Serveur :**

Pour stocker, acquérir et distribuer des informations, j'utilise un serveur Arduino HTTP.

Le Shield Arduino Ethernet fait office de serveur et d'hébergeur. C'est un serveur physique amplement suffisant pour l'utilisation que je veux en faire.

Ce Shield Ethernet de plug sur une carte Arduino UNO et s'alimente grâce à cette dernière en 5V.



Arduino Ethernet Shield

Voici une illustration des réponses qu'affiche le serveur HTTP sur un site Web :

A screenshot of a web browser window displaying sensor data. The address bar shows the IP address 172.16.13.4/. The page content is as follows:

Mode:	Manuel
Direction:	Droite
PH:	7,8
Température:	20 °C
Masse:	5 kg
Bac:	~50 %
GPS	Lat: 43.2422
	Lon: 5.363339

### Communication sans fil (module Wifi) :

Evidement pour que le robot fonctionne sans fil, il a fallu trouver un moyen de faire interagir le robot avec le site Web et l'Application.

Au début, je voulais utiliser le Bluetooth comme moyen de communication. Mais cela aurait entraîné de nombreux problèmes. Par exemple, aucun moyen direct de communiquer avec Internet n'est intégré au Bluetooth, de plus la portée d'action des modules est trop courte pour un objet qui se déplace dans un port.

Le Wifi peut se relayer si le signal est trop faible et sa portée est suffisante. Le protocole pour le transfert de données est le TCP/IP.

Il a aussi fallu faire le choix entre tous les modules wifi, notamment la carte Arduino UNO Wifi et l'ESP8266.

La carte Arduino UNO WIFI est une carte Arduino UNO basique avec un ESP8266 intégré. Cela semble convenir à l'usage que je veux faire du module, cependant, j'ai fait le choix du module ESP8266 car il est moins cher et permet une plus grande liberté lors de l'utilisation de la carte UNO. Il est important de rappeler que le robot à plein de modules reliés à un même microcontrôleur.



Ce module Emetteur-récepteur permet la communication sans fils en wifi via les commandes AT initialement utilisées pour le Bluetooth :

Basic	
Command	Description
AT	Test AT startup
AT+RST	Restart module
AT+GMR	View version info
AT+GSLP	Enter deep-sleep mode
ATE	AT commands echo or not
AT+RESTORE	Factory Reset
AT+UART	UART configuration, <span style="color:red">[@deprecated]</span>
AT+UART_CUR	UART current configuration
AT+UART_DEF	UART default configuration, save to flash
AT+SLEEP	Sleep mode
AT+RFPOWER	Set maximum value of RF TX Power
AT+RFVDD	Set RF TX Power according to VDD33

Les Caractéristiques principales de l'ESP8266 sont les suivantes :

- Alimentation : 3.3V
- Consommation : 300 mA (c'est une consommation conséquente pour être alimentée par le microcontrôleur)
- Mémoire flash de 1MB (convient particulièrement aux objets connectés)
- Protocole : TCP/IP
- CPU : 32bit
- Portée : 250m (contre 20m maxi avec un module Bluetooth basique)

L'alimentation de 3.3V m'a donner du fil à retordre. La carte Arduino UNO et l'alimentation externe fournissent une tension de 5V. Une surtension risque de casser le module (J'en ai fait l'expérience).

Pour remédier à ce problème, il a fallu mettre en place un pont diviseur de tension :

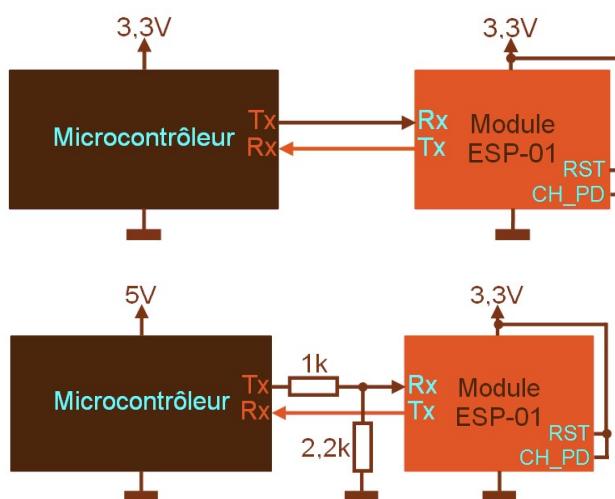
C'est un montage électronique qui permet de réduire une tension à une entrée. Son principe consiste à utiliser des résistances branchées en série. Faut-il encore savoir quelles résistances devaient être utilisées. Pour cela on utilise les lois des mailles et d'Ohm :

On sait que :

$$V_{\text{sortie}} = V_{\text{entrée}} \cdot \frac{R_2}{R_1 + R_2}$$

Pour avoir une tension de sortie de 3.3V, je dois donc utiliser une résistance de 1 kΩ et 2kΩ en théorie. J'ai remplacé la résistance de 2kΩ par une résistance de 2.2kΩ car je n'en n'avais aucune à portée de main. Cela nous fait une tension d'entrée de 3.125 V, le signal minimum recevable par le microcontrôleur Arduino est de 2.5V :

$$3.125 = 5 \cdot \frac{2200}{1000 + 2200}$$

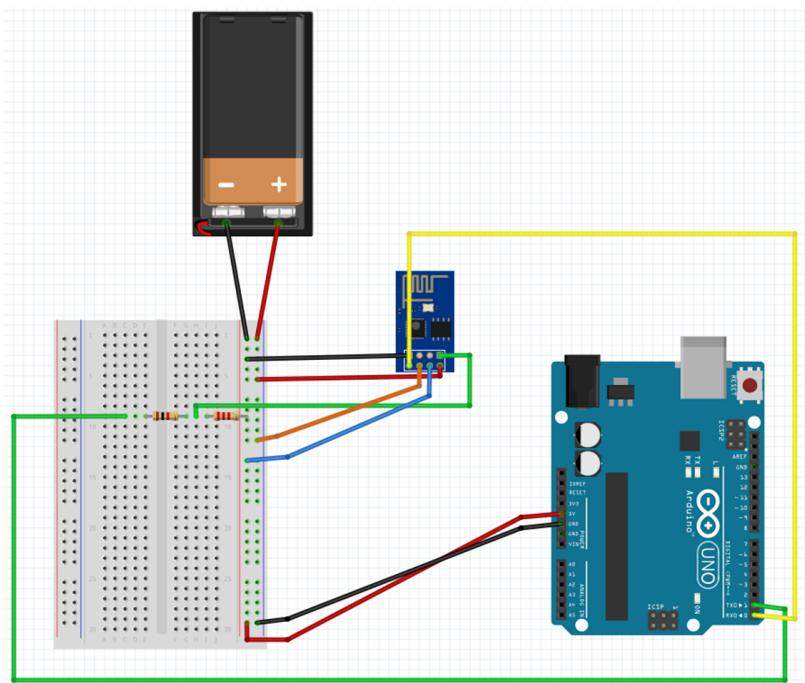


L'ESP8266 est relié au microcontrôleur via les liaisons séries RX et TX. Pour le prototypage, j'ai utilisé une carte Arduino Mega car la carte Arduino UNO nécessitait l'utilisation d'une bibliothèque <SoftwareSerial.h>. Cette dernière n'est pas nécessaire sur la carte Arduino Mega. C'est payer plus cher pour moins de travail, donc à l'avenir j'utiliserais une carte Arduino UNO.

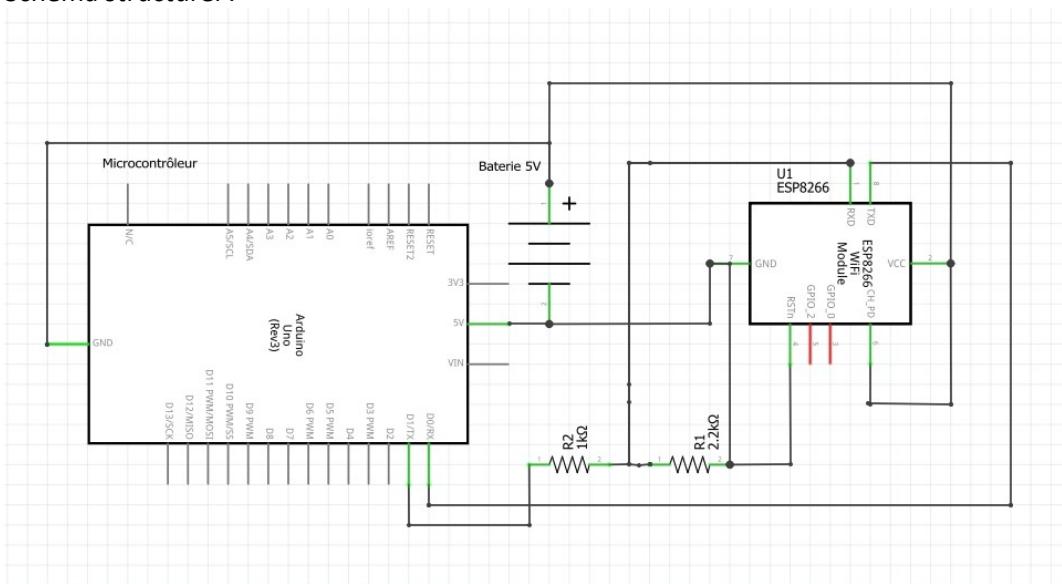


Broches  
ESP8266

## Schéma de câblage :



## Schéma structurel :

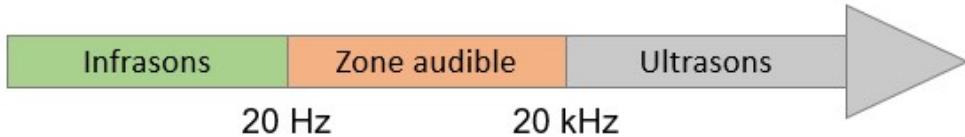


## Capteur de distance :

Pour mesurer les distances autour du robot et éviter les obstacles, J'utilise un capteur à ultrasons (HC-SR04).

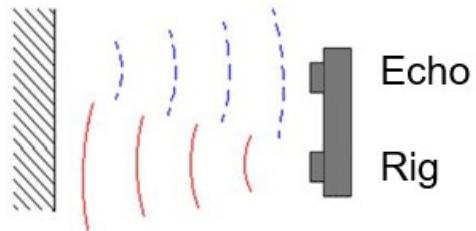
Au début, je voulais utiliser les rayons lumineux infra-rouge pour mesurer les distances. Cela n'a pas été possible car les capteurs infra-rouges ont une portée de moins d'un mètre et les rayonnements du soleil entraînent souvent des dysfonctionnements, surtout quand on l'utilise dans des espaces ouverts comme des ports.

L'ultrason est une onde qui se propage au travers de supports fluides, solides, gazeux ou liquides. La gamme de fréquences des ultrasons se situe au-dessus de 20 000 Hertz. Cette fréquence est trop élevée pour être perçues par l'oreille humaine.



Pour mesurer une distance, un capteur à ultrasons émet à intervalles réguliers de courtes impulsions sonores. Ces impulsions se propagent dans l'air à la vitesse du son. Lorsqu'elles rencontrent un objet, elles se réfléchissent et reviennent sous forme d'écho au capteur. Celui-ci calcule alors la distance le séparant de la cible sur la base du temps écoulé entre l'émission du signal et la réception de l'écho. Le capteur à ultrasons comporte 2 broches permettant cela :

La broche Trig qui est une entrée de déclenchement de la mesure. La broche Echo est la sortie de mesure de données en écho.



Les principales caractéristiques du capteur HC-SR04 sont les suivantes :

- Alimentation : 5V
- Portée : de 6cm à 4m
- Fréquence d'émission d'ondes : 40 kHz
- Signal : analogique
- Angle de mesure efficace : 15°

Pour déclencher une mesure, il faut présenter une impulsion "high" (5 V) d'au moins 10 µs sur l'entrée "Trig". Le capteur émet alors une série de 8 impulsions ultrasoniques à 40 kHz, puis il attend le signal réfléchi. Lorsque celui-ci est détecté, il envoie un signal "high" sur la sortie "Echo", dont la durée est proportionnelle à la distance mesurée.

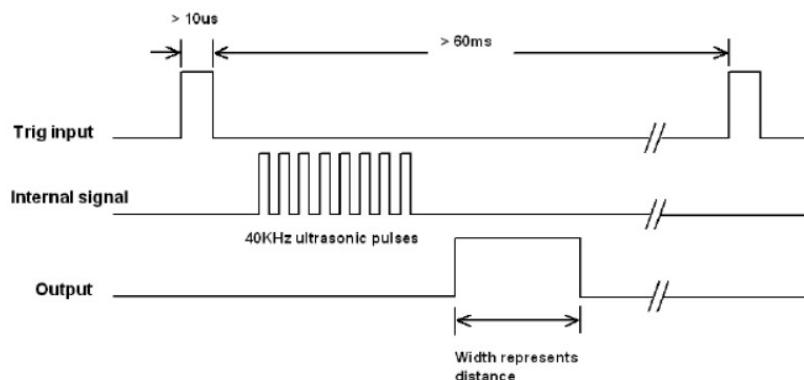


Schéma de câblage :

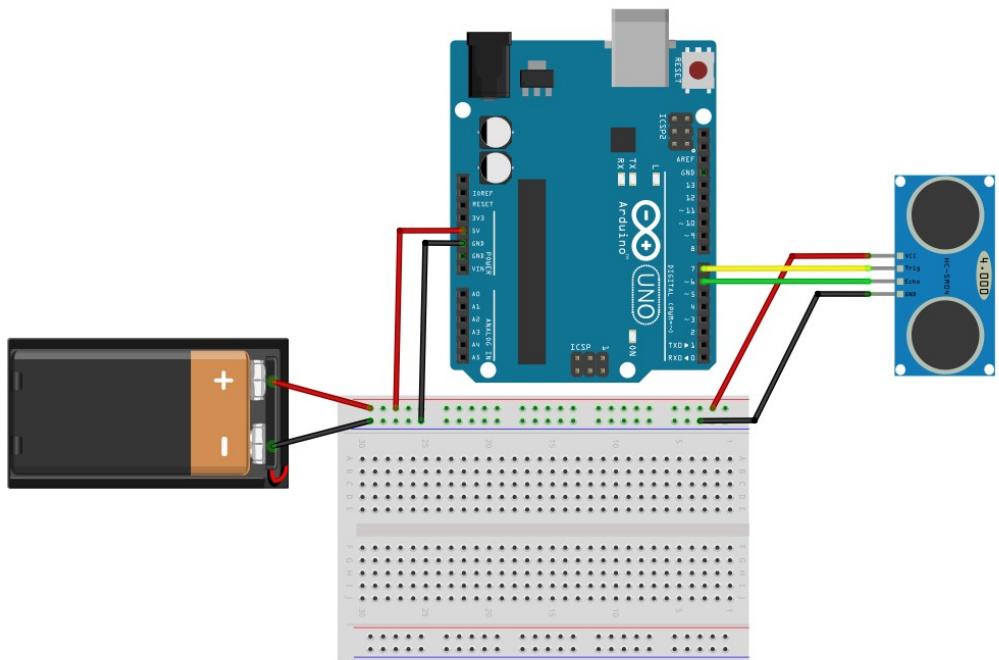
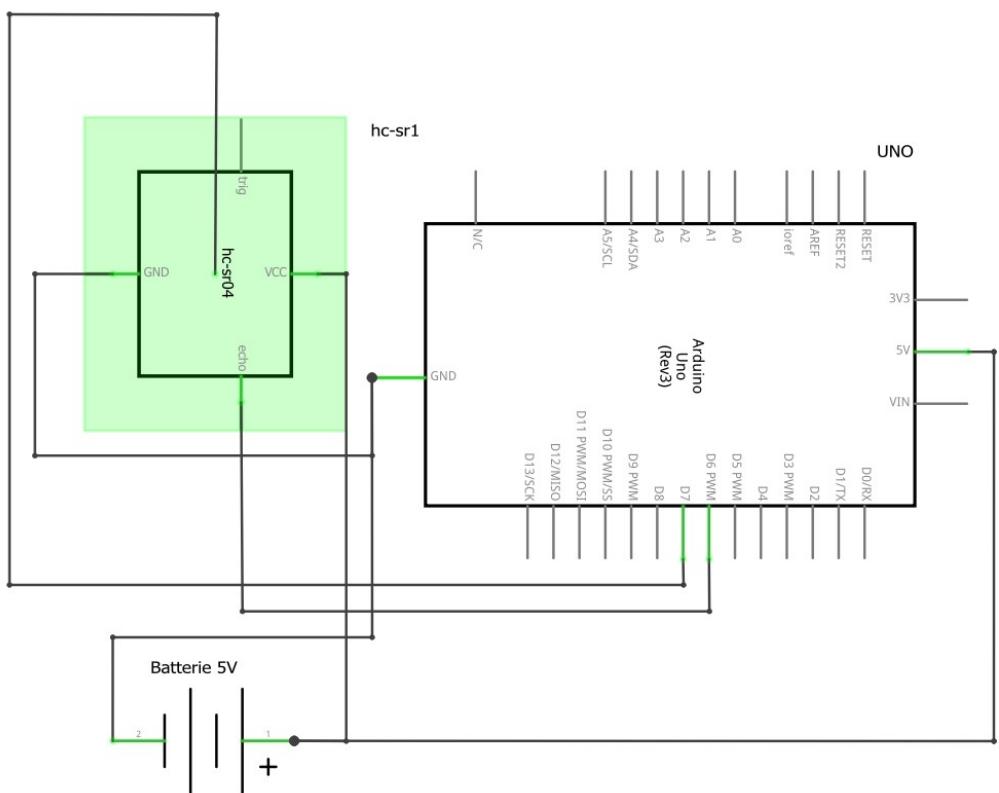


Schéma structurel :



## Direction (Orientation) :

Le robot se dirige à l'aide d'un propulseur monté sur un servomoteur. C'est un moyen inspiré des pod sur les bateaux.

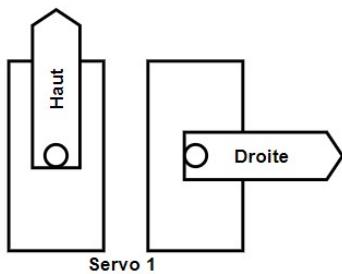
Le servomoteur utilisé est le modèle SG90.

Ses caractéristiques principales sont les suivantes :

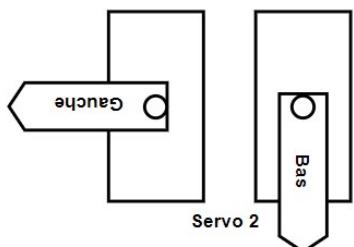
- Alimentation : 4.8 à 6Vcc
- Course : 2x90°
- Couple : 1.6 kg.cm à 5V
- Vitesse de rotation : 0.15s/90°
- Signal : numérique rectangulaire
- Nécessite la bibliothèque <Servo.h>



Servomoteur SG90



Étant donné que je n'avais pas de servomoteur à rotation de 360°, j'en utilise deux de 180°. Chaque servomoteur dirige le robot dans 2 directions chacun :



La longueur du signal à 5V indique l'angle de rotation que doit avoir le servomoteur. Plus le signal est long, plus l'angle de rotation l'est aussi.

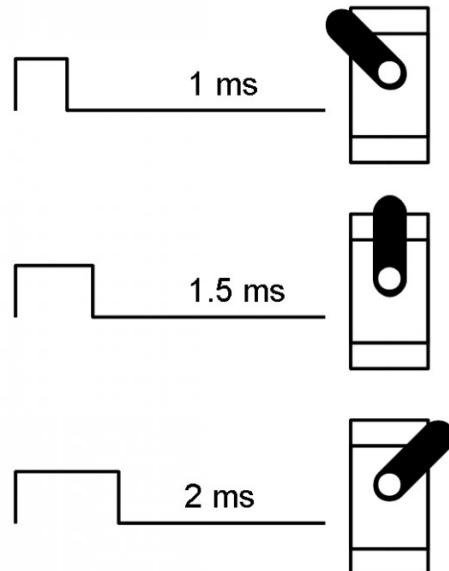


Schéma de câblage :

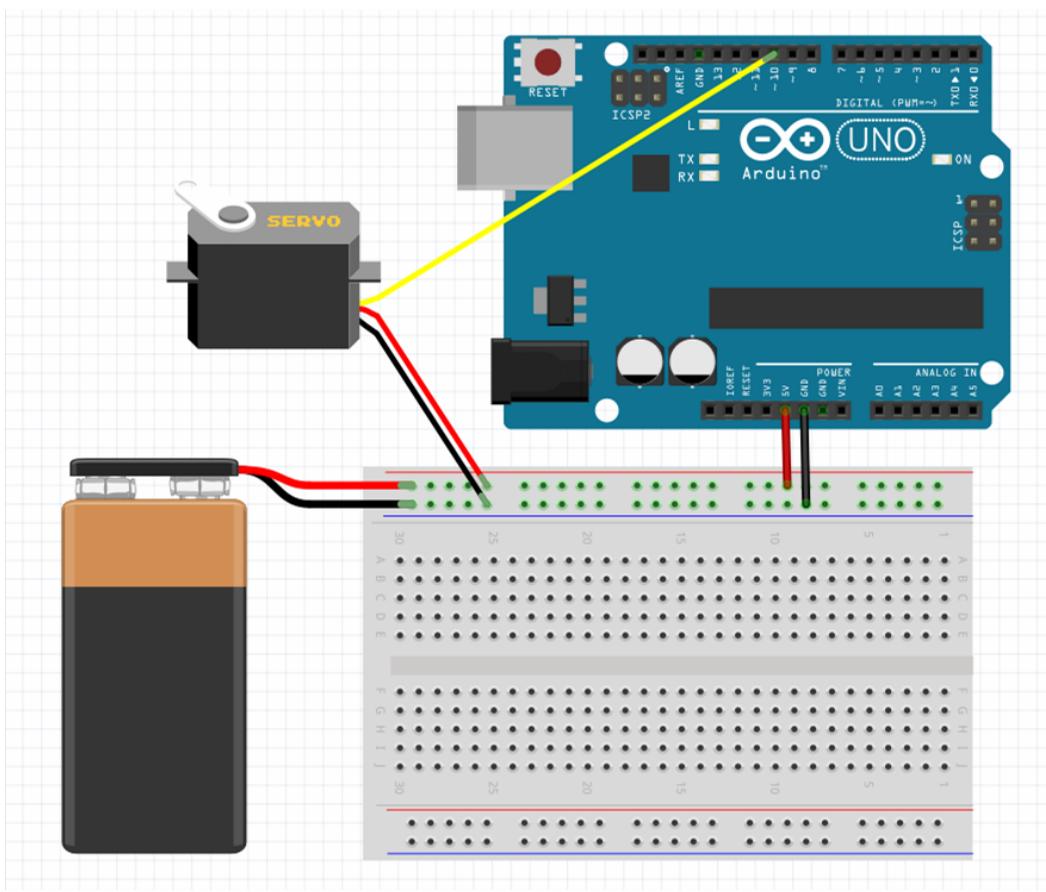
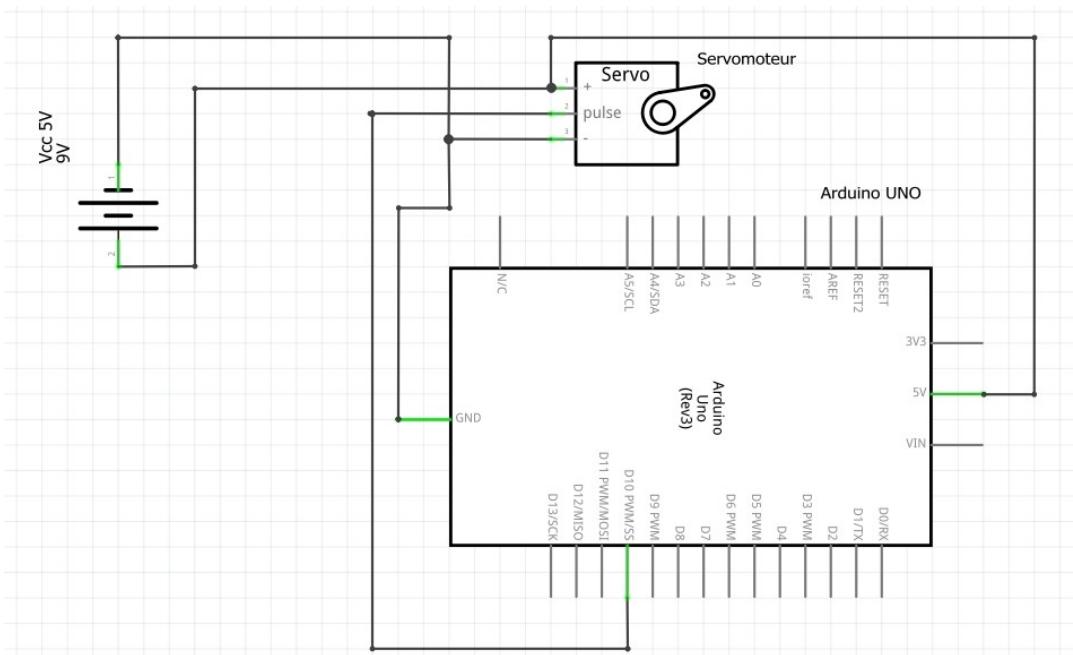


Schéma structurel :



## Propulsion du robot :

Sur le servomoteur présenté ci-avant, est monté un moteur muni d'une hélice qui fait office de propulseur.

Le moteur utilisé est le Moteur CC RE850. C'est un moteur à courant continu sur lequel je ne le fais pas varier la vitesse, cette dernière varie en fonction de la tension qui l'alimente. J'ai fait le choix de l'alimenter avec la même batterie que les autres modules en 5V car la vitesse de rotation est suffisante pour déplacer le robot à allure lente.



Moteur CC  
RE850

Les caractéristiques du moteur CC RE850 sont les suivantes :

- Alimentation max : 12 Vcc
- Consommation : 1.90 A à vide et 10.82 A en charge
- Vitesse de rotation : 9778 t/min à vide et 8311 t/min en charge
- Couple en charge : 92 mN.m (920 g.cm)
- Couple bloqué : 614 mN.m (6.14 kg.cm)

Pour faire tourner ou stopper le robot, il faut l'alimenter ou non le moteur. Pour cela, j'utilise un module relais GT1080.

Dans le programme, il fonctionne un peu comme une LED, c'est une sortie que l'on met à 0 ou 1.

C'est un concept tout ou rien (TOR).

Le module relais s'alimente en 5V et peut couper un courant électrique d'une puissance allant jusqu'à 270 W.

$$P = U \cdot I$$

$$P = 5 \cdot 10.82$$

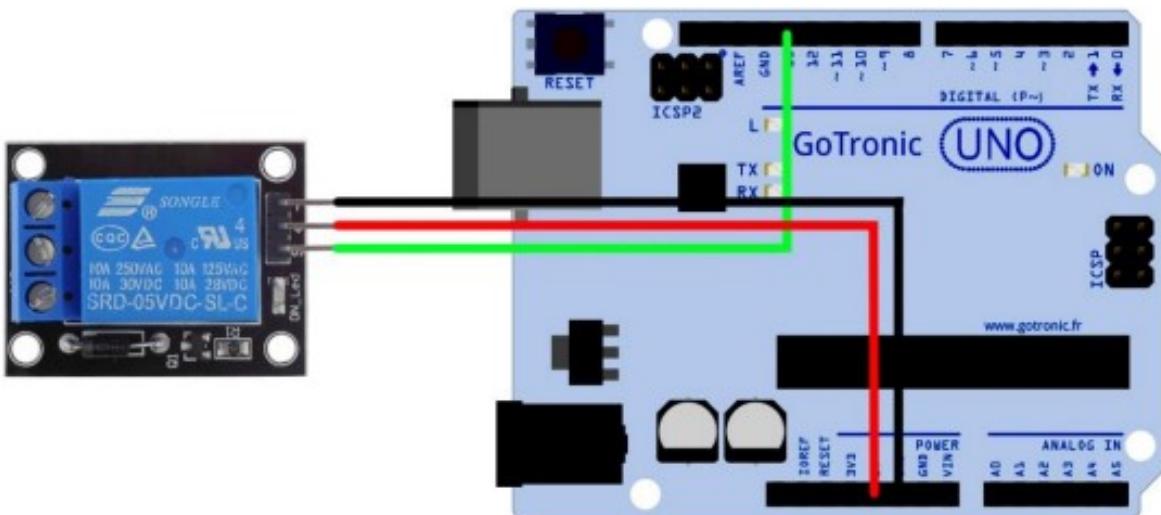
$$P = 54.1 \text{ W}$$

Donc  $54.1 \text{ W} < 270 \text{ W}$

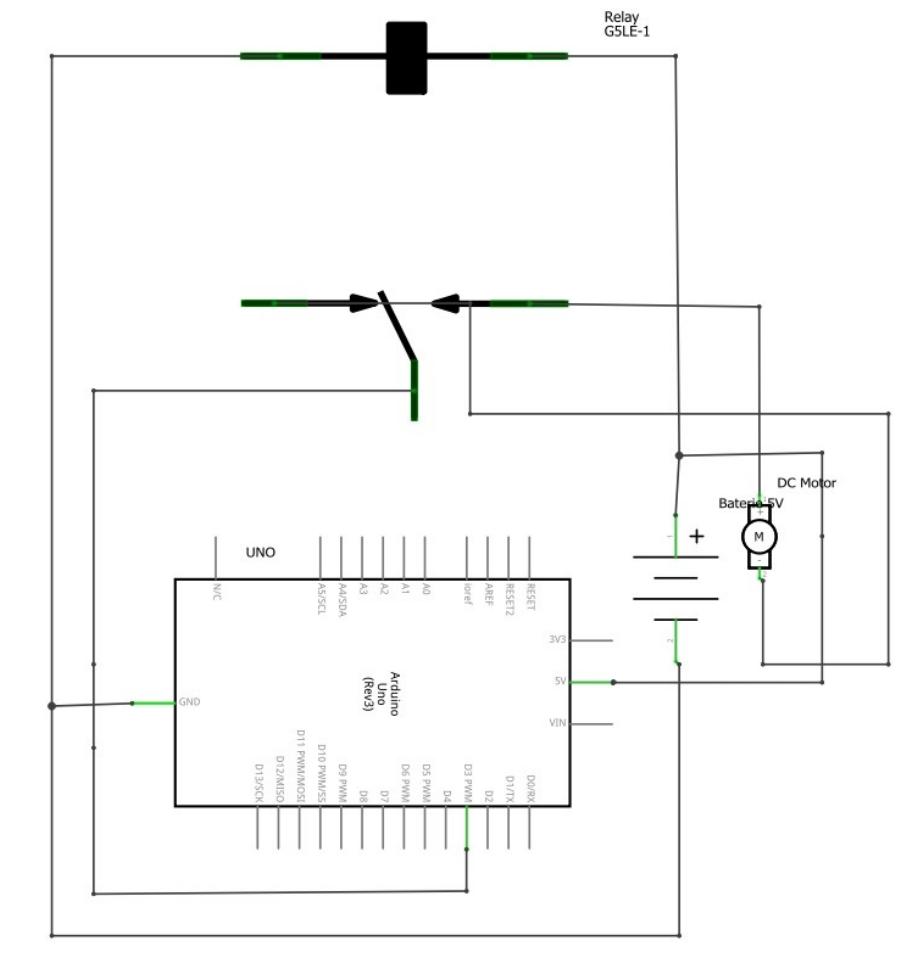


Module relais  
GT1080

## Schéma de câblage :

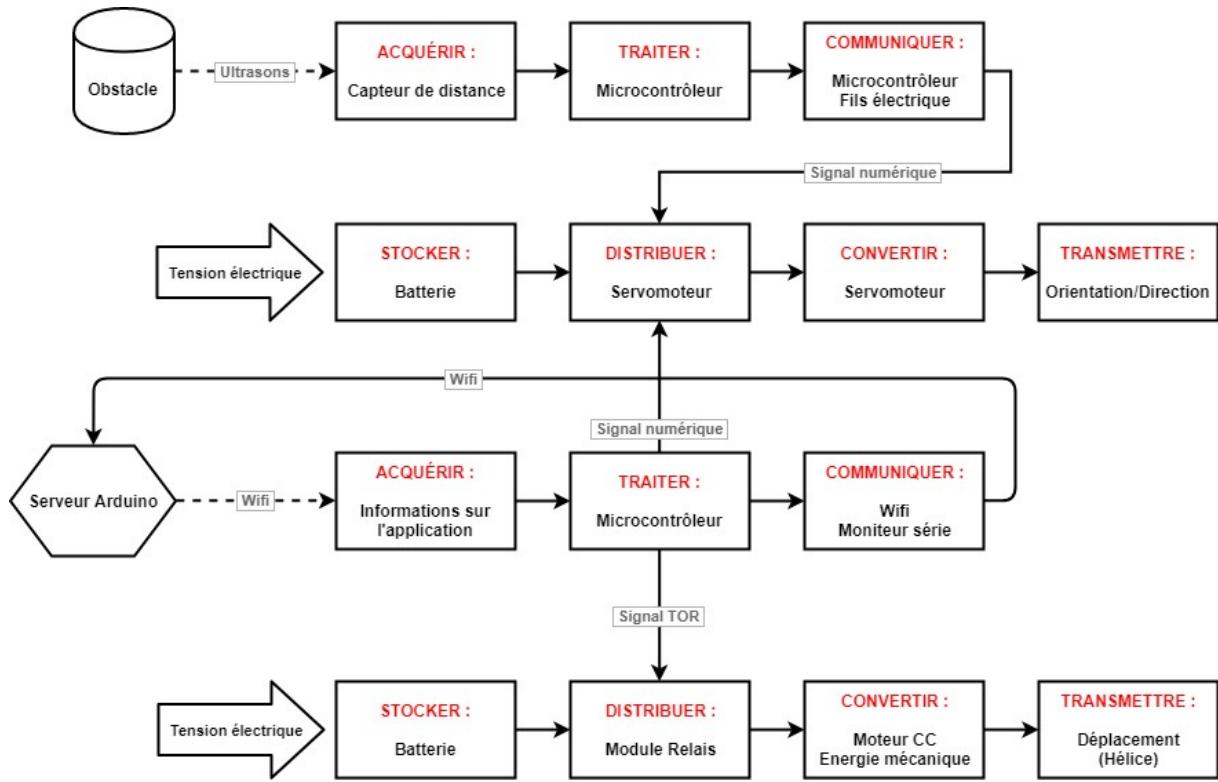


## Schéma structurel :



### **3 – Chaine d'information et d'énergie**

Chaine d'énergie et d'information de ma partie du système :



Sur le robot :

En mode manuel, le robot acquiert la réponse du serveur en Wifi et la traite avec le microcontrôleur. Ce dernier envoie un signal TOR au module relais qui distribue ou non une tension électrique stockée dans une batterie. Quand la tension électrique passe, elle se convertit en énergie mécanique grâce au moteur CC. Cette énergie mécanique est transmise par une hélice et qui permet le déplacement du robot.

En mode manuel et automatique, le robot acquiert la réponse du serveur en Wifi et la traite avec le microcontrôleur. Ce dernier envoie un signal numérique au servomoteur qui oriente le robot. Ce servomoteur distribue et convertit une tension électrique stockée dans une batterie en énergie mécanique. Cette dernière oriente le robot.

En mode automatique, quand un obstacle est détecté trop près à l'aide des ultrasons par le capteur de distance, le microcontrôleur communique une information au servomoteur.

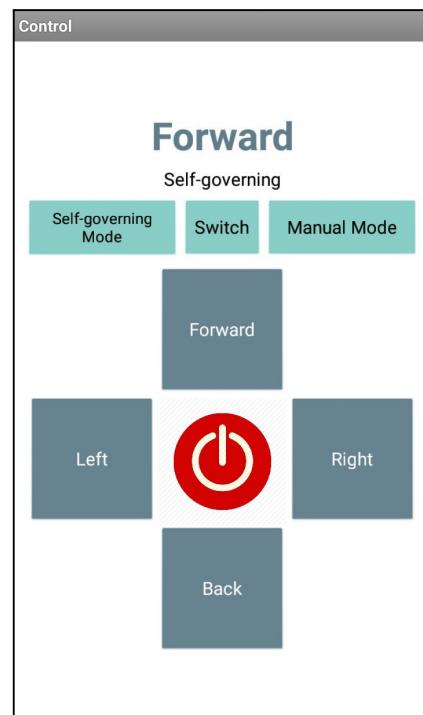
## V – Expérimentation

### 1 – Programmation

*Les programmes sont disponibles sur [github.com/Rabire/MEDUSA/](https://github.com/Rabire/MEDUSA/)*

#### 1.1 - Application :

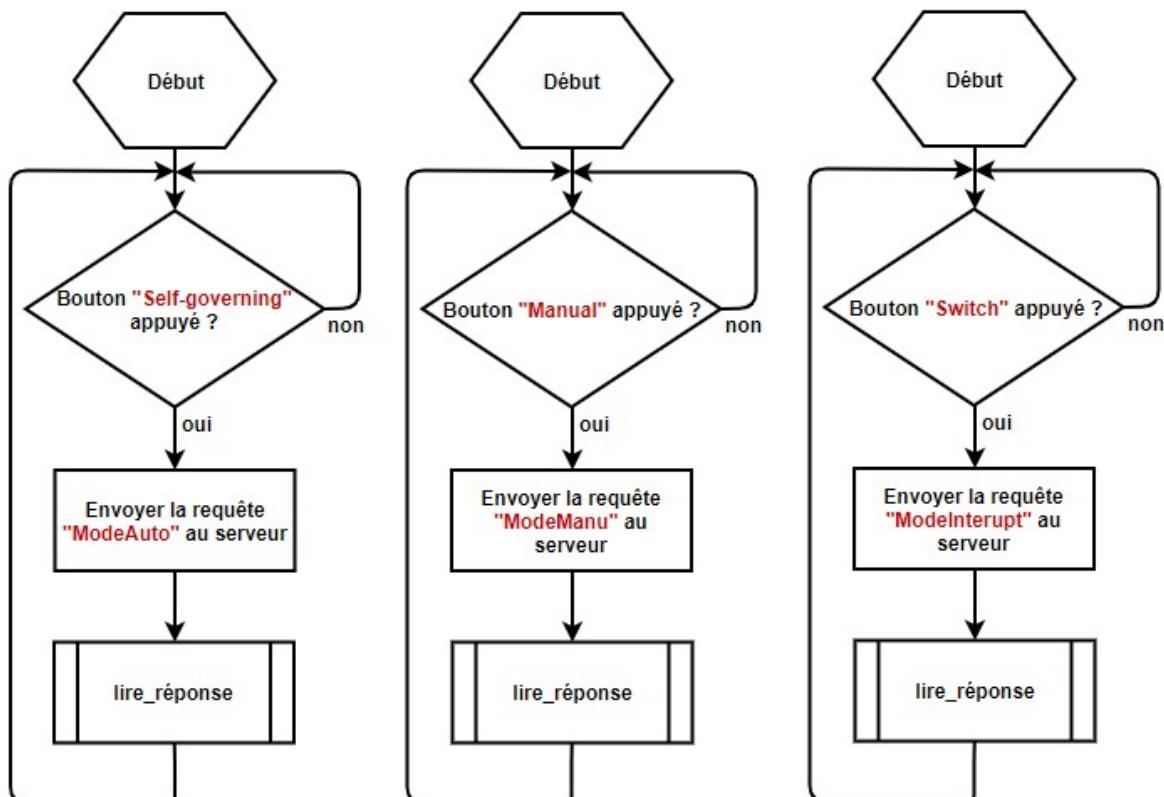
Capture d'écran de l'application finale :



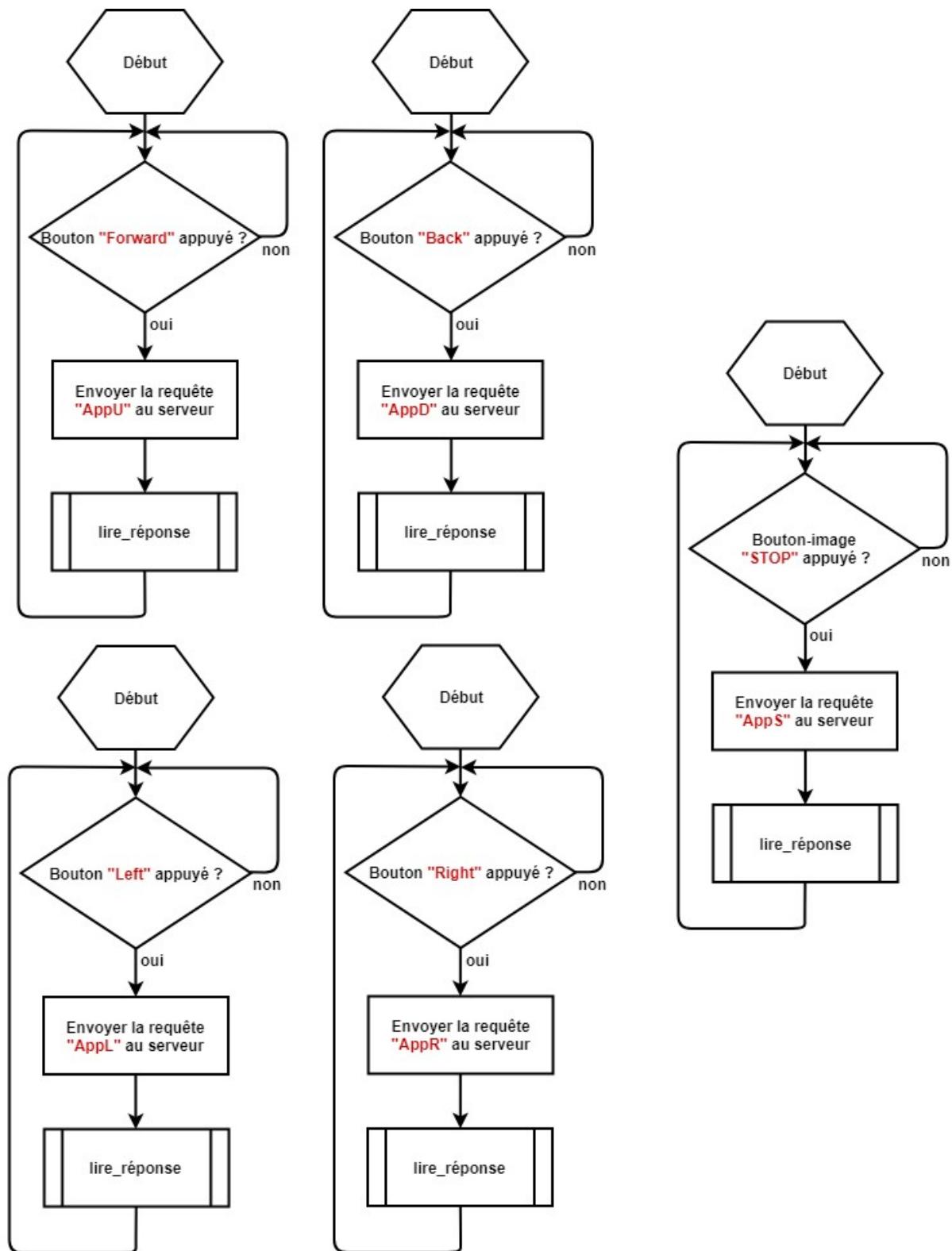
Algorigrammes (très) détaillés de l'application :

Boutons des modes :

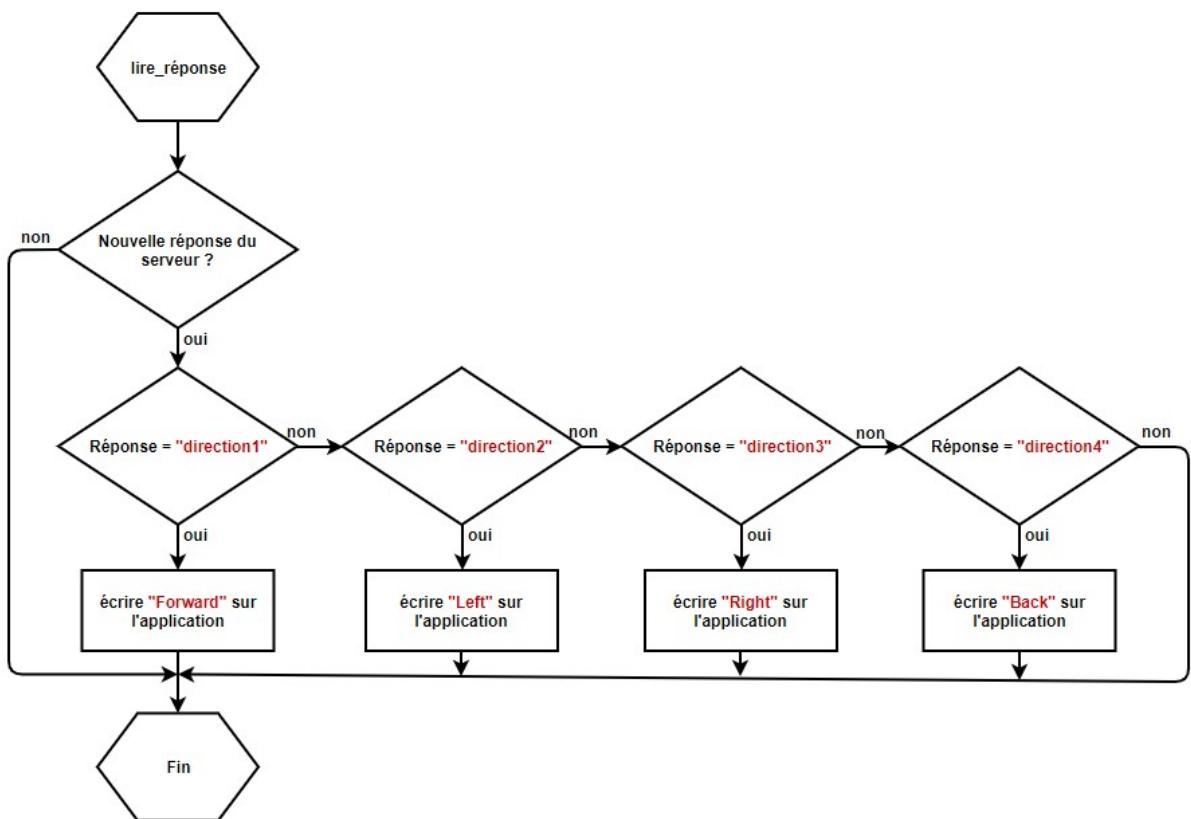
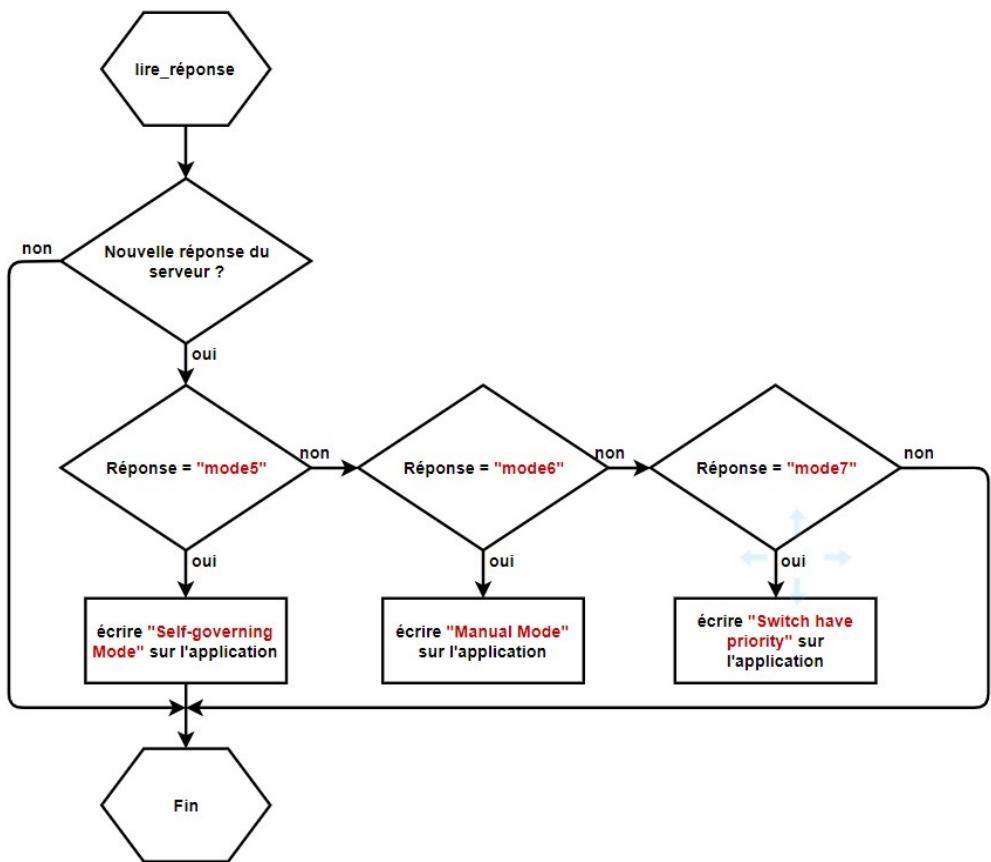
(Cf Block programme 1)



Boutons des directions de téléguidage :  
 (Cf Block programme 2)



Lecture de la réponse du serveur en fonction des boutons appuyés:  
(Cf Block programme 3)



Block programme 1 :

```
when button_selfG .Click
do call Web1 .Post Text
    text " ModeManu "
when button_manual .Click
do call Web1 .Post Text
    text " ModeAuto "
when button_switch .Click
do call Web1 .Post Text
    text " ModelInterup "
```

Block programme 2 :

```
when Button_Left .Click
do call Web1 .Post Text
    text " appL "
when Button_Forward .Click
do call Web1 .Post Text
    text " appU "
when Button_Right .Click
do call Web1 .Post Text
    text " appR "
when Button_Back .Click
do call Web1 .Post Text
    text " appD "
when Button_Stop .Click
do call Web1 .Post Text
    text " appS "
```

Par moments, l'application rencontre une erreur. Cela est souvent lié à un soucis de synchronisation avec le serveur. Cette erreur entraîne un arrêt de l'application ou l'empêche de continuer à fonctionner normalement.

Pour remédier à ce problème, j'anticipe les erreurs en les traitant :

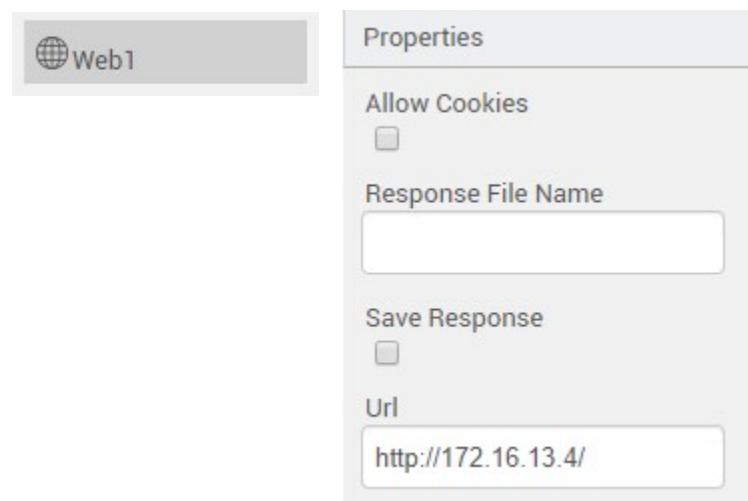
```
when Screen1 .Error Occurred
component functionName errorNumber message
do set Label_Direction .Text to " ERROR "
```

#### Runtime Error

```
/tmp/1517487597203_0.47973805
6892541-0/youngandroidproject/../
src/com/thunkable/android/
saimch965/My_Cash_Valut/
Screen1.yail:59:92: unbound
location null
```

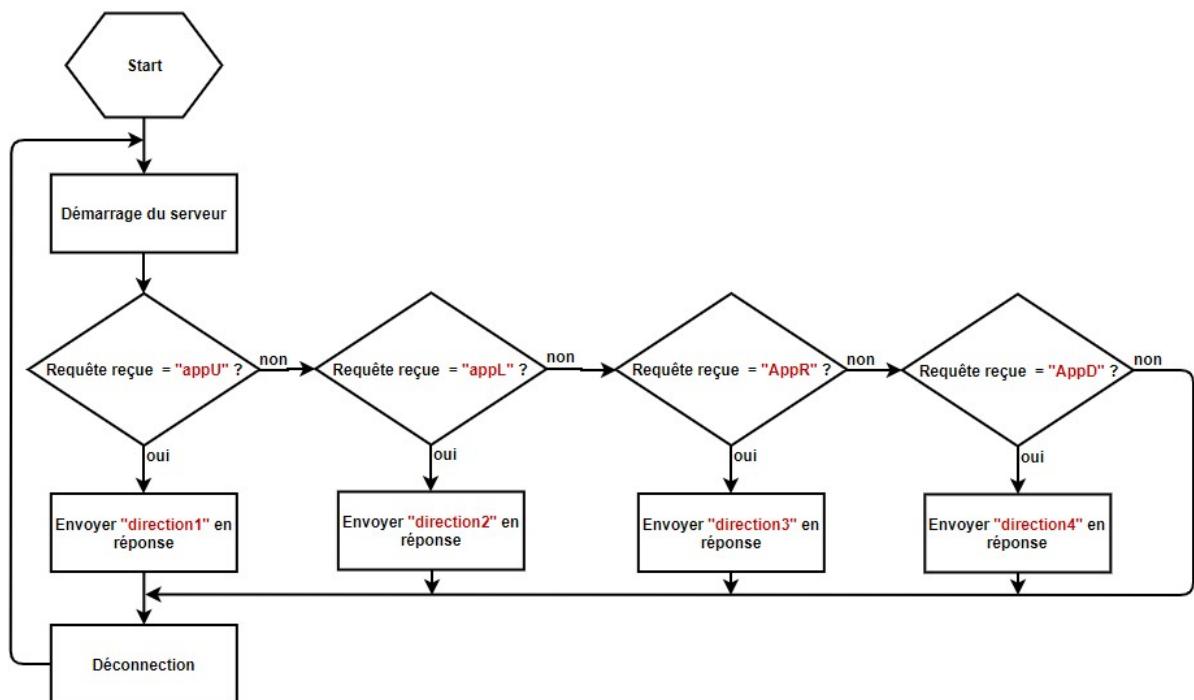
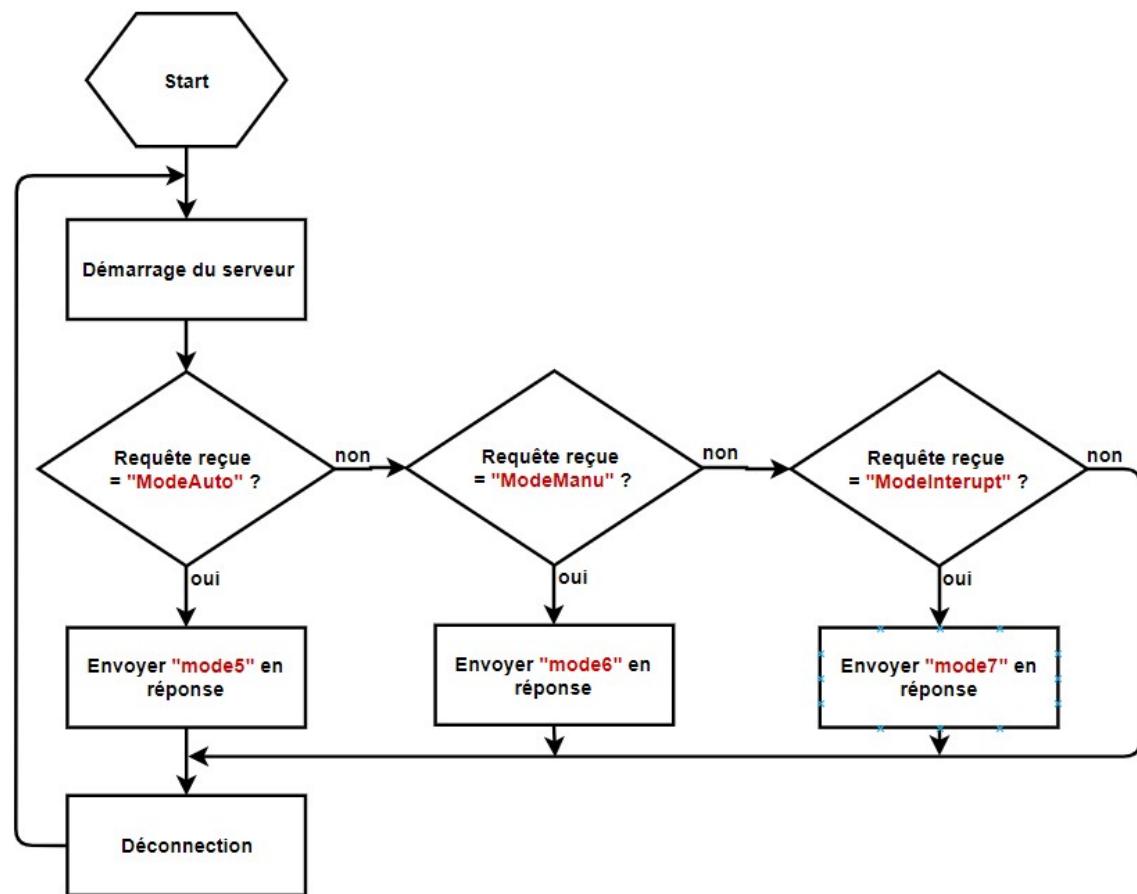
END APPLICATION

Block programme 3 :



## 1.2 - Serveur :

Algorigrammes détaillés du serveur :



Le serveur ne traite que les requêtes qu'il attend. C'est une sorte de filtrage. Cela l'empêche d'écrire n'importe quelle requête qu'il reçoit.

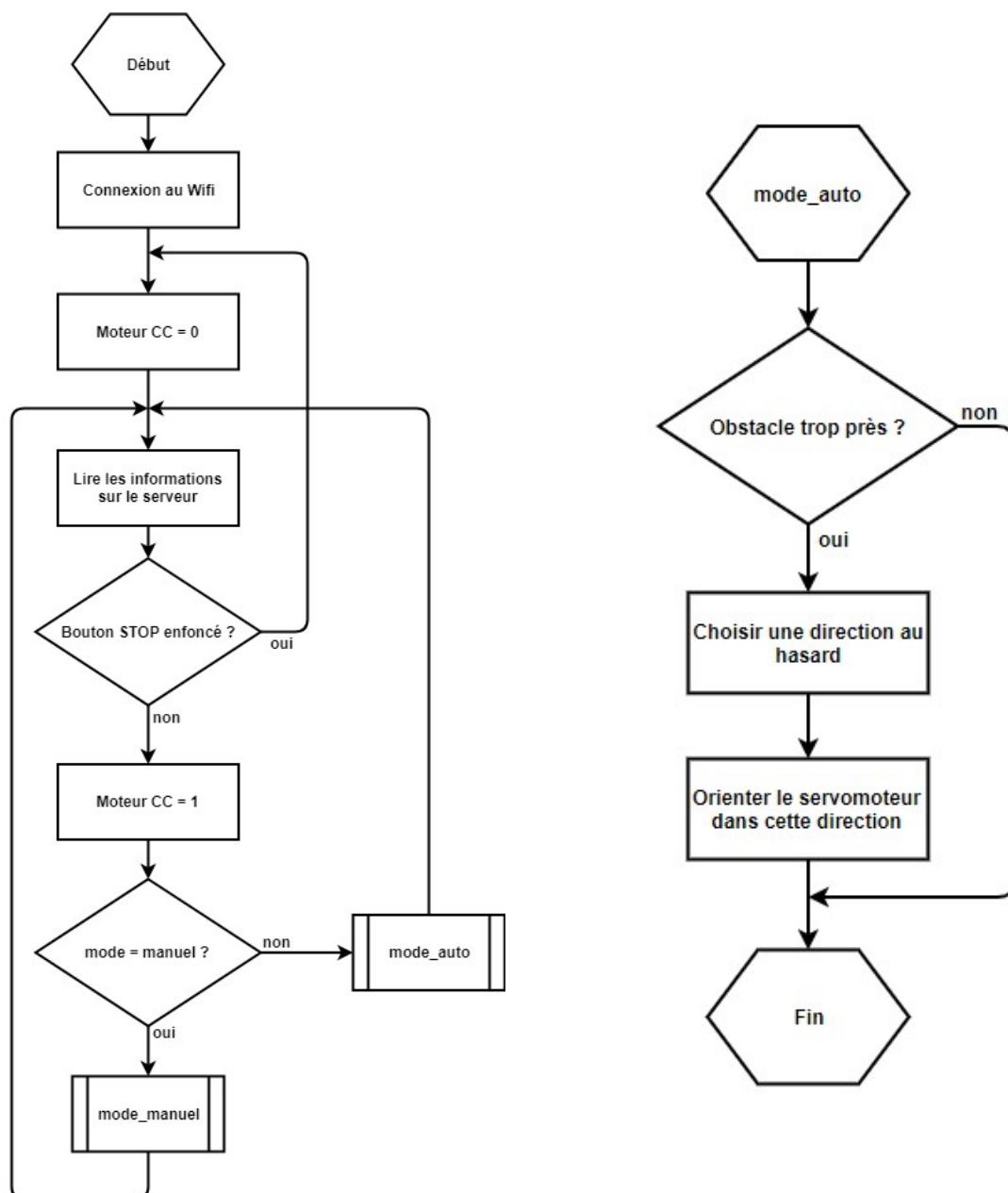
(Cf Annexe 1 – Programme serveur)

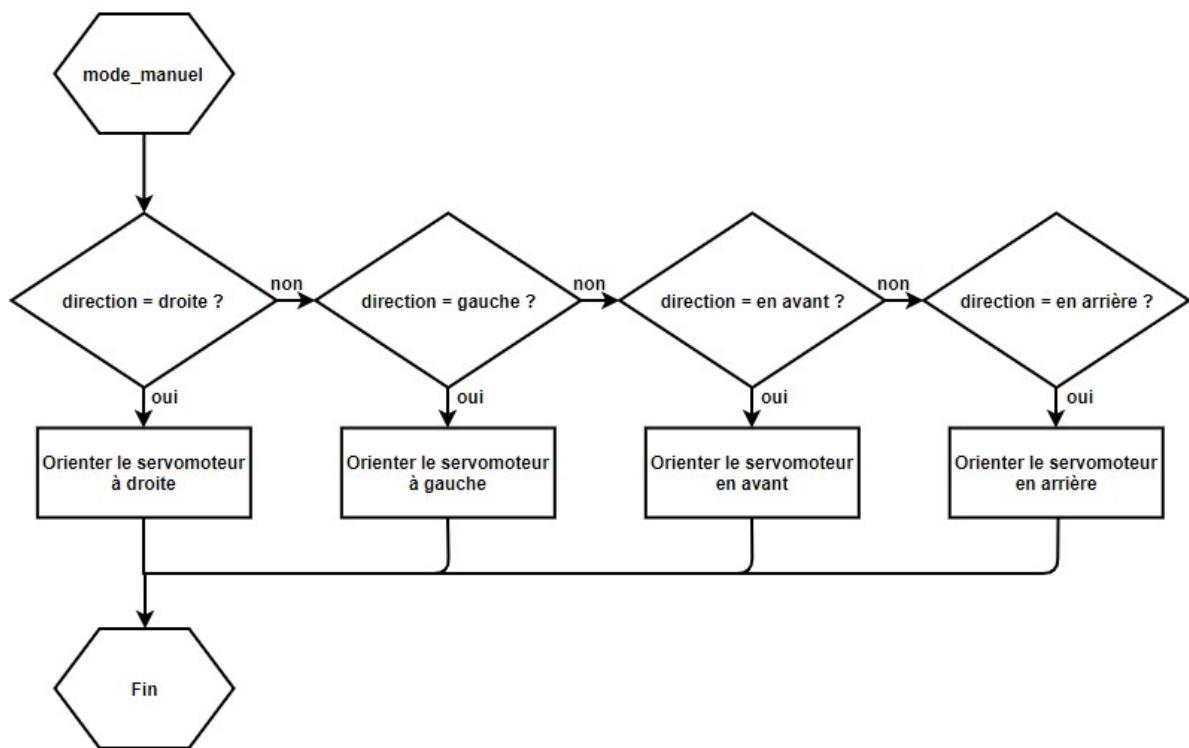
Ce tableau à plus de sens maintenant :

Requêtes envoyées par l'application	Réponses envoyées par le serveur	Interprétation par le robot
appS	direction0	Arrêter le robot
appU	direction1	Diriger le robot en avant
appL	direction2	Diriger le robot à gauche
appR	direction3	Diriger le robot à droite
appD	direction4	Diriger le robot en arrière
ModeAuto	mode5	Mode Automatique
ModeManu	mode6	Mode Manuel
ModeInterrupt	mode7	Mode Interrupteur

### 1.3 -Robot :

Algorigrammes du robot : (Cf Annexe 2 – Programme robot)





### Moniteur série :

Appuie sur le bouton « Manual Mode » et « Left » de l'application

```

Démarrage...
Connection au wifi...
Wifi connecté
.....fin du void setup.......
```

Appuie sur le bouton « Right » de l'application

```

direction2 mode6      ← Mode manuel et direction gauche (réponse du serveur suite à l'ordre de l'application)
Mode manuel (interrupteur à 0)   ← Interprétation + vérification par le robot
servo1left()                ← Changement d'orientation du moteur
```

Appuie sur le bouton « Self-governing Mode » de l'application

```

direction3 mode6      ← Direction droite (réponse du serveur suite à l'ordre de l'application)
Mode manuel (interrupteur à 0)   ← Interprétation + vérification par le robot
servo2right()                ← Changement d'orientation du moteur
```

```

mode5      ← Mode automatique (réponse du serveur suite à l'ordre de l'application)
Mode automatique (interrupteur à 0)
4056      ← Distance en mm dans la direction du robot
servoUp()     ← orientation aléatoire qu'à choisis le robot
```

Défilement automatique      Nouvelle ligne      9600 baud

## **V – Expérimentation**

### **1 – Problèmes rencontrés**

Ce ne sont que les problèmes sur lesquelles j'ai passé plusieurs heures à trouver une solution.

#### **Module Wifi :**

Faire fonctionner ce fichu module ESP8266 fut un réel cauchemar. Sans compter les nombreux problèmes liés à l'alimentation 3.3V, j'ai eu beaucoup de mal à faire fonctionner ce module. Il était impossible de piloter le module avec les commandes AT de façon automatique. Je suis venu à bout de ce problème avec l'aide du professeur.

#### **La lecture des réponses du serveur :**

Le robot devait lire les réponses du serveur. Pour cela, il récupérait le code HTML de la page web du serveur http. Dans un premier temps, il a fallu se débarrasser de tout ce qui ne m'intéressait pas (notamment l'entête). De plus, des caractères ASCII de saut de ligne et de retour chariot sont présents et indispensables dans le code HTML. Pour conter de problème, je n'ai autorisé que les caractères « lisibles » à apparaître dans la réponse lue par le robot.

#### Caractères spéciaux :

```
/*wifi*/  
void AT(String mes)  
{  
    Serial1.print(mes);  
    Serial1.write(13); //caractères invisibles indispensables pour l'envoie des requêtes  
    Serial1.write(10);  
}
```

#### Filtrages de ces caractères :

```
while (car != ':');  
else if (car > 32) // enlève les caractères ascii invisibles  
    texte += car;
```

#### Table ascii :

9	011	09	0001001	HT	Horizontal Tab (tabulation horizontale)	
10	012	0A	0001010	LF	Line Feed (saut de ligne)	
11	013	0B	0001011	VT	Vertical Tab (tabulation verticale)	
12	014	0C	0001100	FF	Form Feed (saut de page)	
13	015	0D	0001101	CR	Carriage Return (retour chariot)	
14	016	0E	0001110	SO	Shift Out (fin d'extension)	

Le 32 ème caractère est le premier lisible ( c'est l'espace )

31	037	1F	0011111	US	Unit Separator ( séparateur d'unité )	
32	040	20	0100000	SP	Espace ( Space en anglais )	
33	041	21	0100001	!	Point d'exclamation	

### Faux contacte du capteur à ultrasons :

Le matériel utilisé était défectueux, un faux contact empêchait le bon fonctionnement du robot. Par moments, le capteur à ultrasons affichait une distance de 0mm car il avait un faux contact ou que le capteur n'était pas précis. J'ai d'abord refais le câblage en entier, mais rien n'a changé.

J'ai donc décidé d'ignorer ces 0 et de traiter que les valeurs comprises entre 1 et la distance minimum séparant l'obstacle du robot.

Tant que l'ai une distance > 1, je fais les manipulations dessus :

```
if (distance_mm > 1)      //empêche les faux contactes de tourner inutilement le moteur
{
    Serial.println(distance_mm);      //distance en mm s'écris dans le moniteur
}
delay(100); //Délai d'attente pour éviter d'afficher trop de résultats à la seconde

if (distance_mm < 200 && distance_mm > 1)  //si le capteur de distance repère un obstacle à moins de 20cm
{
```

## Annexe 1 – Programme serveur

```
Serveur

#include <SPI.h>
#include <Ethernet2.h>
#include <Server.h>
#include <Client.h>
#define Led 13
int direc ;
int mode ;

----- Paramètres du serveur ---
byte mac[] = { 0x90, 0xA2, 0xDA, 0x10, 0xF7, 0x63 }; //adresse MAC du shield Arduino Ethernet
byte ip[] = { 172, 16, 13, 4 }; //172.16.10.4 //adresse du serveur
byte passerelle[] = { 172, 16, 0, 252 };
byte masque[] = { 255, 255, 0, 0 };
EthernetServer serveurMinip(80);

String chaineRecue = ""; // déclare un string vide global
int comptChar = 0; // variable de comptage des caractères reçus

void setup()
{
    Serial.begin(9600);
    Ethernet.begin(mac, ip, passerelle, masque);
    serveurMinip.begin();
}

void loop()
{
    while (!serveurMinip.available());
    EthernetClient client = serveurMinip.available();

    if (client)
    {
        chaineRecue = "";
        comptChar = 0;

        if (client.connected())
        {
            ///////////////////// Reception de la requête envoyée par le client /////////////////////

            while (client.available())
            {
                char c = client.read();
                comptChar++;
                if (comptChar > 130) chaineRecue = chaineRecue + c;
            }

            if (chaineRecue.indexOf("appU") != -1) { //ici, quand je recois appU, "direc" prends la valeur 1
                Serial.println("requête appU recue");
                direc = 1; //Up
            }
            if (chaineRecue.indexOf("appL") != -1) { //Quand je recois appL, "direc" prends la valeur 2
                Serial.println("requête appL recue");
                direc = 2; //Left
            }
            if (chaineRecue.indexOf("appR") != -1) { //Quand je recois appR, "direc" prends la valeur 3
                Serial.println("requête appR recue");
                direc = 3; //Right
            }
            if (chaineRecue.indexOf("appD") != -1) { //Quand je recois appD, "direc" prends la valeur 4
                Serial.println("requête appD recue");
                direc = 4; //Down
            }
            if (chaineRecue.indexOf("appS") != -1) { //Quand je recois appS, "direc" prends la valeur 0
                Serial.println("requête appS recue");
                direc = 0; //Stop
            }
        }
    }
}
```

```

if (chaineRecue.indexOf("ModeManu") != -1) {      //ici, quand je recois appU, "mode" prends la valeur 6
    Serial.println("requête ModeManu recue");
    mode = 5;
}
if (chaineRecue.indexOf("ModeAuto") != -1) {      //Quand je recois appS, "mode" prends la valeur 5
    Serial.println("requête ModeAuto recue");
    mode = 6;
}
if (chaineRecue.indexOf("ModeInterup") != -1) { //Quand je recois appS, "mode" prends la valeur 7
    Serial.println("requête ModeInterup recue");
    mode = 7;
}

// envoi d'un entete standard de réponse http
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("Access-Control-Allow-Origin: *");
client.println("Connection: close");
client.println();

//entete de la page html avec rafraîchissement automatique toutes les 0.5s
client.print("<head><meta http-equiv=\"refresh\" content=\"0.5;url=http://172.16.13.4\" /></head>");
//la div permet au collègue qui s'occupe du site de lire le contenu plus facilement
client.print("<div id=\"kardech\">");

client.print("direction");
client.print(direc);      //= direction0; direction1; direction2; direction3; direction4;

client.print("</div>");   //fin div id=kardech

client.print("mode");
client.print(mode);

client.print("Fin");     //ce mot me permet d'encadrer la valeur du mode et de la lire plus facilement

delay(1);
client.stop();
}
}

```

*Le programme est disponible sur [github.com/Rabire/MEDUSA/blob/master/Serveur\\_Rabire.ino](https://github.com/Rabire/MEDUSA/blob/master/Serveur_Rabire.ino)*

## Annexe 2 – Programme robot

```
Robot_Rabire

#define interrupteur 7
#define moteur 10
#include <Servo.h>      //library servo

/*servo*/
Servo monServo1;
Servo monServo2;
int po = 0;
String message = "";           //message= chaîne de caractère de base qui est envoyé
String texte;                 //texte= chaîne de caractère reçue par le serveur

/*ultrasons*/
const byte TRIGGER_PIN = 2;    // Broche TRIGGER
const byte ECHO_PIN = 3;        // Broche ECHO
const unsigned long MEASURE_TIMEOUT = 25000UL;   // 25ms = ~8m à 340m/s Constantes pour le timeout
const float SOUND_SPEED = 340.0 / 1000;          //Vitesse du son dans l'air en mm/us

long randNumber;  //direction au hasard

/*SERVOMOTEURS*/
void servolleft() //fonction qui dirige le servol à gauche
{
    for (po = 1; po < 90; po++)
    {
        monServo1.write(po);
    }
}
void servoldown() //fonction qui dirige le servol en arrière
{
    for (po = 90; po < 180; po++)
    {
        monServo1.write(po);
    }
}
void servo2right() //fonction qui dirige le servo2 à droite
{
    for (po = 1; po < 90; po++)
    {
        monServo2.write(po);
    }
}
void servo2up() //fonction qui dirige le servo2 en haut
{
    for (po = 90; po < 180; po++)
    {
        monServo2.write(po);
    }
}

/*Wifi*/
void AT(String mes)
{
    Serial1.print(mes);
    Serial1.write(13); //caractères invisibles indispensables pour l'envoie des requêtes
    Serial1.write(10);
}

void envMessage() //fonction qui envoie le message
{
    int nbCar = message.length();
    AT("AT+CIPSEND=" + String(nbCar));
    AttendConnect("OK");
    Serial1.print(message);
    //Serial1.write(13);
    //Serial1.write(10);
    Serial1.println();
}
```

```

void recMessage() //fonction qui recoit les réponses du serveur
{
    texte = ""; //réinitialisation du message envoyé au cas ou il prends une valeur avant (GPS)
    do
        if (Serial1.available() > 0)
        {
            char car = Serial1.read();
            if (car == '+')
                do
                {
                    if (Serial1.available() > 0) car = Serial1.read();
                }
                while (car != ':');
            else if (car > 32) // enleve les caractes ascii invisibles
                texte += car;
        }
    while (texte.indexOf("CLOSE") == -1);
    while (Serial1.available() > 0) Serial1.read();
    Serial.print(texte.substring(texte.indexOf("direction"), (texte.indexOf("</div>"))));
    Serial.print(" "); //je ne garde que la partie de réponse du serveur qui me concerne (direction)
    Serial.print(texte.substring(texte.indexOf("mode"), (texte.indexOf("Fin"))));
    Serial.print(" "); //je ne garde que la partie de réponse du serveur qui me concerne (mode)
}

void AttendConnect(String mot)
{
    texte = "";
    do
        if (Serial1.available() > 0)
        {
            char car = Serial1.read();
            texte += car;
            //Serial.print(car);
        }
    while (texte.indexOf(mot) == -1);
    while (Serial1.available() > 0) Serial1.read();
}
void setup()
{
    Serial.begin(9600);
    Serial1.begin(9600);

    pinMode(interrupteur, INPUT_PULLUP); //interrupteur défini
    pinMode(moteur, OUTPUT); //moteur (direction) définie

    Serial.println("Demarrage...");
    Serial.println("Connection au wifi...");
    AT("AT+CWJAP=\"Wifiprojet\", \"projet123\""); //connexion au Wifi
    AttendConnect("OK");
    Serial.println("Wifi connecté");

    /*ultason*/
    pinMode(TRIGGER_PIN, OUTPUT); //Initialise les broches capt ultason
    digitalWrite(TRIGGER_PIN, LOW); // La broche TRIGGER doit être à LOW au repos
    pinMode(ECHO_PIN, INPUT); //Initialise les broches capt ultason

    /*servomoteurs*/
    monServo1.attach(A4);
    monServo2.attach(A2);
    monServo1.write(po);
    monServo2.write(po);

    randomSeed(analogRead(0)); //direction random

    Serial.println(".....fin du void setup.....");
}

```

```

void loop()
{
    AT("AT+CIPSTART=\"TCP\", \"172.16.13.4\", 80"); //connexion au serveur
    AttendConnect("OK");
    envMessage(); // Envoie de la requete au serveur
    recMessage(); // reception de la reponse du serveur
    AT("");
    delay(300);

    /* MODE MANUEL */
    if (texte.substring(texte.indexOf("mode"), (texte.indexOf("Fin")))) == "mode6") //si le mode est manuel
        if (digitalRead(interrupteur) == 0) //si l'interupepteur l'autorise
    {
        Serial.println("Mode manuel (interrupteur à 0)");
        digitalWrite(moteur, HIGH); //faire tourner le moteur
        if (texte.substring(texte.indexOf("direction"), (texte.indexOf("</div>")))) == "direction2")
        { //si le serveur dit d'aller à gauche
            servolleft(); //orienter le moteur à gauche
            Serial.println("servolleft()");
        }
        if (texte.substring(texte.indexOf("direction"), (texte.indexOf("</div>")))) == "direction4")
        { //si le serveur dit d'aller en arrière
            servoldown(); //orienter le moteur en arrière
            Serial.println("servoldown()");
        }
        if (texte.substring(texte.indexOf("direction"), (texte.indexOf("</div>")))) == "direction1")
        { //si le serveur dit d'aller en avant
            servo2up(); //orienter le moteur en avant
            Serial.println("servo2up()");
        }
        if (texte.substring(texte.indexOf("direction"), (texte.indexOf("</div>")))) == "direction3")
        { //si le serveur dit d'aller à droite
            servo2right(); //orienter le moteur à droite
            Serial.println("servo2right()");
        }
        if (texte.substring(texte.indexOf("direction"), (texte.indexOf("</div>")))) == "direction0")
        { //si le serveur dit d'arreter le moteur
            digitalWrite(moteur, LOW); //arreter le moteur
        }
    }
    digitalWrite(TRIGGER_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIGGER_PIN, LOW);
    long measure = pulseIn(ECHO_PIN, HIGH, MEASURE_TIMEOUT); //Mesure le temps entre l'envoi de l'impulsion u
    float distance_mm = measure / 2.0 * SOUND_SPEED; //Calcul la distance à partir du temps mesuré
}

```

```

/*MODE AUTOMATIQUE*/
if (texte.substring(texte.indexOf("mode"), (texte.indexOf("Fin")))) == "mode5") //si le mode est automatique
  if (digitalRead(interupteur) == 1) //si l'interupteur l'autorise
  {
    Serial.println("Mode automatique (interupeur à 1)");
    digitalWrite(moteur, HIGH);
    if (distance_mm > 1) //empêche les faux contactes de tourner inutilement le moteur
    {
      Serial.println(distance_mm); //distance en mm s'écris dans le moniteur
    }
    delay(100); //Délai d'attente pour éviter d'afficher trop de résultats à la seconde

    if (distance_mm < 200 && distance_mm > 1) //si le capteur de distance repère un obstacle à moins de 20cm
    {
      randNumber = random(1, 4); //generer un numéro aléatoire entre let 4 compris
      if (randNumber == 1) // si le numéro aléatoire = 1
      {
        servo2up(); //orienter le moteur en avant
        Serial.println("servo2up()");
      }
      if (randNumber == 2) // si le numéro aléatoire = 2
      {
        servolleft(); //orienter le moteur à gauche
        Serial.println("servolleft()");
      }
      if (randNumber == 3) // si le numéro aléatoire = 3
      {
        servo2right(); //orienter le moteur à droite
        Serial.println("servo2right()");
      }
      if (randNumber == 4) // si le numéro aléatoire = 4
      {
        servoldown(); //orienter le moteur en bas
        Serial.println("servoldown()");
      }
    }
  }
}

```

*Le programme est disponible sur [github.com/Rabire/MEDUSA/blob/master/Robot\\_Rabire.ino](https://github.com/Rabire/MEDUSA/blob/master/Robot_Rabire.ino)*