POLYTECH

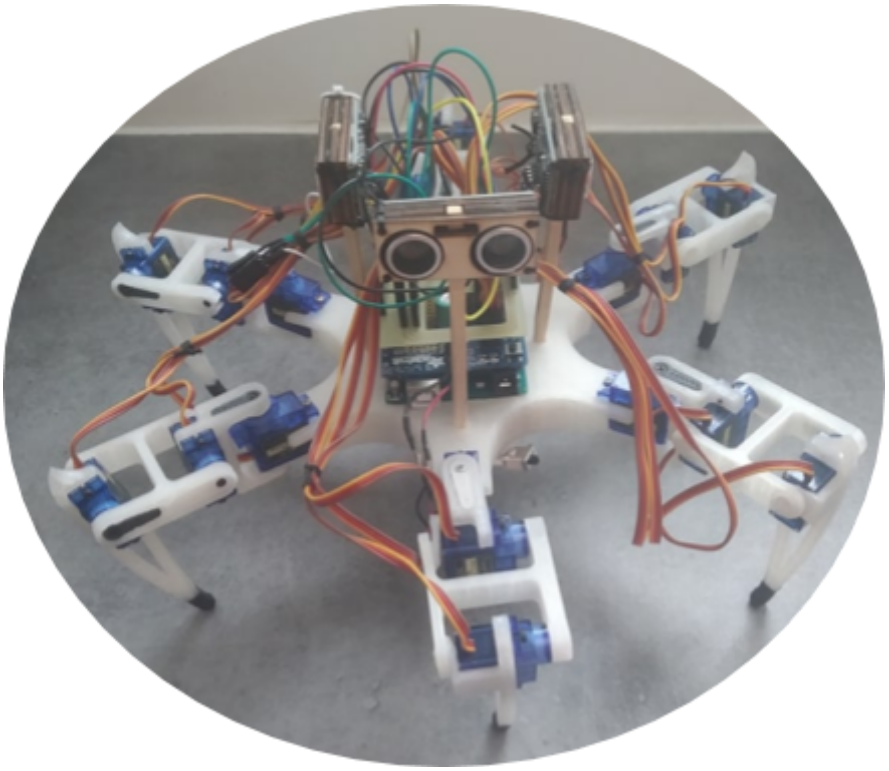
Université de Lille

Projet IMA4

Robot hexapode de mesure de RSSI WiFi

Rémi Foucault - Hugo Velly

2018-2019



Remerciements

C’est avec plaisir que nous tenions à remercier :

- Nos tuteurs Xavier Redon, Alexandre Boé et Thomas Vantroys qui nous ont conseillé quant au matériel à utiliser, aux points importants du projet et ont toujours été disponibles pour répondre à nos questions.

- Thierry Flamen qui s’est rendu totalement disponible, notamment pendant la première semaine des vacances d’avril. Par son expérience théorique et pratique, il nous a aidé, conseillé et guidé lors de la conception et du prototypage de notre carte WiFi.

- Les Fab’ Manager qui ont pris le temps de nous former, de nous aider, de nous rediriger vers les bons outils lorsque cela était nécessaire.

- Théau Moinat qui a toujours pris de son temps lorsque nous avions des questions concernant l'électronique et qui a grandement participé à la conception de l’antenne du pcb.

Sommaire

Introduction

Aujourd'hui la technologie WIFI est une des solutions les plus utilisées pour accéder au réseau internet, mais lors de l'installation de celle-ci dans de grands bâtiments, la couverture réseau n'est souvent pas optimale.

Réaliser une carte d'un bâtiment sur laquelle est affichée l'intensité du réseau WIFI (RSSI) permet très facilement de repérer les points faibles et les points forts du réseau pour pouvoir ensuite l'optimiser. Bien que des logiciels de mesure RSSI existent déjà, ils demandent à l'utilisateur de se déplacer dans tout le bâtiment pour prendre lui même les mesures. Une solution pour faciliter cette démarche serait d'associer la mesure de RSSI à un robot autonome.

Dans le cadre de notre projet de quatrième année à Polytech Lille nous avons donc choisi de réaliser un robot hexapode qui mesure le flux RSSI. Dans ce rapport nous allons explorer les différentes étapes du processus de réalisation de ce projet depuis le choix du sujet jusqu'à la réalisation du prototype.

I - Présentation du projet

1. Analyse du sujet

L'objectif de notre projet est de pouvoir cartographier l'intensité du signal WiFi dans un bâtiment (Polytech Lille par exemple) à l'aide d'un robot hexapode.

Un des premiers aspects du projet est l'autonomie, le but étant de limiter la nécessité d'un être humain lors de la prise de mesures. Le robot doit donc transporter avec lui une source d'alimentation.

Le robot doit être autonome dans ses déplacements lorsqu'il parcours le bâtiment, cela inclut la rencontre d'obstacles et d'escaliers. Le choix d'un hexapode, comme proposé dans le sujet, nous paraît approprié ; ses six pattes équipées de servomoteurs lui permettent de se déplacer dans toutes les directions à l'horizontal mais aussi de gravir des marches d'escalier.

Pour pouvoir se déplacer correctement le robot doit aussi avoir conscience de son environnement, c'est pourquoi il est équipé de capteurs. Des capteurs ultrasons nous semble être la bonne solution, leur utilisation est simple et leur précision suffisante pour que le robot puisse se diriger dans son environnement.

Lors de son déplacement, le robot doit mesurer la puissance du signal Wifi et faire correspondre cette mesure à sa position dans le bâtiment. Il faut donc lire le RSSI (des modules électroniques existent pour ça) et savoir où se trouve le robot dans le bâtiment. Pour que le robot se localise nous pensons à des marqueurs placés stratégiquement dans le bâtiment que le robot lira pour avoir une idée de sa position. Pour la lecture de marqueurs le plus simple nous semble l'utilisation de QR codes qui seraient reconnus par une Raspberry Pi équipée d'une caméra.

Pour contrôler les servomoteurs et effectuer la lecture du RSSI, l'utilisation d'une simple carte Arduino Uno nous semble appropriée.

2. Objectifs

Nous sommes conscients que la description du projet faites ci dessus est assez ambitieuse, c'est pourquoi nous nous sommes fixé des sous-objectifs. Le but étant de parcourir au maximum la liste ci-dessous :

- Conception du robot

- Modéliser le robot en s'inspirant de modèles existants et disponibles en ligne
- Réaliser le robot en impression 3D au Fabricarium de Polytech Lille
- Monter le robot avec ses 18 servomoteurs

- Déplacement et détection d'obstacles

- Implémenter l’arduino sur le robot
- Implémenter un shield PWM pour contrôler les moteurs
- Implémenter les capteurs à ultrason
- Coder le déplacement et la prise en compte d'obstacles

- Mesure RSSI

- Concevoir, sous la forme d’un shield, un pcb intégrant un ESP8266-04
- Coder la mesure et le traitement du RSSI

- Positionnement

- Mettre en place un système de marqueur de type QR code
- Implémenter une Raspberry Pi avec une caméra
- Coder la lecture du marqueur
- Coupler la mesure du RSSI avec la position
- Adapter la trajectoire du robot en fonction de sa position

- Cartographie

- Coder la réalisation d’une carte de type “heat map” à partir des données RSSI/position

3. Matériel

Le matériel suivant est utilisé pour ce projet :

- 1 Arduino Uno
- 18 servomoteurs Tower Pro SG90
- 4 capteur de distance ultrason
- 1 shield PWM
- 1 Module WiFi Arduino ESP8266
- 4 Piles rechargeables 1,2V pour l'alimentation des servomoteurs
- 1 batterie externe pour l'alimentation de l'arduino

II - Conception mécanique

Pour la réalisation du robot nous parcourons le web à la recherche d'hexapodes déjà existants, le but étant de comprendre comment ils fonctionnent mécaniquement et de s'inspirer de modèles existants.

Un robot hexapode mime les articulations des insectes, il est constitué du châssis (le corps) et de 6 pattes. Chaque patte est décomposée en 3 parties : la coxa (la hanche), le fémur et le tibia.

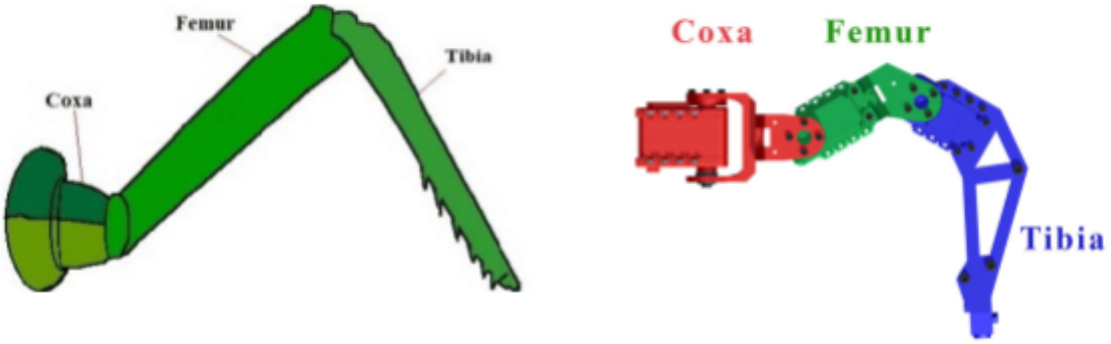


Figure 1 : Comparaison insecte / robot

Il existe 2 type de morphologie pour les robots hexapodes, ceux avec 3 pattes de chaque côté et ceux avec les 6 pattes réparties de façon hexagonale. C'est cette seconde morphologie que nous voulons utiliser, car elle permet au robot de se déplacer à l'horizontale dans toutes les directions sans avoir à tourner sur lui-même.

1. Modélisation 3D

La première étape de la conception du robot passe par la modélisation sur ordinateur. Après nos recherches nous avons décidé de modéliser nous même le châssis et les pièces utiles aux capteurs. Pour les pattes, nous allons utiliser, après une légère modification, celles du robot quadrupède *Spider Robot*, posté par l'utilisateur *Regishu* sur le site de partage 3D *Thingiverse* et libre de droit.

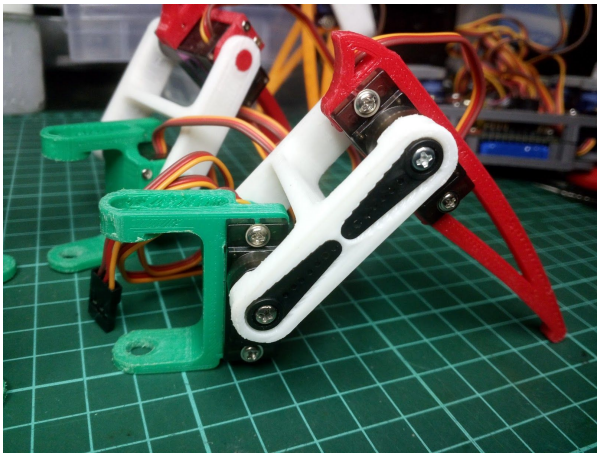


Figure 2 : Inspiration pour le modèle de patte

Pour modéliser le châssis nous utilisons le logiciel Autodesk Fusion 360, le logiciel à l'avantage d'être gratuit et plutôt facile d'utilisation pour les personnes qui, comme nous, n'ont jamais fait de modélisation. Notre châssis doit correspondre à la morphologie souhaitée et utiliser le moins de PLA possible pour minimiser le poids tout en s'assurant que la structure reste assez robuste et permettre l'implémentation de tous les composants électroniques sur le robot. Nous nous arrêtons sur ce modèle final.

Figure 3 : Modélisation 3D du châssis

Le châssis est creux pour pouvoir y fixer un boitier 4 piles par le dessous, une empreinte est également présente pour visser une arduino uno sur le dessus.

Pour modéliser les supports des capteurs à ultrasons nous utilisons le logiciel Onshape, l'un de nous ayant, entre temps, reçu une formation sur ce logiciel pendant ses cours. Nous voulons placer les capteurs à une hauteur d'une dizaine de centimètres du châssis pour éviter que les pattes ne passent devant lorsqu'elles effectuent leurs mouvements. Avec toutes ces contraintes, nous avons conçu le design suivant.

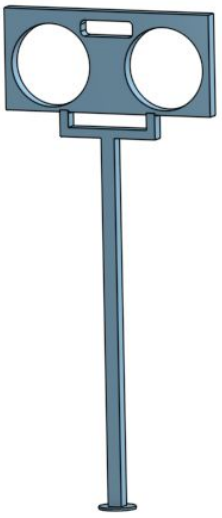


Figure 4 : Modélisation 3D du support pour les capteurs ultrason

2. Réalisation

Pour pouvoir utiliser les imprimantes du Fabricarium de Polytech en toute autonomie nous avons suivis la formation Impression 3D proposé par le Fabricarium.

Après un prototypage des pattes pour obtenir le jeu mécanique souhaité, nous imprimons le châssis et assemblons le tout avec les servomoteurs.

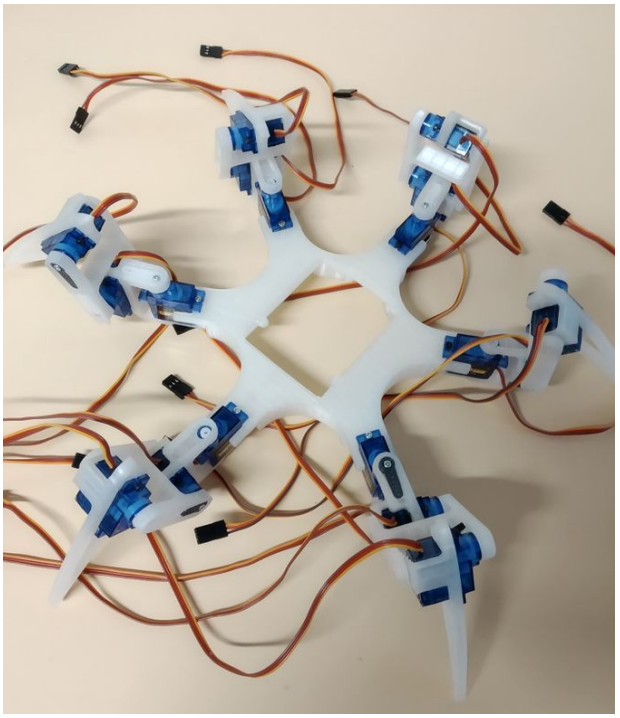


Figure 5 : Structure du robot

Les supports pour capteurs à ultrason devaient eux aussi être imprimés mais suite aux conseils des Fab' Managers nous avons modifié le design pour les réaliser à la découpeuse laser. En effet lorsque le projet le permet utiliser la découpeuse laser est beaucoup plus rapide (2 min de découpe contre normalement 20 min d'impression) et permet donc un gain de temps considérable, notamment en prototypage. Une fois les supports découpés dans du bois puis assemblés, les capteurs y sont encastrés et l'ensemble est maintenant par du fil.

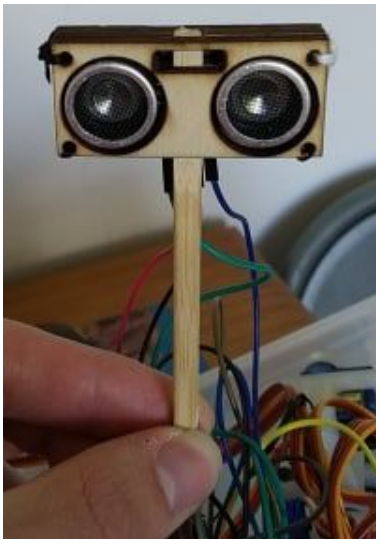


Figure 6 :Support de capteur ultrason

Finalement, des trous seront percés dans le châssis pour y fixer les tiges en bois des capteurs. Nous avons décidé d'utiliser 3 capteurs pour permettre à notre robot d'être conscient au maximum de son environnement et de ne pas entrer en collision avec les obstacles qui l'entourent si bien de face que sur les côtés. Comme le robot ne devrait pas faire de trajet en marche arrière il ne nous semble pas utile d'avoir un capteur à l'arrière. Nous envisageons d'ajouter un second capteur à l'avant, qui serait incliné différemment par rapport à l'horizontale afin de différencier les murs des escaliers.

La partie mécanique du robot est maintenant complète. Il faut à présent y implémenter l'électronique.

III - Conception Électronique

Pour ce qui concerne la conception électronique du robot, elle s'oriente autour de la carte Arduino Uno. En effet pour le contrôle des servomoteurs nous utiliserons un shield PWM qui se positionne sur l'Arduino. Ensuite nous réaliserons un second shield qui viendra se placer au dessus du premier et qui permettra principalement à l'Arduino de communiquer avec l'ESP8266.

Pour alimenter notre Arduino, nous utilisons une batterie externe portable qui fournit du 5V nécessaire à l'alimentation de la carte.

1. Mise en place du shield PWM

Le shield PWM que nous avons commandé est construit par Adafruit. Il permet de contrôler jusqu'à 16 ports PWM grâce à l'utilisation du bus I2C. Notre robot comprenant 18 servomoteurs, nous utiliserons les ports PWM classiques de l'Arduino Uno pour contrôler les 2 servomoteurs restants.

Le shield est fourni presque prêt à l'utilisation. Il nous a suffit de souder les différents headers fournis. Il y avait également sur le shield un emplacement prévu pour une capacité de découplage. Au vu de notre utilisation du shield (16 servomoteurs), nous avons décidé de souder une capacité de 1000 uF.

Concernant l'alimentation du shield, les composants sont alimentés directement depuis l'Arduino. Pour l'alimentation des servomoteurs il est par contre nécessaire d'utiliser une source extérieure. Nous avons donc décidé d'utiliser un bloc de 4 piles rechargeables LR6 1,2V qui fournissent bien les 4,8V nécessaires au fonctionnement des servomoteurs. Nous avons ensuite fixé le bloc de piles sous le robot et nous avons soudé un interrupteur nous permettant facilement de contrôler l'alimentation des servomoteurs, ce qui s'avère très pratique lors des essais de déplacement du robot.

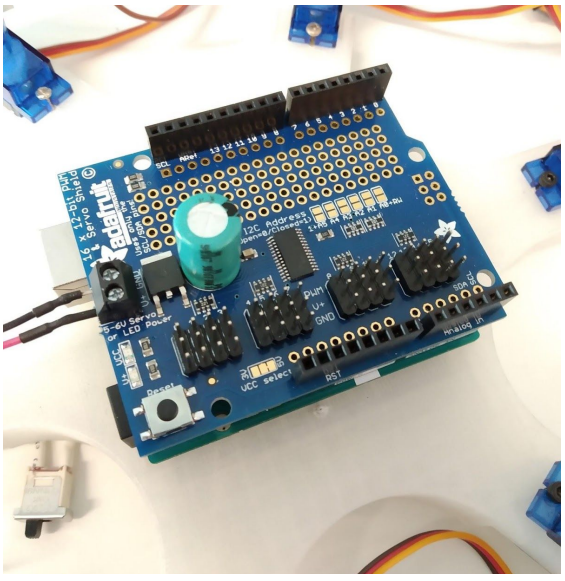


Figure 7 : Shield PWM

2. Création du second shield

Ce second shield doit gérer la communication avec l'ESP8266 et fournir deux rangées de headers : une alimentation 5V et une masse communes pour les capteurs ultrason ainsi que les deux servomoteurs qui ne sont pas contrôlés par le shield PWM.

La partie délicate de la conception de ce PCB est l'ESP8266. En effet il s'agit d'un microcontrôleur permettant de travailler avec le signal WiFi, vendu sous différentes versions, avec la présence ou non d'une antenne par exemple. La version que nous avons commandée est l'ESP-04, qui ne possède pas d'antenne. Nous devons donc ajouter sur notre PCB une antenne pour que l'ESP puisse capter le réseau WiFi. Il existe différents types d'antennes mais nous avons choisi d'en réaliser une nous même en suivant la datasheet d'un composant possédant une antenne WiFi 2,4 GHz tracée en cuivre sur le composant.

Pour que l'antenne ne soit pas gênée par d'éventuels rayonnements de l'Arduino ou du shield PWM, il est préférable de positionner notre second shield au dessus du premier. Cependant les servomoteurs étant connectés au shield PWM, notre second shield doit donc posséder un trou central permettant de laisser passer les câbles des moteurs ainsi que la capacité du shield PWM qui est assez haute.

Nous avons donc commencé la création de ce PCB sur le logiciel Fritzing qui nous semblait adapté à la conception d'une carte plutôt simple comme celle ci. Cependant lorsque nous avons voulu commencer à concevoir l'antenne en trace sur le PCB, nous nous sommes rendu compte que Fritzing n'était pas le logiciel le plus adapté et nous avons réalisé que Altium Designer serait bien plus pratique pour créer l'antenne. Nous avons donc recommencer la carte sur Altium. Nous avons également pu aisément créer la forme spéciale du PBC alors que cela avait plutôt difficile sur Fritzing.

Pour l'antenne nous avons créé un composant reprenant les mesures exactes fournies dans la datasheet.

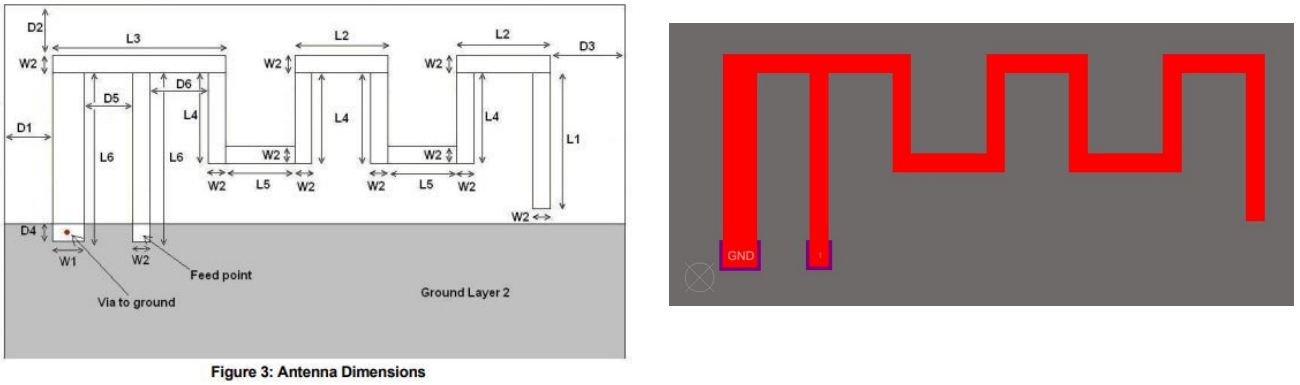


Figure 3: Antenna Dimensions

Figure 8 : Exemple d'antenne et réalisation sur Altium

On peut alors passer au routage et relier l'antenne à l'ESP8266. Nous avons trouvé sur internet une bibliothèque contenant les différents types d'ESP, ce qui nous a évité d'avoir à recréer ce composant.

L’alimentation de l’ESP se fait avec la broche 3,3V de l’Arduino, le pin CH_PD qui permet d’activer l’ESP est relié à une sortie digitale de l’Arduino. Enfin la communication entre l’ESP et l’Arduino se fait par liaison série. Pour cela on relie donc les Rx et Tx de l’ESP au ports séries UART de l’Arduino Uno.

Pour obtenir un fonctionnement optimal de l’antenne, il est préférable d’avoir une ligne de masse la plus grosse possible et d’avoir des angles arrondis pour les lignes reliées à l’antenne. Il faut également que la ligne reliant l’antenne à l’ESP transmette une impédance de 50Ω.

Les Schematic et PCB de notre carte sont visibles en annexe page 24.

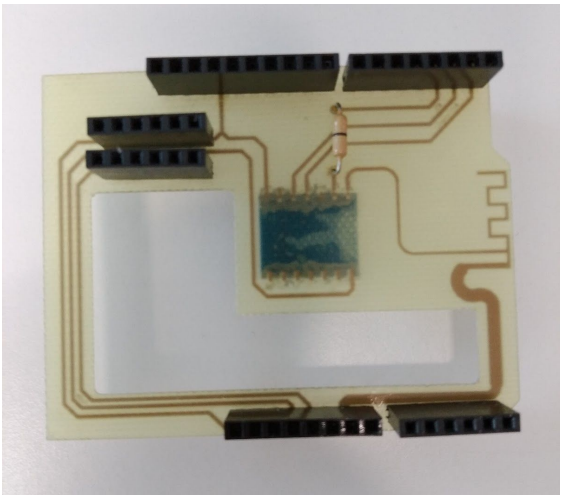


Figure 9 : Notre carte électronique

Après avoir fait graver cette carte, nous l’avons essayé cependant nous n’avons pas réussi à communiquer avec l’ESP. Nous avons donc entrepris des recherches approfondies sur l’ESP et le problème peut venir du fait que nous utilisons la liaison UART de l’Arduino. Nous avons alors modifié le PCB en rayant les pistes et en soudant de petits fils pour créer de nouvelles liaisons. Nous avons relié l’ESP8266 à deux entrées/sorties classiques de l’Arduino que nous pouvons utiliser comme liaison série grâce à la bibliothèque SoftwareSerial. Nous avons également lu que pour certains modules, le niveau de tension envoyé par l’Arduino peut être trop important pour l’ESP. Nous avons donc également ajouté un petit montage pour baisser la tension allant du Tx de l’Arduino vers le Rx de l’ESP.

Avec ces modifications nous n’avons toujours pas réussi à communiquer avec l’ESP et nous ne sommes pas capable de comprendre d’où vient le problème.

Après avoir passé un certain temps à essayer de faire fonctionner cette carte, nous arrivions proche de l’échéance du projet sans avoir de lecture du réseau WiFi. Nous avons donc obtenu un autre type d’ESP8266, le modèle 01 qui possède une antenne intégrée et est prêt à l’utilisation avec l’Arduino. Il nécessite juste d’être branché comme le montre le montage

suivant. Avec cet autre modèle, nous avons pu communiquer avec l'ESP et travailler avec le réseau WiFi.

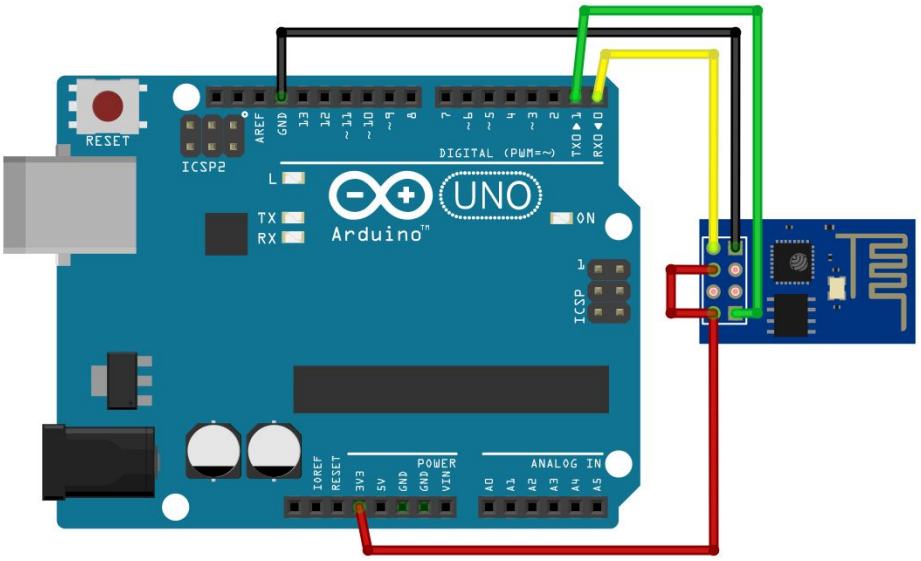


Figure 10 : Montage de l'ESP8266-01 sur l'Arduino

IV - Fonctionnement du robot

Le robot étant conçu, il faut maintenant le programmer. Pour le code nous utilisons les bibliothèques classiques Arduino et une bibliothèque Adafruit pour le shield PWM écrite en C++. Nous utilisons donc pour la programmation l'IDE Arduino qui est pratique pour la gestion des bibliothèques.

Le code du robot doit lui permettre de se déplacer, de prendre en compte les obstacles et de mesurer le RSSI.

1. Déplacement

    a. Contrôler et reconnaître chaque servomoteur

Avant de détailler le code permettant au robot de se mouvoir, nous devons expliquer comment nous contrôlons les servomoteurs.

Deux des 18 servomoteurs utilisés sont connectés à des entrées/sorties PWM de l'arduino Uno et sont contrôlés à l'aide de la fonction *write(int position)* de la bibliothèque arduino *servo.h*. Cette fonction met le moteur à la position en degrés indiquée en paramètre.

Les 16 autres moteurs sont connectés au Shield PWM et sont donc contrôlés à l'aide de la fonction *setPWM(int pin, int on, int off)* de la bibliothèque *Adafruit_PWMServoDriver*. Cette fonction allume le moteur spécifié par le *pin*, de la pulsation *on* à la pulsation *off.* A l'aide de cette fonction, le moteur tourne jusqu'à une position donnée. Il suffit alors d'établir de façon empirique une correspondance entre la position du moteur et la valeur *off* associée, *on* étant toujours à 0 pour notre utilisation.

On défini donc les deux pulsations qui correspondent aux positions extrêmes de chaque type de servomoteur (tibia, fémur et coxa) par rapport à la structure de notre robot. On enregistre ces données dans une structure *legPart* contenant les deux variables *legMin* et *legMax*. La structure comprend également une position *legCalibrate* qui correspond à la position de base du robot se tenant debout. Après nos mesures on obtient donc 3 structures *legPart* différentes correspondant aux trois types de servomoteurs : *tibia*, *femur* et *coxa*. Ces 3 structures sont universelles pour les 6 pattes, celles ci étant conçues exactement de la même manière.

Les 16 servomoteurs du shield étant commandés par une valeur en pulsation alors que les deux derniers branchés directement sur l'Arduino étant commandés en degrés, nous avons mis en place deux fonctions qui nous permettent à tout moment de passer de pulsation en degré et vice versa à l'aide de la fonction map.

```
// Permet de convertir les degrés en pulselen
int degToPuls(int degrees, struct legPart leg) {
  return(map(degrees, 0, 180, leg.legMin, leg.legMax));
}
```

```
// Permet de convertir les pulselen en degrés
int pulsToDeg(int pulselen, struct legPart leg) {
  return(map(pulselen, leg.legMin, leg.legMax, 0, 180));
}
```

Maintenant que l'on sait comment contrôler chaque servomoteur, il faut trouver un moyen de sélectionner facilement un moteur bien spécifique parmi tous ceux du robot. Nous utilisons pour cela un tableau à 2 dimensions dans lequel sont stockés les numéros de pin du shield PWM. Par exemple pour atteindre le moteur contrôlant le tibia de la patte numéro 2 (pin 6 sur le shield) il faut taper *patte[2][0]* qui retournera 6. La valeur 100 est associée aux deux moteurs non reliés au shield (*patte[5][1]* et *patte[5][2]*) pour les différencier. Vous trouverez un schéma explicatif sur la numérotation des pattes en annexe page 25.

b. Contrôler une patte et définir un mouvement

Maintenant que nous savons sélectionner un moteur et choisir sa position nous pouvons contrôler toute une patte. La fonction *mouvPatte()* prend en paramètre le numéro de la patte souhaitée et la commande en pulsation des 3 moteurs associées :

```
//commander une patte
void mouvPatte(int patteNum, int tibiaPuls, int femurPuls, int coxaPuls){

    pwm.setPWM(patte[patteNum][0],0, tibiaPuls);

    if(patte[patteNum][1]==100) //femur 5 non relié au shield
      {servo16.write(pulsToDeg(femurPuls,femur));}
    else
      {pwm.setPWM(patte[patteNum][1],0,femurPuls);}

    if(patte[patteNum][2]==100) //coxa 5 non relié au shield
      {servo17.write(pulsToDeg(coxaPuls,coxa));}
    else
      {pwm.setPWM(patte[patteNum][2],0, coxaPuls);}
}
```

Si l'on essaye de faire bouger la patte comportant les deux servomoteurs non reliés au shield PWM (valeur de 100 dans le tableau *patte*), alors la fonction convertit la commande en degrés et utilise bien la fonction *write()* de la bibliothèque *servo.h*.

Avec cette fonction permettant de faire bouger n'importe quelle patte dans une position voulue, on peut maintenant coder le déplacement du robot qui consiste simplement en une succession de mouvements des pattes. Cette première fonction nous permet de faire avancer le robot d'un pas.

```
void faireUnPas(int mode, int vitesse)
{
  mouvPatte((2+mode)%6,TIBIA_CALIBRATE, 350, 230);
  delay(vitesse);
```

```
  mouvPatte((2+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, 230);
  delay(vitesse);
  mouvPatte((4+mode)%6,TIBIA_CALIBRATE, 350, 410);
  delay(vitesse);
  mouvPatte((4+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, 410);
  delay(vitesse);
  mouvPatte((0+mode)%6,TIBIA_CALIBRATE, 350, COXA_CALIBRATE);
  mouvPatte((3+mode)%6,TIBIA_CALIBRATE, 350, COXA_CALIBRATE);
  delay(vitesse);
  mouvPatte((2+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, COXA_CALIBRATE);
  mouvPatte((4+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, COXA_CALIBRATE);
  mouvPatte((1+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, 410);
  mouvPatte((5+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, 230);
  delay(vitesse);
  mouvPatte((0+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, COXA_CALIBRATE);
  mouvPatte((3+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, COXA_CALIBRATE);
  delay(vitesse);
  mouvPatte((1+mode)%6,TIBIA_CALIBRATE, 350, COXA_CALIBRATE);
  delay(vitesse);
  mouvPatte((1+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, COXA_CALIBRATE);
  delay(vitesse);
  mouvPatte((5+mode)%6,TIBIA_CALIBRATE, 350, COXA_CALIBRATE);
  delay(vitesse);
  mouvPatte((5+mode)%6,TIBIA_CALIBRATE, FEMUR_CALIBRATE, COXA_CALIBRATE);
  delay(vitesse);
}
```

Les positions ont été ajustées de façon empirique jusqu'à obtenir le mouvement souhaité, les positions *CALIBRATE* sont les positions de base, quand le robot se tient debout, que nous avons définies globalement. Le paramètre *vitesse* défini le délai entre chaque changement de position et donc la vitesse de déplacement de l'hexapode. Le paramètre *mode* permet de choisir dans quel sens le robot fait un pas, le mode étant le numéro de patte qui sera considéré comme la patte avant lors du mouvement. Les paramètres à sélectionner pour *mode* sont définies globalement. Pour leurs noms, on a pris comme référence le moteur 0 comme moteur avant.

```
#define AVANT 0
#define AVANT_GAUCHE 1
#define AVANT_DROITE 5
#define ARRIERE 3
#define ARRIERE_GAUCHE 4
#define ARRIERE_DROITE 2
```

Vous pourrez trouver un schéma explicatif du fonctionnement des modes en annexe page 26.

Nous pouvons ensuite créer d'autres fonctions de déplacement comme par exemple *tourneGauche()* ou *tourneDroite()*, cette dernière dont vous pouvez voir le code en annexe page 27. Pour pouvoir aller plus loin dans le déplacement, nous avons par la suite ajouter la programmation des capteurs ultrason.

2. Les capteurs

Les capteurs à ultrasons fonctionnement assez facilement. En plus de l'alimentation et de la masse, les capteurs ont une broche TRIGGER et une broche ECHO. Pour envoyer le signal ultrason, il suffit d'activer la broche TRIGGER qui envoi un signal, ce signal est alors réfléchi sur l'objet le plus proche et retourne au capteur sur la broche ECHO. En mesurant le temps séparant l'envoi et la réception du signal et en connaissant la vitesse du son, on peut alors déterminer la distance qui sépare le capteur d'un mur ou d'un obstacle.

Nous créons la structure *capteurs* définie en global, qui nous permettra de stocker les distances de nos 3 capteurs puis avec la fonction *getCapteurs()* on calcule et enregistre ces distances dans la structure.

```
struct capteurs { //structure donnant accès aux valeurs des 3 capteurs
  int avant;
  int droite;
  int gauche;
} capteurs;

struct capteurs cap;

void getCapteurs()
{
  digitalWrite(TRIGGER_AVANT, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_AVANT, LOW);
  long measure_avant = pulseIn(ECHO_AVANT, HIGH, MEASURE_TIMEOUT);
  cap.avant = measure_avant / 20.0 * SOUND_SPEED;

  digitalWrite(TRIGGER_DROITE, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_DROITE, LOW);
  long measure_droite = pulseIn(ECHO_DROITE, HIGH, MEASURE_TIMEOUT);
  cap.droite = measure_droite / 20.0 * SOUND_SPEED;


  digitalWrite(TRIGGER_GAUCHE, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_GAUCHE, LOW);
  long measure_gauche = pulseIn(ECHO_GAUCHE, HIGH, MEASURE_TIMEOUT);
  cap.gauche = measure_gauche / 20.0 * SOUND_SPEED;
}
```

On peut alors créer des déplacement plus complexes pour l'hexapode comme par exemple la fonction *longeMur()* qui avance parallèlement à un mur, situé à la droite du robot, s'écarte si le robot est trop près du mur et se rapproche si il en est trop loin.

```
void longeMur(int vitesse)
{
  getCapteurs();
  if(cap.droite<15)
  {
    faireUnPas(AVANT_GAUCHE,vitesse);
  }
  else if(cap.droite>25)
  {
    faireUnPas(AVANT_DROITE,vitesse);
  }
  else
  {
    faireUnPas(AVANT,vitesse);
  }
}
```

Cette simple fonction recalcule à tout moment les états des capteur et agis en conséquence. Si l'on est trop proche du mur à notre droite, on se décale vers la gauche et si l'on est trop éloigné du mur, on se décale à droite. Sinon nous continuons tout droit.

Avec le code permettant de récupérer les états des capteurs et les fonctions de déplacement, nous pouvons maintenant imaginer toutes sortes de trajectoires pour le robot.

3. Le RSSI

Pour faire interagir l'ESP8266-01 avec l'Arduino et récupérer la valeur du RSSI, nous commençons par utiliser la liaison série UART de l'Arduino. Lorsque celle ci est connectée en USB à un ordinateur, nous utilisons le moniteur série de l'IDE pour communiquer. La vitesse de transmission est de 115200 bauds et le mode de transmission est New Line et Carriage Return. Pour communiquer avec l'ESP il faut utiliser les commandes AT. Pour commencer on peut envoyer la simple commande "AT" et si l'ESP est correctement mis en place, il nous répond simplement "OK". Si la connexion est bien établie, on peut alors passer à des commandes plus intéressantes. En envoyant par exemple "AT+CWLAP" on obtient la liste des réseaux WiFi à proximité ainsi que des informations sur ces réseaux dont notamment le RSSI (troisième paramètre de chaque réseau WiFi).

```
AT+CWLAP
+CWLAP:(0,"eduroam",-90,"bc:f1:f2:7f:41:e8",1,-46)
+CWLAP:(0,"PolytechLille",-90,"bc:f1:f2:7f:41:e4",1,-46)
+CWLAP:(0,"PolytechLilleStaff",-88,"bc:f1:f2:7f:41:e7",1,-46)
+CWLAP:(4,"PolytechGuests",-89,"bc:f1:f2:7f:41:e3",1,-46)
+CWLAP:(0,"PolytechLille",-51,"c4:14:3c:38:2c:a2",2,-44)
+CWLAP:(0,"eduroam",-52,"c4:14:3c:38:2c:a6",2,-44)
+CWLAP:(4,"PolytechGuests",-52,"c4:14:3c:38:2c:a1",2,-44)
+CWLAP:(0,"PolytechLilleStaff",-51,"c4:14:3c:38:2c:a5",2,-44)
+CWLAP:(0,"LILLE1",-91,"bc:f1:f2:7f:41:e0",1,-46)
+CWLAP:(0,"PolytechLille",-87,"c8:00:84:84:00:52",4,-46)
+CWLAP:(0,"eduroam",-91,"c8:00:84:84:00:56",4,-44)
+CWLAP:(4,"PolytechGuests",-88,"c8:00:84:84:00:51",4,-46)
+CWLAP:(0,"PolytechLilleStaff",-90,"c8:00:84:84:00:55",4,-46)
```

Figure 11 : Réponse à la commande AT+CWLAP

Avec cette méthode, nous pouvons afficher le RSSI sur le moniteur série cependant il est difficile de récupérer cette valeur dans une variable pour la traiter ensuite. Pour cela nous avons mis en place un autre moyen d'obtenir le RSSI.

Pour cela, nous connectons notre ESP sur les pins 6 et 7 de l'Arduino et nous utilisons la bibliothèque *SoftwareSerial* pour les utiliser comme liaison série. Nous utilisons ensuite la bibliothèque *WiFiEsp* qui contient notamment une fonction renvoyant le RSSI d'un signal WiFi. Après avoir défini la vitesse de transmission avec la fonction *setEspBaudRate()*, la fonction *mesureRSSI()* vérifie la connexion au réseau WiFi avec lequel on désire travailler et se reconnecte si nécessaire. Nous pouvons ensuite mesurer le RSSI du signal avec un appel à la fonction *RSSI()*. On peut alors stocker dans une variable la valeur du RSSi à cet instant. Vous pouvez trouver le code correspondant en annexe page 28.

Cette méthode fonctionne mais les mesures de RSSI prennent un peu de temps car l'utilisation de ports classique comme liaison série est moins stable qu'une liaison série réelle et il faut donc plusieurs essais avant que la mesure du RSSI ne soit faite correctement. Malgré ce petit bémol nous arrivons tout de même à obtenir la valeur du RSSI d'un réseau WiFi au choix que nous stockons dans une variable.

Il reste maintenant l'exploitation de cette valeur. Pour cela, il est possible de créer avec l'ESP8266 un serveur web. Nous avons pu le faire dans un premier temps avec les commandes AT sur le moniteur série. Voici quelques-unes des commandes utiles à ce processus.

AT+CWJAP="SSID","MOTDEPASSE" pour se connecter à un réseau WiFi

AT+CIFSR pour obtenir l'adresse IP locale sur laquelle sera mis en place le serveur web

AT+CIPMUX=1 pour passer le serveur en mode multithreadé

AT+CIPSERVER=1,2020 pour créer le serveur sur le port 2020

Le serveur est alors mis en place et lorsque l'on rentre l'adresse IP et le port choisi dans un navigateur sur le même réseau WiFi, une connexion TCP est initialisée. Une requête HTTP est alors envoyée par le navigateur au serveur web. Cette trame reçue par l'ESP est de la forme : **+IPD, ID, LEN: suivi de la requête HTTP** où ID est le numéro de connexion et LEN le nombre d'octets reçus.

On peut alors envoyer une réponse à cette requête avec la commande suivante.

AT+CIPSEND=ID,LEN: suivi de la réponse HTTP où ID est le numéro de connexion est LEN le nombre d'octets à envoyer

Finalement pour arrêter la connexion, on peut utiliser la commande :

AT+CIPCLOSE

Avec cette méthode, on peut transmettre un simple code HTML affichant par exemple sur un site internet la valeur mesurée en directe du RSSI. Pour cela il faudrait créer un programme qui communique directement avec l'ESP sans passer par le moniteur série pour créer le serveur web.

V - Bilan et ouverture

Aujourd'hui, alors que notre projet prend fin, notre robot est capable de se déplacer de façon autonome. Grâce à ses capteurs il peut éviter un obstacle, détecter les murs et par exemple les longer. Nous sommes également capables de lire le flux RSSI et nous avons été capable de mettre en place un serveur web pour le traitement de cette donnée.

Cependant à deux jours de la fin du projet, nous n'avons plus réussi à communiquer avec l'ESP8266. Nous n'avons pas eu le temps de comprendre d'où venait le problème ou comment le résoudre.

Concernant la conception du robot, celui ci n'est pas capable de se lever lui même. En effet les servomoteurs ne sont pas assez puissants par rapport au poids de l'hexapode. De même il n'est malheureusement pas capable de monter des escaliers. Par ailleurs nous avons remarqué que les moteurs consomment beaucoup d'énergie et nous avons très souvent dû recharger nos piles. En effet dès que les piles commençaient à faiblir, le moteurs avaient du mal à rester en position et le robot avait vite tendance à s'affaisser lors de son déplacement.

Si nous devions recommencer, nous changerions les servomoteurs pour des plus puissants afin d'éliminer le problème de poids et pouvoir travailler sur la prise d'escalier.

Si nous avions plus de temps, nous programmerions un code permettant la mise en place d'un serveur web pour lire en direct les données prise par le robot. Nous travaillerions également sur la solution de positionnement en mettant en place notre idée de marqueurs QR code avec une Raspberry Pi.

Finalement, dans une optique de produit fini, le robot pourrait être équipé d'une "user friendly" interface sur smartphone ou ordinateur permettant de le configurer, de le diriger manuellement, de visualiser en direct la carte de bâtiment avec l'intensité du wifi.

Conclusion

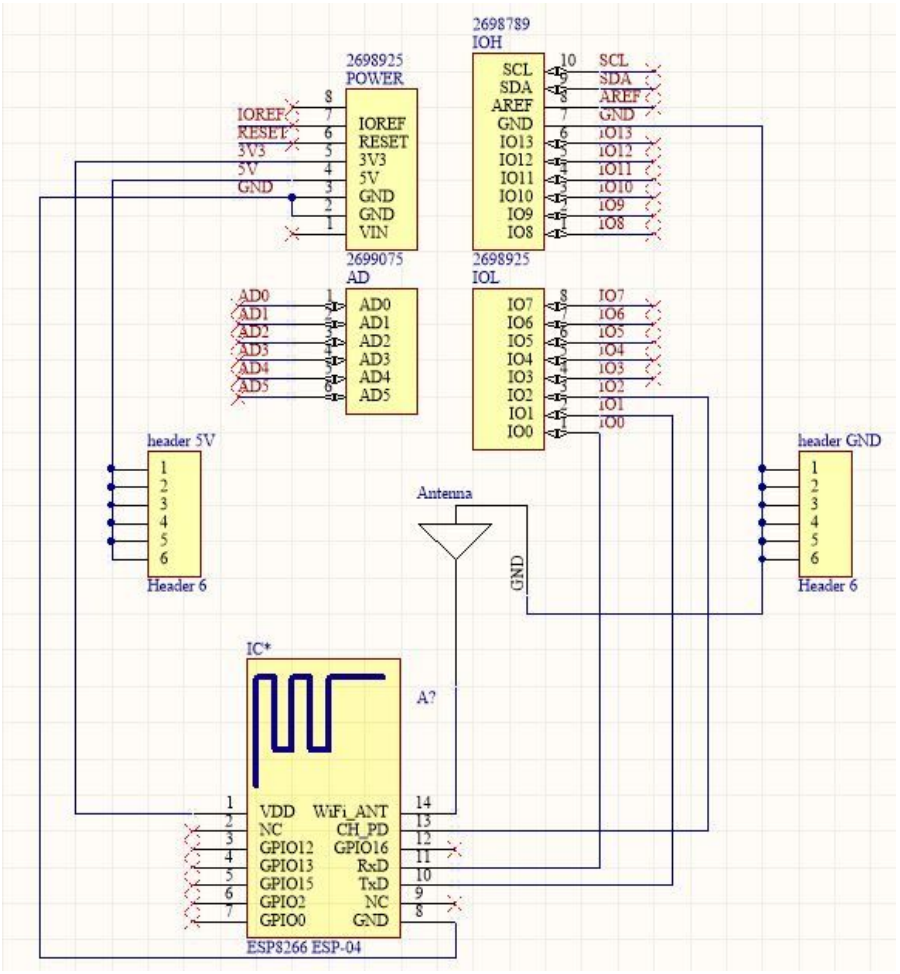
Ce projet suscita chez nous un réel intérêt. La pluralité des domaines abordés (modélisation 3D, impression 3D, découpe laser, conception de PCB, prototypage de PCB, programmation, …) nous a permis de ne jamais nous ennuyer, d'apprendre et de gagner en compétences tout au long du projet.

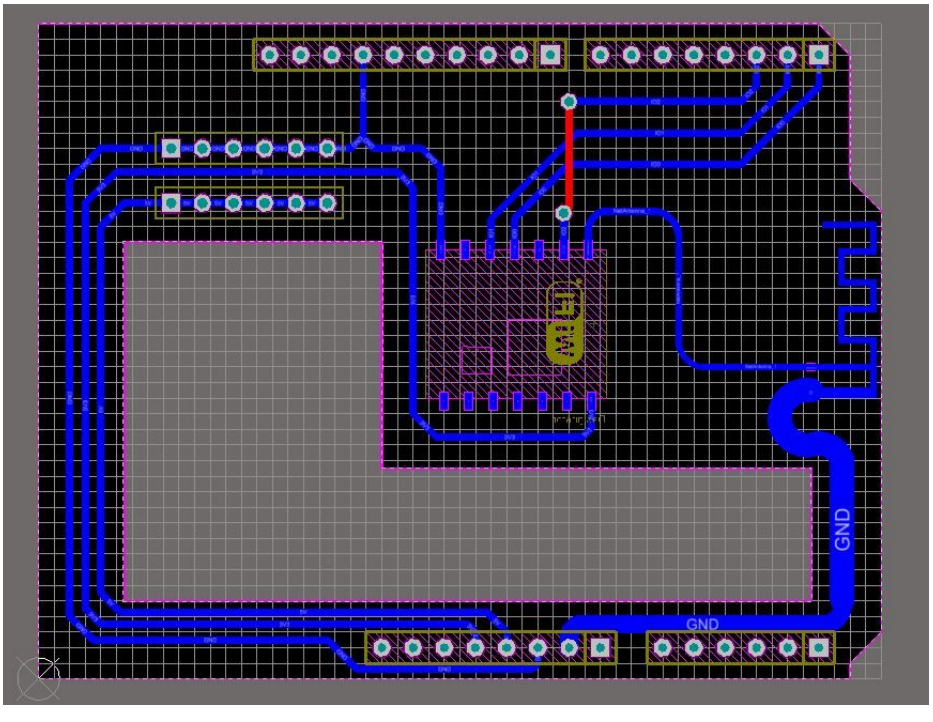
La partie carte WiFi reste notre plus grand regret, nous somme déçus de n'avoir pas pu faire fonctionner l'ESP-04 après autant d'heures passées dessus mais nous ne désespérons pas pour autant. Nous sommes aujourd'hui plus expérimentés en conception de pcb, notamment concernant les propriétés physique des éléments des piste d'un pcb après avoir travaillé sur l'antenne, et il n'est pas dit que nous ne concevions pas dans le futur une nouvelle carte qui cette fois fonctionnera.

Nous savions dès le départ que le sujet était ambitieux et bien que nous n'avons pas réussi à aller au bout, nous sommes tout de même fiers de ce que nous avons réussi à réaliser.

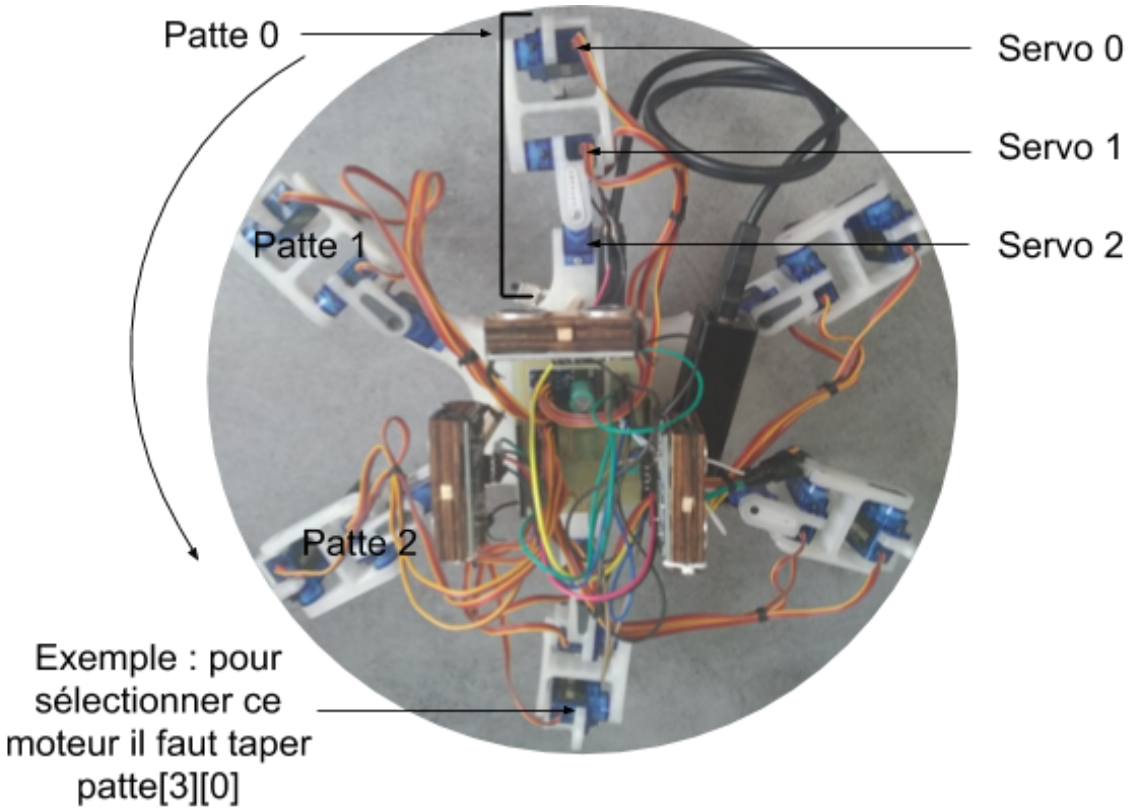
ANNEXES

Fichiers Schematic et PCB de notre shield sous Altium





Numérotation des pattes et servomoteurs

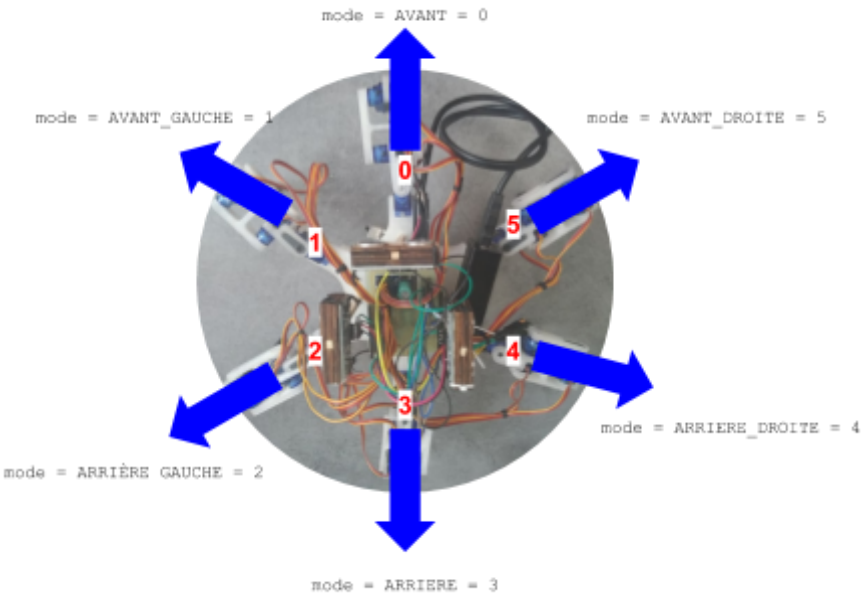


Explication du fonctionnement des *mode* de la fonction *faireUnPas()*

Tous les mouvements sont pensés de base avec la patte 0 étant considérée comme la “tête” du robot (*mode*=0).
Dans la fonction *faireUnPas()*, les mouvements de patte sont cependant paramétrés par *mode* qui défini la patte considérée comme patte avant.
Partie du code :

```
mouvPatte((0+mode)%6,TIBIA_CALIBRATE, 350, 230);
```

En incrémentant avec *mode* le numéro de la patte dans les appels de *MouvPatte()*, on décale dans le sens trigonométrique toutes les pattes et donc on décale la direction du pas.
Le modulo 6 permet une rotation cyclique, ainsi la patte 5 devient la patte 0 lors de l’incrémentation.



Code *tourneDroite()*

```
void tourneDroite(int vitesse){
      mouvPatte(1,TIBIA_CALIBRATE,350,230);
      mouvPatte(4,TIBIA_CALIBRATE,350,230);
      delay(vitesse);
      mouvPatte(1,TIBIA_CALIBRATE,FEMUR_CALIBRATE,230);
      mouvPatte(4,TIBIA_CALIBRATE,FEMUR_CALIBRATE,230);
      delay(vitesse);
      mouvPatte(2,TIBIA_CALIBRATE,350,230);
      mouvPatte(5,TIBIA_CALIBRATE,350,230);
      delay(vitesse);
      mouvPatte(2,TIBIA_CALIBRATE,FEMUR_CALIBRATE,230);
      mouvPatte(5,TIBIA_CALIBRATE,FEMUR_CALIBRATE,230);
      delay(vitesse);
      mouvPatte(0,TIBIA_CALIBRATE,350,230);
      mouvPatte(3,TIBIA_CALIBRATE,350,230);
      delay(vitesse);
      mouvPatte(0,TIBIA_CALIBRATE,FEMUR_CALIBRATE,230);
      mouvPatte(3,TIBIA_CALIBRATE,FEMUR_CALIBRATE,230);
      delay(vitesse);
      all(TIBIA_CALIBRATE,FEMUR_CALIBRATE,COXA_CALIBRATE);
      delay(vitesse);
}
```

Code de mesure du RSSI

```
#include "WiFiEsp.h"

char ssid[] = SECRET_SSID;    //  your network SSID (name)
char pass[] = SECRET_PASS;   // your network password

WiFiEspClient  client;

#include "SoftwareSerial.h"
SoftwareSerial Serial1(6, 7);
#define ESP_BAUDRATE  115200


void setup() {
  //Initialize serial and wait for port to open
  Serial.begin(9600);

  // initialize serial for ESP module
  setEspBaudRate(ESP_BAUDRATE);

  Serial.print("Searching for ESP8266...");
  // initialize ESP module
  WiFi.init(&Serial1);

}

void loop() {
  mesureRSSI();
  delay(5000);
}

// This function attempts to set the ESP8266 baudrate. Boards with additional hardware serial ports
// can use 115200, otherwise software serial is limited to 19200.
void setEspBaudRate(unsigned long baudrate){
  long rates[6] = {115200,74880,57600,38400,19200,9600};

  Serial.print("Setting ESP8266 baudrate to ");
  Serial.print(baudrate);
  Serial.println("...");

  for(int i = 0; i < 6; i++){
    Serial1.begin(rates[i]);
    delay(100);
    Serial1.print("AT+UART_DEF=");
    Serial1.print(baudrate);
    Serial1.print(",8,1,0,0\r\n");
    delay(100);
  }
  Serial1.begin(baudrate);
}


void mesureRSSI(){
  Serial.println("Prise de mesure\n");
  // Connect or reconnect to WiFi
  if(WiFi.status() != WL_CONNECTED){
    //Serial.print("Attempting to connect to SSID: ");
    //Serial.println(SECRET_SSID);
    while(WiFi.status() != WL_CONNECTED){
      WiFi.begin(ssid, pass);  // Connect to WPA/WPA2 network. Change this line if using open or WEP network
      //Serial.print(".");
      delay(100);
    }
    //Serial.println("\nConnected");
  }

  int rssi = 0;
  while(rssi==0){
    rssi = WiFi.RSSI();
  }
  Serial.print("RSSI : ");
  Serial.println(rssi);
}
```