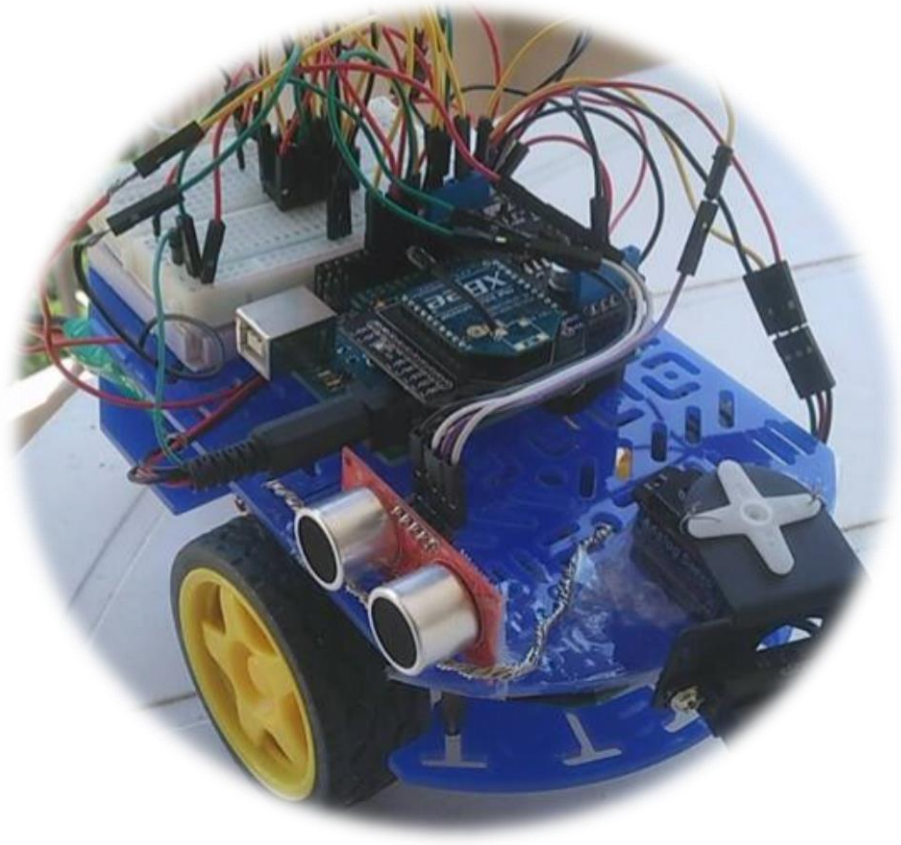
UPSSITECH
ÉCOLE D'INGÉNIEURS INTERNE

Année 2016 - 2017

Projet Xion
Rapport Final



PARTIE 3

CARTOGRAPHIE ROBOTIQUE D'UNE SALLE

Enseignant responsable :
Michaël LAUER

CALESTANI Brice | CORTACERO Kévin | MORANT Thibaut
PECCHIOLI Mathieu | SAUVAGE Thomas

Table des matières

I. Introduction

Le projet que nous allons vous présenter dans ce dossier a été développé dans le cadre d'un projet de robotique des étudiants de première année de l'UPSSITECH en spécialité Systèmes Robotiques et Interactifs (SRI). Ce projet, rentrant dans les maquettes pédagogiques sous le nom de projet fil rouge portera le nom de Projet Xion conformément aux vœux de l'équipe d'étudiant y ayant travaillé dessus.

L'objectif principal de ce développement est la « Réalisation d'un robot mobile autonome de cartographie ».

II. Rappel du cahier des charges

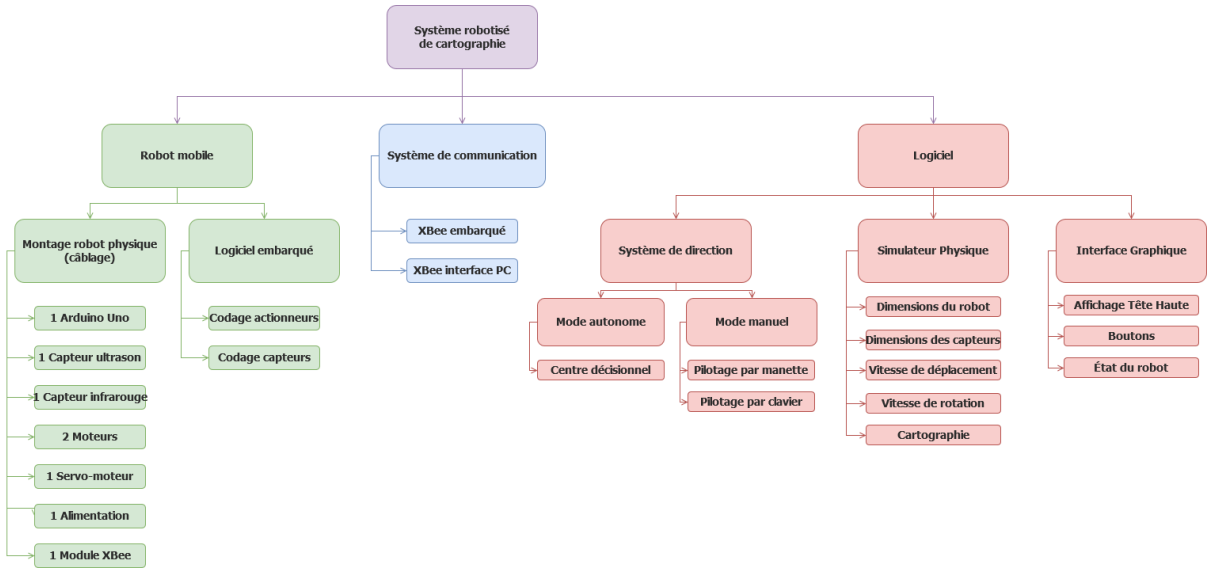
Suite à la réception du cahier des charges par notre client, nous avons réalisé un tableau récapitulatif des fonctions contraintes que nous devions respecter. Nous avons également pris la décision d'enrichir les demandes initiales avec des compléments utiles à l'utilisateur :

Fonction	Description	Flexibilité
FP1	Le robot doit suivre les contours de la pièce	0
FP2	Le logiciel doit disposer d'une interface graphique cartographiant la pièce en temps réel	0
FP3	Le logiciel doit disposer d'une partie décisionnelle autonome (non embarqué sur le robot)	0
FP4	Le logiciel doit disposer d'un mode de commande manuel	0
FP5	Le logiciel doit permettre d'afficher l'état des capteurs du robot pour l'utilisateur	0
FP6	La communication entre le robot et le logiciel doit être distante	0
FP7	Le robot doit embarquer un système anticollisions	0
FS1	Le logiciel peut disposer d'un mode de simulation autonome	2

III. Analyse du cahier des charges

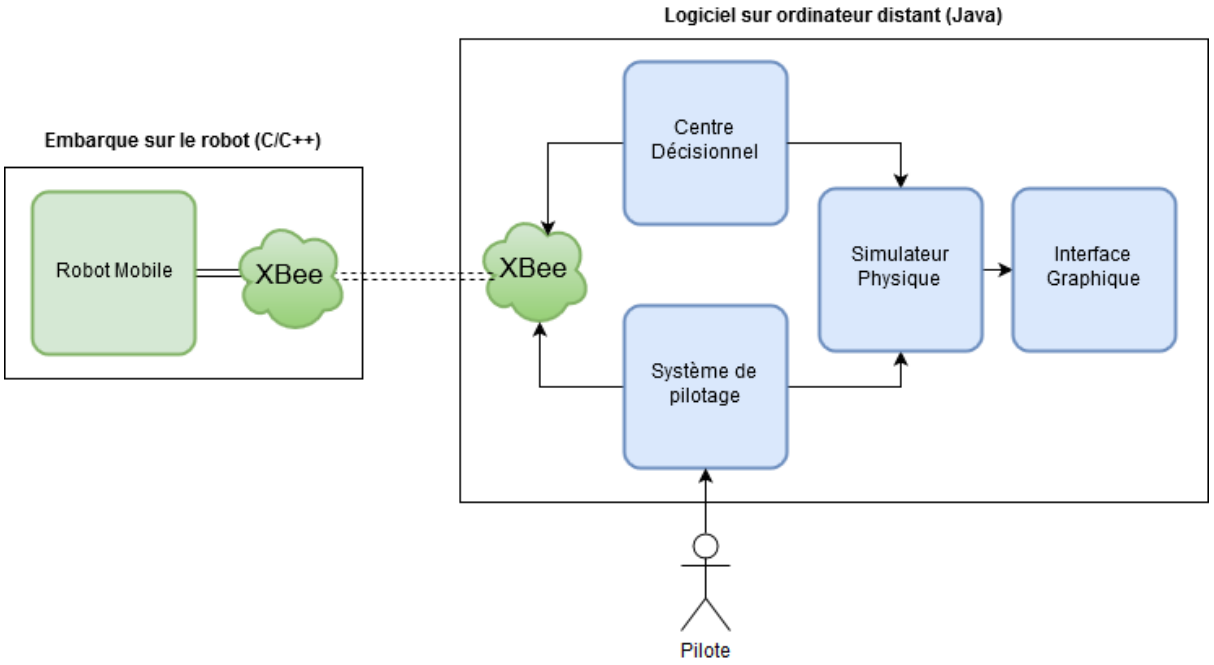
1. Analyse produit

La première étape de l'analyse du cahier des charges a été, comme nous l'avons vu précédemment, de reformuler ce dernier de manière à n'obtenir un récapitulatif des besoins à remplir. Dans un second temps, la suite logique de l'analyse a consisté en la réalisation d'une analyse produit. Pour ce faire, nous avons pris la décision de réaliser un arbre produit résumant l'ensemble des éléments constituants du produit final. Voici cet arbre :



2. Communication des composants

Durant la partie précédente, nous avons réalisé l'analyse de la composition de notre produit. Dans la logique des choses, nous nous sommes donc ensuite intéressé à la réalisation d'une architecture modulaire nous permettant de remplir l'ensemble des tâches demandées. Cependant, Nous avons souhaité en plus de ce cahier des charges rendre notre produit le plus modulaire possible. De ce fait, l'architecture que nous avons développée s'agence tel que présenté ci-dessous :



Les différentes parties sont :

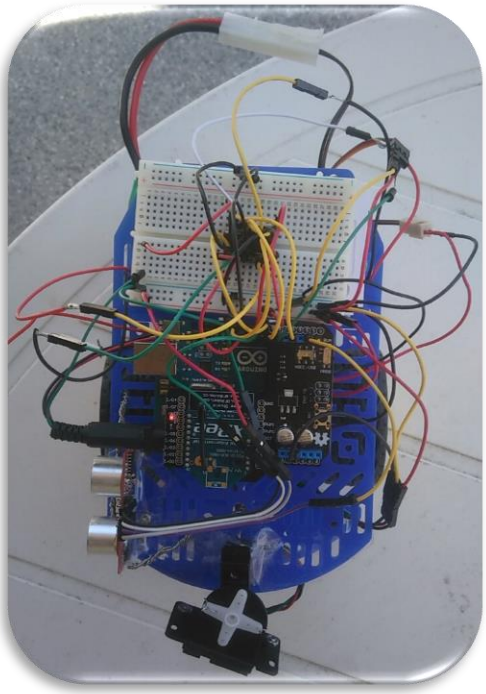
- *Robot mobile :* partie aussi bien matérielle que logicielle, elle comprend le montage du robot mobile ainsi que le développement du contrôle des actionneurs et des capteurs.
- *Xbee :* composé d'un point de vue matériel de deux xbee montés soit du cotés robot mobile sur un Shield Arduino soit sur un adaptateur USB cotés ordinateur, cette partie correspond à la liaison dématérialisée entre les deux blocs que sont le robot mobile et l'ordinateur.
- *Centre décisionnel :* ce module permet lorsque l'on est en mode automatique de définir le comportement du robot mobile à partir des données des capteurs
- *Simulateur physique :* véritable antichambre de l'interface graphique, le simulateur physique permet de générer la cartographie de la pièce ainsi que la position du robot. Il effectue la conversion entre le monde réel et les données virtuelles perçues.
- *Interface Graphique :* lien avec l'utilisateur, l'interface graphique permet d'informer l'opérateur et de lui fournir l'ensemble des données dont il a besoin pour commander en mode manuel par exemple
- *Système de pilotage :* utilisé en mode manuel, le système de pilotage permet de faire de lien entre les commandes entrée par l'utilisateur et les informations que le robot doit appliquer.

Les flèches de notre schéma représentent l'ensemble des communications entre les modules. Ces données transmises permettent à chacun de réaliser son travail de manière indépendant des autres.

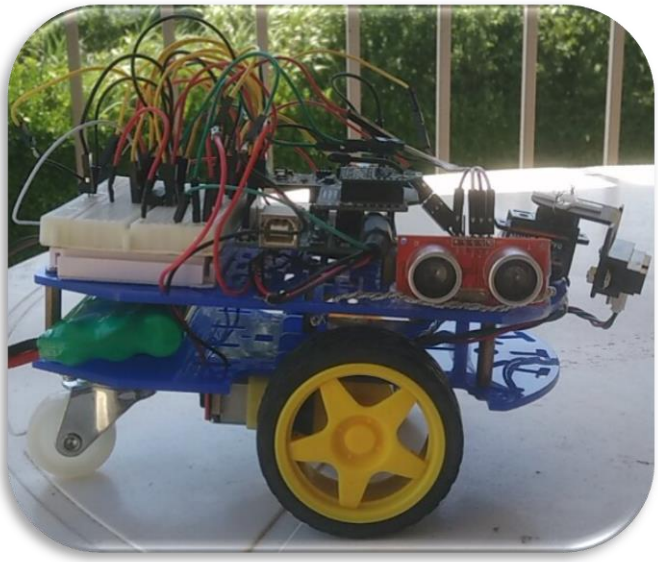
3. Listes des composants

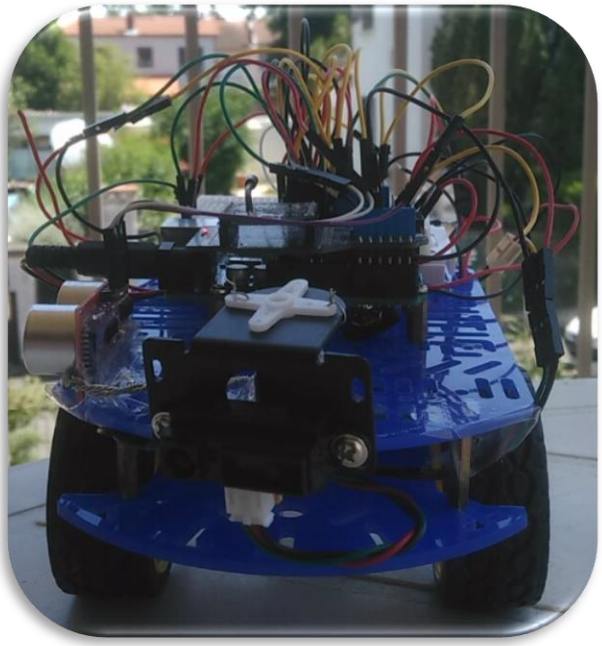
Pour mener à terme le projet Xion, nous avons eu besoin d'une partie matérielle. Voici donc la liste de l'ensemble des composants utilisés :

- ✓ 1 châssis robotique composer de deux plateformes espacées avec des entretoises, deux roues fixes montées sur des moteurs à courant continu et une roues folles pour obtenir une meilleure stabilité
- ✓ 1 batterie 7V – 2A
- ✓ 1 planche de prototypage type breadboard
- ✓ 1 servomoteur de petite puissance
- ✓ 1 capteur de distance ultrasons
- ✓ 1 capteur de distance infra-rouge
- ✓ 1 support en équerre pour le capteur infra-rouge et sa visserie
- ✓ 1 pont en H de type L293
- ✓ 1 Arduino Uno
- ✓ 1 Shield Xbee pour Arduino Uno
- ✓ 1 adaptateur USB pour Xbee
- ✓ 2 puces Xbee
- ✓ Des câbles jumpers de prototypage rapide



L'ensemble de ses composants monté sur le robot mobile nous donnent un rendu final comme présenté sur les deux photos suivantes :





4. Choix technologiques matériel

Afin de réaliser les fonctionnalités voulues et nécessaire au projet Xion, nous avons dû faire des choix technologiques. Chaque choix a été réfléchis de manière à optimiser soit la fiabilité du produit final, soit la facilité de mise en œuvre.

Prenons tout d'abord le cas des capteurs de distance. Nous aurions pu utiliser deux capteurs de distance de même type mais nous avons décidé de tirer parti de deux technologies que sont le capteur infra-rouge et le capteur ultrason de manière à utiliser le premier en qualité de parechoc avant et le second pour du mapping latéral. Afin de renforcer l'efficacité de notre parechoc avant et pour supprimer les angles morts, nous avons monté le capteur avant (capteur IR) sur un servomoteur vertical afin de réaliser un balayage. La fiabilité de la distance obtenue avec un capteur ultrason latéral a également été choisie.

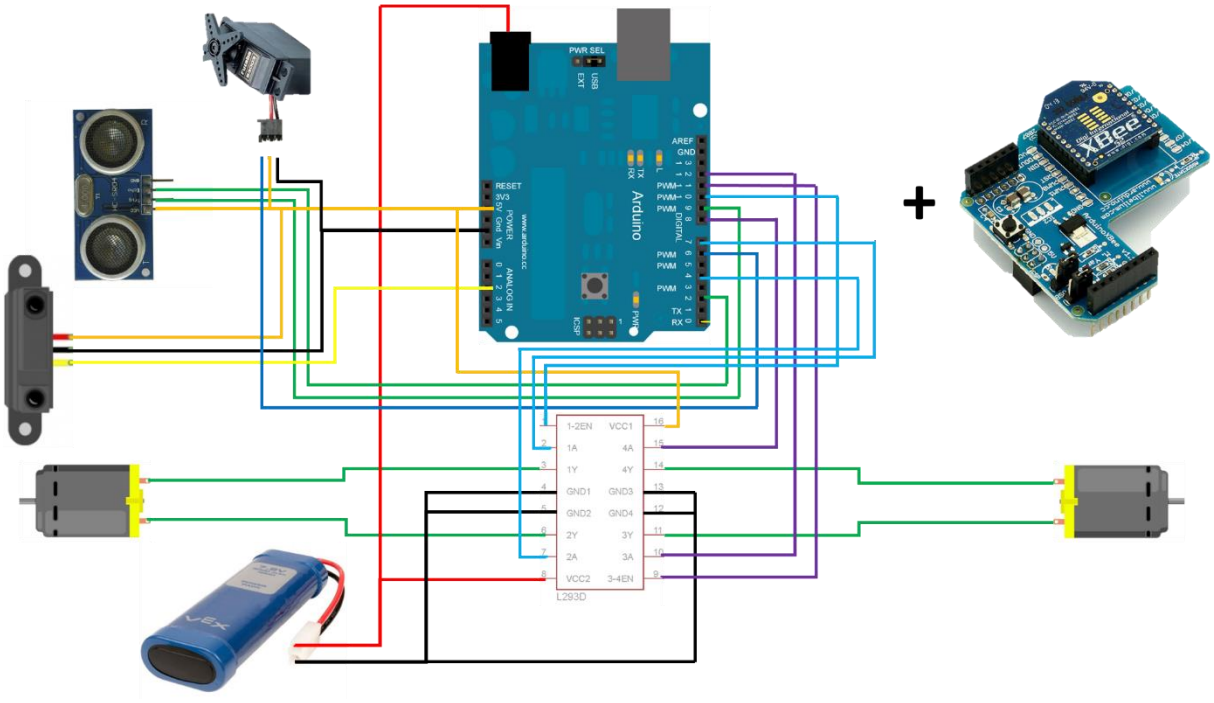
Concernant l'utilisation d'une planche de prototypage et de jumper, cette solution a permis à l'équipe de développement de tester plusieurs configurations de matériel avec un temps de montage/démontage négligeable.

Pour l'autonomie, une batterie 7V – 2A permet un rechargement plus accessible comparé à un rack de piles. De plus, une fois chargé, l'autonomie globale du robot mobile s'en est vu grandement augmenté permettant la réalisation de cession de test bien plus longues.

IV. Conception

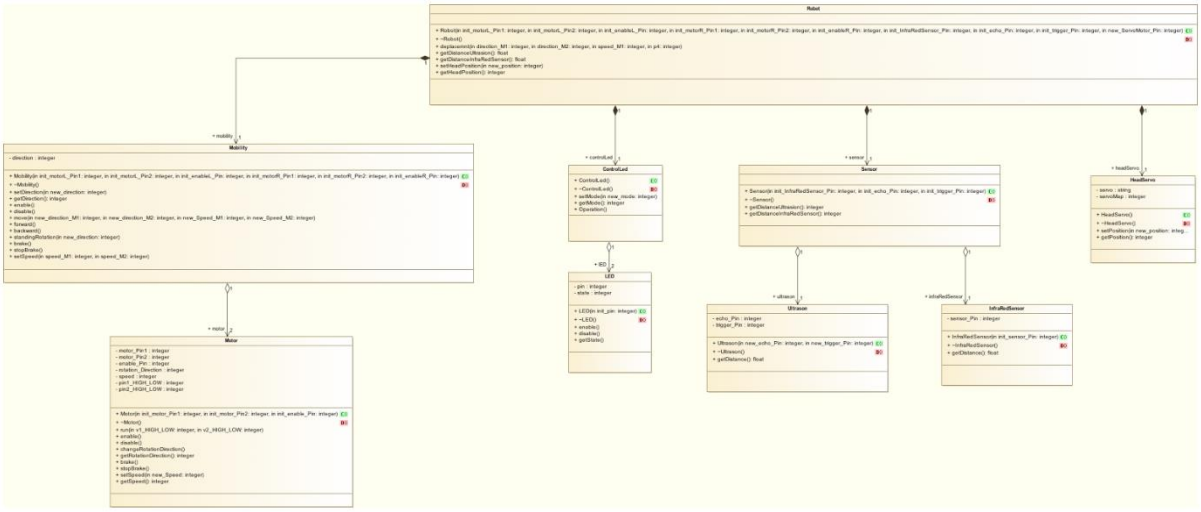
1. Câblage du robot mobile

Une fois l'ensemble des éléments de matériel défini, nous avons pu réaliser le montage de la plateforme robotique autonome. Afin de simplifier les explications de montage, nous préférons vous montrer un schéma global des interconnexions :



Ce schéma est disponible en *ANNEXE 1* pour une meilleure visibilité. Il est à noter que le montage du Shield Xbee sur l'Arduino Uno n'a pas été représenter pour faciliter la compréhension.

2. Programmation Robot mobile



Pour la réalisation du programme contenu dans l'Arduino, il était plus pertinent de concevoir directement une bibliothèque spécifique au robot permettant ainsi d'obtenir un code propre et structuré, mais aussi de partir à la découverte du C++.

Par ailleurs, il existe un Sketch qui est utilisé comme main afin de permettre une communication entre le robot et le centre décisionnel.

Pour la structure de la bibliothèque, il était intéressant de partir sur une classe, Robot, qui regroupe toutes les fonctionnalités du robot. Ainsi on obtient trois autres classes, Mobility, ControlLed, Sensor et HeadServo.

La classe Mobility est utilisé pour contrôler le déplacement du robot grâce à deux instances de la classe Motor, elle offre la possibilité de se déplacer en marche avant, marche arrière, tourner à droite ou à gauche en modifiant uniquement la vitesse ou en inversant le sens de rotation des moteurs. Il est aussi possible de s'arrêter en roue libre mais aussi en frein magnétique.

La classe ControlLed, permet de gérer l'affichage des différents modes d'utilisation du robot, c'est à dire, le mode manuel ou le mode automatique grâce à deux instances de la classe LED. La fonctionnalité est implémentée mais comme le robot n'est pas encore équipé de LEDs, elle n'est pas utilisée.

La classe Sensor, permet de gérer le retour des capteurs ultrason et infrarouge grâce au deux classes qu'ils leurs sont associés. L'ultrason est utilisé pour détecter des objets survenant du côté droit mais aussi de suivre un mur. L'infrarouge quant à lui, monté sur un servomoteur va servir uniquement de capteur tout ou rien afin de détecter les objets venant de face.

La classe HeadServo, est utilisé afin d'obtenir un champ de détection plus large pour les objets venant de face. Pour cela l'utilisation d'une instance de la classe servo est utilisé.

Il est à noter que le diagramme des classes participantes montré au début de cette partie est disponible en *ANNEXE 2* pour une meilleure lisibilité.

3. Communication

Le module de communication utilise comme support matériel les circuits Xbee ainsi que leur protocole de communication ZigBee. L'avantage de cette technologie est qu'une fois paramétré à l'aide du logiciel d'analyse de paquet dédié XCTU, pour que les deux circuits se voient et puissent communiquer, l'envoi et la réception de donnée cotés Arduino et cotés ordinateur se font par un pilotage standard de port série.

Une fois l'envoi de donnée mis en place, le gros du travail a consisté en la réalisation des trames de données sécurisés de manière à donner un sens aux informations transmises. Faisons tout d'abord un état des lieux des données transitant dans chaque sens :

- De l'Arduino vers le PC :
  - La distance du capteur ultrason
  - La distance du capteur infra-rouge
  - L'angle du servomoteur réalisant le balayage
- Du PC vers l'Arduino :
  - Le mode de fonctionnement manuel ou automatique
  - La consigne de vitesse du moteur droit
  - La consigne de vitesse du moteur gauche
  - Le sens de rotation du moteur droit
  - Le sens de rotation du moteur gauche

Afin d'obtenir la modularité la plus grande, nous avons fait le choix d'assigner une trame unique pour chaque donnée. Nous avons donc un total de huit trames possibles transitant par le Xbee. La structure d'une trame est basée sur le protocole RS232 avec un caractère de début de trame et un caractère de fin de trame. La structure générique peut être résumé tel que suit :

Caractère de début* | *Type de trame* | *Donnée sur un char* | *Caractère de fin

Le tableau suivant résume les numéros de chaque type de trame :

Type de trame	Numéro d'identification
Distance Ultrasons	0
Distance Infrarouge	1
Angle servomoteur	2
Mode de fonctionnement	0
Consigne de vitesse pour le moteur droit	1
Consigne de vitesse pour le moteur gauche	2
Sens de rotation pour le moteur droit	3
Sens de rotation pour le moteur gauche	4

L'envoi de l'ensemble des informations se fait de manière automatique et répété selon une période paramétrable. Actuellement, le temps de transmission entre deux trames de même type est d'environ cent millisecondes.

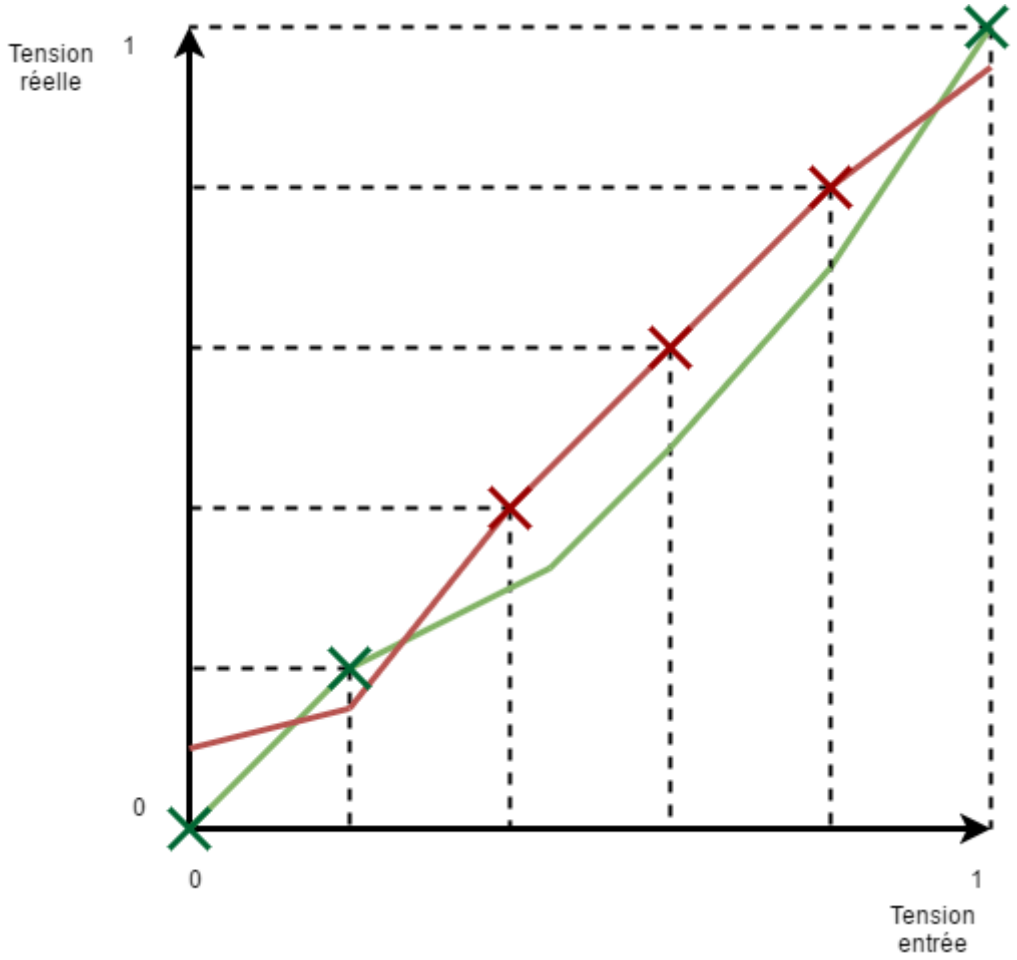
4. Etalonnage

Le moteur de chaque roue ne sont parfaitement identiques. Ainsi, même si on envoi une tension identique dans le moteur de gauche et dans le moteur de droite, on n'assure pas que la vitesse de rotation soit identique et le robot n'aura pas de trajectoire rectiligne.

Il est nécessaire pour effectuer chaque mouvement de réaliser un étalonnage des moteurs, c'est à dire, trouver pour chaque valeur de tension d'un moteur, la valeur de tension de l'autre moteur correspondant à une vitesse de rotation. Afin de réaliser cet étalonnage, nous sélectionnons plusieurs valeurs de tensions centrales entre la 40% et 100% de la tension maximale. Nous modifions alors la tension de la roue droite ou de la roue gauche de telle sorte que soit la roue droite, soit la gauche soit alimenté par la tension centrale, la tension de l'autre roue sera choisi par l'utilisateur de telle sorte que l'utilisateur considère que le robot va suffisamment droit (pour la trajectoir linéaire par exemple). Il est nécessaire de réaliser un étalonnage entre les 2 roues lorsqu'elles ont une tension d'entrée même parité et de parités opposés, c'est pour cela que l'on réalise un étalonnage pour 4 types de mouvement : en avant (FORWARD), en arrière (BACK), rotation sur place droite (RIGHT ROT), rotation sur place gauche (LEFT ROT). Le robot est considéré comme étalon lorsqu'il réussit à aller droit lorsque l'utilisateur ne parvient pas à percevoir de décalage sur la trajectoire rectiligne sur 5 ou 6 mètre et une rotation est considérée comme bien étalonné si le robot fait 2 ou 3 tours sur lui même sans modifier la position du point central entre les 2 roues du robot.

```
*****************************************************************************************
*                                      CALIBRATION                                      *
*****************************************************************************************
*|||--> OFF                                                                             *
*****************************************************************************************
*/-----------------------------------------------------------------------------------\*
*|0 - FORWARD   / C = 0.4 \    | L: 0.4 |  [>>>]  | R: 0.4 |                          |*
*\-----------------------------------------------------------------------------------/*
* 1 - FORWARD   / C = 1.0 \    | L: 1.0 |  [>>>]  | R: 1.0 |                           *
* 2 - BACK      / C = 0.4 \    | L: 0.4 |  [>>>]  | R: 0.4 |                           *
* 3 - BACK      / C = 1.0 \    | L: 1.0 |  [>>>]  | R: 1.0 |                           *
* 4 - LEFT ROT  / C = 0.4 \    | L: 0.4 |  [>>>]  | R: 0.4 |                           *
* 5 - LEFT ROT  / C = 1.0 \    | L: 1.0 |  [>>>]  | R: 1.0 |                           *
* 6 - RIGHT ROT / C = 0.4 \    | L: 0.4 |  [>>>]  | R: 0.4 |                           *
* 7 - RIGHT ROT / C = 1.0 \    | L: 1.0 |  [>>>]  | R: 1.0 |                           *
*****************************************************************************************
* ---COMMANDS---                                                                        *
* >> s [x]                                                                              *
* >> s cal                                                                              *
* >> add [type] [power]                                                                 *
* >> del                                                                                *
* >> del [x]                                                                            *
* >> delall                                                                             *
* >> lp [power]                                                                         *
* >> rp [power]                                                                         *
* >> showcal [type] [power]                                                             *
* >> trycal [type] [power]                                                              *
* >> run                                                                                *
* >> runcon //TODO                                                                     *
* >> stop                                                                               *
* >> generate                                                                           *
* >> order                                                                              *
* >> com                                                                                *
* >> quit                                                                               *
*****************************************************************************************
>>
```

Après la création de chaque étalon, nous les plaçons sur une courbe, en interpolant les données par des fonctions affines donnant un filtre de valeur des tensions bijectif (que nous transposons ensuite sur des valeurs entières de 0 à 255). Chaque valeur d'entrée du filtre ressort la tension réelle à envoyer dans chaque roue pour qu'elle corresponde à son étalon. présenté comme suit :



Avant chaque envoi de tension de moteurs par la communication Xbee, les vitesses demandées passent par ce filtre d'étalonnage afin de réaliser les mouvements souhaités.

5. Système décisionnel

Le robot doit être capable de longer les murs d'une salle sans les percuter en restant suffisamment proche pour cartographier ces murs. Il est nécessaire que le robot soit capable de réaliser cette tâche de façon autonome c'est-à-dire sans autre action d'un utilisateur que la mise en route de prise de décision.

Afin de permettre cette prise de décision, nous implémentons une machine à état pour définir en fonction des états précédents et des valeurs des capteurs la tension à fournir aux moteurs au moteur droit et gauche du moteur.

En effet, le robot possède à sa disposition un capteur à ultrason latéral avec une détection de plusieurs dizaines de centimètres ; On considérera trois niveaux de

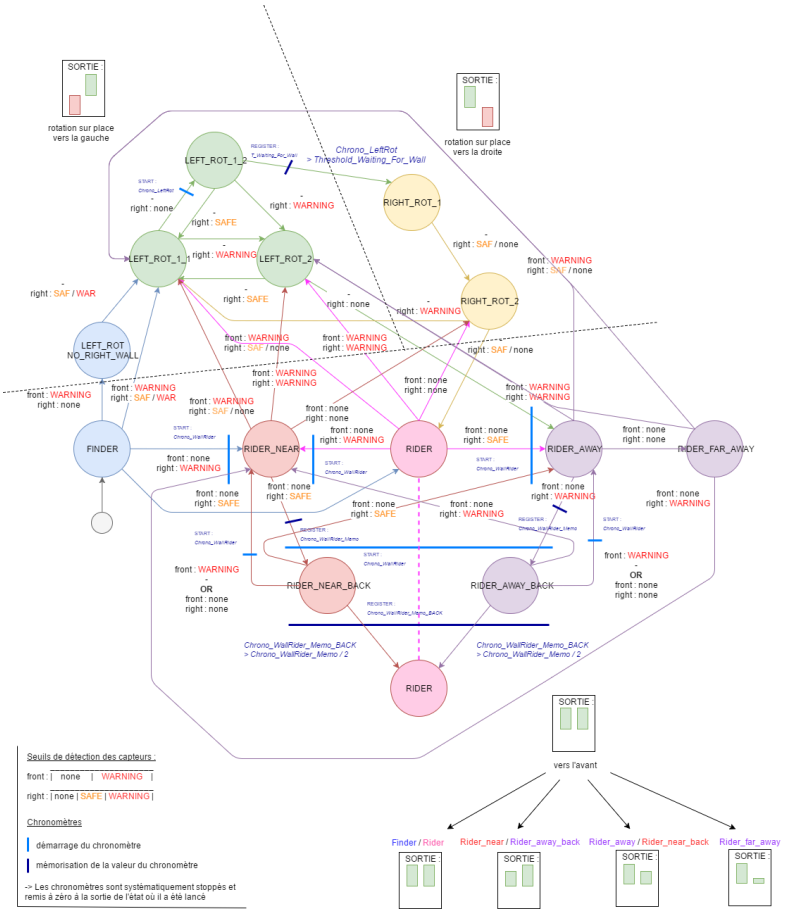
détection pour ce capteur. Le robot possède également un capteur frontal infrarouge qui balaye le devant du robot ; On prendra pour ce capteur 2 niveaux de détection.

Chaque état définit la tension à délivrer dans chacune des moteurs liés à chaque roue. Soient les tensions sont opposées, pour effectuer une rotation sur place, soient les tensions sont de la même parité pour avancer ou reculer (sachant que l'on ne tentera pas de reculer dans ce cadre de prise de décision.

Nous considérons 6 types d'états pour la machine et leurs sorties :

- Les états de départs (en bleu) le robot n'entre dans ces états qu'au démarrage de la prise de décision et n'y retourne pas. On avance droit jusqu'à rencontrer le premier mur.
- Les états de rotation vers la gauche (en vert) lorsqu'on rencontre un mur face à soi.
- Les états de rotation vers la droite (en jaune) lorsqu'on ne détecte plus de mur à droite ce qui signifie certainement qu'il y a un angle de mur ouvert où l'on se trouve.
- Les états de "ride" font avancer le robot droit. L'état RIDER fait avancer le robot tout droit mais cherche à savoir si le robot est plutôt proche du mur ou plutôt éloigné par considération de la distance du robot avec un seuil de détection entre plusieurs niveaux. Les états RIDE_AWAY et RIDER_NEAR font avancer le robot et font retourner le robot vers ce seuil (RIDE_AWAY : apporte une tension un peu plus grande à gauche, RIDE_NEAR à droite).

<u>Machine à état complète :</u>

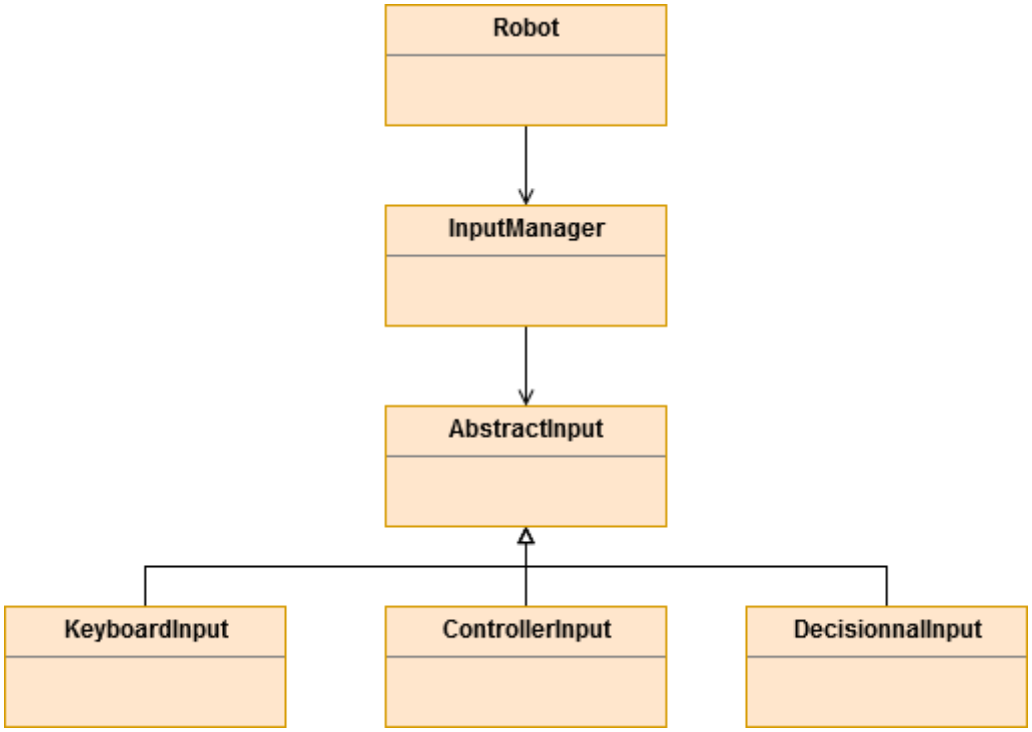


La machine à état du système décisionnel est disponible pour une meilleure lisibilité en *ANNEXE 3*.

Explication générique de la MAE en 5 points :

- Le Robot démarre dans l'état FINDER et part tout droit jusqu'à trouver un mur en face et passe alors dans l'état LEFT_ROT_NO_RIGHT_WALL puis LEFT_ROT_1 lorsqu'il repère le mur dans son capteur latéral. Il peut également passer dans l'état RIDER, si un mur est directement repéré à côté de lui. Ainsi le robot se place dans l'état faisant correspondre au mieux l'état de la Machine à état à sa configuration par rapport à son espace.
- Lorsque le robot entre dans l'état LEFT_ROT_1_1, il a détecté qu'il s'est trop rapproché d'un mur (soit parce qu'il y a un angle de mur face à lui ou un obstacle, soit parce qu'il n'est pas allé suffisamment droit lors des états de "ride" du mur). Pour différencier les cas d'angle de mur et de mauvaise trajectoire, il va commencer à tourner vers la gauche et démarrer un chrono lorsqu'il perd la détection du mur, s'il ne retrouve pas un mur c'est qu'il n'y avait pas de mur face à lui, il reprend donc son trajet en ligne droite en s'éloignant légèrement du mur (RIGHT_ROT, RIDER_AWAY) pour éviter un éventuel obstacle qui aurait pu déclencher cet état.
- RIGHT_ROT est état qui permet au robot de retrouver le mur lorsque son capteur latéral s'en est éloigné après les passages dans les états LEFT_ROT_1.
- LEFT_ROT_2 est un état de rotation qui oblige le robot à se préparer à s'éloigner du mur lorsqu'il est arrivé beaucoup trop près.
- Lorsque le robot "ride" un mur, son but est d'avoir une trajectoire parallèle au mur qu'il suit sur sa droite. il passe donc régulièrement dans les états RIDER, qui le fait aller tout droit, RIDER_AWAY qui le rapproche lentement du mur, et RIDER_NEAR qu'il l'en rapproche. Lorsque le robot sort d'un état RIDER_AWAY ou RIDER_NEAR, il entre dans la version "back" de ces états, c'est à dire, des états que le robot va garder la moitié du temps où il était dans la version classique de ces mêmes états avec des vitesses pour les roues droite et gauche inversées. Pour retrouver une trajectoire au mieux parallèle avec le mur avec de retourner dans l'état RIDER. L'état RIDER_FAR_AWAY se lance lorsque le robot s'éloigne trop du mur de droite ; Le déclenchement de cet état est semblable à celui de RIGHT_ROT et son but est le même (rapprocher le robot du mur) mais afin d'éviter les instabilités d'états il ne fait pas que simplement tourner le robot sur place mais il le faut aussi légèrement avance pour limiter le nombre de changement d'état lors d'une rotation autour d'un angle de mur ouvert sur la salle. Chacun de ces état de "ride" teste à tous moments si un mur se présente en face d'eux pour démarrer une rotation vers la gauche.

6. Système de pilotage



Le système de pilotage doit être une solution parallèle à la prise de décision du robot, nous avons donc généralisé la gestion des actionneurs du robot.

a) Clavier

Nous avons donc choisi le pilotage par clavier car cela permet de s'abstenir de regarder l'écran pour piloter le robot dans la pièce. Cette solution est d'ailleurs plus confortable qu'un contrôle à la souris.

b) Manette

D'autre part, nous avons voulu ajouté la possibilité de diriger le robot par une manette quelconque (Xbox, PS4, PS3…) car les joysticks permettent de jauger la vitesse de déplacement et de rotation de manière plus précise. C'est d'ailleurs une solution couramment utilisée dans le pilotage de robots comme les drones.

7. Simulation physique

Afin de garantir un affichage proche de la réalité, il est important de simuler au mieux le comportement du robot dans son environnement. Nous supposons donc un sol plat, sans perturbations et unique : celui du bâtiment de l'AIP.

Nous pouvons alors définir une échelle permettant de relier l'environnement logiciel avec la réalité : 1cm = 3px.

Il est également important d'établir des relations mathématiques entre la puissance appliquée sur chaque moteur et le déplacement (ou la rotation) du robot.

8. Interface graphique

9. Affichage de l'état du robot

10. Affichage de la puissance des moteurs

11. Gestion de l'état du robot

12. Verrou / sécurité du robot (Homme mort)

13. Mapping de l'environnement

V. Intégration

VI. Validation

Thibaut

Pour valider les différentes fonctions du robot, c'est à dire que ce soit sur le déplacement ou les résultats retourné par le capteur, il a fallu tester dans un premier temps chaque partie séparément.

Pour le déplacement, une fois la conception terminée, le teste c'est déroulé tout d'abord en partant de la position arrêt puis vérifier que chaque mode de déplacement était atteignable depuis celui-ci. La même procédure a été appliqué pour chaque mode de déplacement évoqué dans la partie robot mobile.

Pour les capteurs, il a fallu contrôler que chacun d'entre eux renvoient bien des informations, puis pour l'ultrason que les données reçues étaient valide en terme de distance. Concernant l'infrarouge, un souci a été décelé sur le capteur, il a fallu en prendre un autre mais aucune formule nous permettait d'estimer une distance cohérente. On a donc décidé d'interpréter directement la tension retournée par le capteur, ainsi on a pu établir un schéma d'utilisation de celui-ci.

En suivant cette procédure on a pu montrer que le robot fonctionnait selon nos désirs.

Brice

Les principaux tests de la machine à état ont été effectués sur le programme de simulation. Ce programme permet de tester la machine à état sur différents types d'obstacles, différents seuils de détection et différentes vitesses de sortie.

VII. Organisation

1. Temporelle

Pour l'organisation temporelle du projet nous avons décidé de la diviser en trois phases parallèles.

La première phase consistait à réaliser le montage du robot, c'est à dire, choisir les types de capteurs, le type d'alimentation pour l'Arduino et les moteurs, la mise en place d'un servomoteur ou non, ce qui a permis de définir la disposition de chaque élément sur le robot, afin que celui-ci soit le plus fonctionnel possible. Mais aussi la réalisation du moteur physique pour pouvoir simuler le robot dans un environnement quelconque. Et enfin la calibration, qui permettra de rééquilibrer la vitesse des moteurs au cours du temps.

La deuxième phase consistait à mettre en place la prise de décision en fonction des différentes valeurs que peut retourner le robot, c'est à dire, les différentes manœuvres d'évitement mais aussi le suivi de mur. Puis la conception de la bibliothèque que l'Arduino devra utiliser afin de faire fonctionner le robot. L'IHM, permettant à l'utilisateur d'interagir avec le système du robot. Et enfin, la communication, phase importante du projet car elle permet de faire transiter toutes les informations du robot vers le PC et inversement.

La dernière phase a été assigné à l'intégration de chacune des parties et à la réalisation du rapport.

2. Physique

En ce qui concerne l'organisation physique, nous avons pu utiliser les locaux de l'AIP afin d'utiliser tout le matériel mis à notre disposition pour concevoir le robot. Nous avions aussi utilisé nos machines personnelles ainsi que celle présente dans les salles afin de concevoir la partie logicielle. Par ailleurs, une utilisation cross-Platform a été faite car la conception a été réaliser sur trois systèmes d'exploitation diffèrent, Linux, OS X et Windows.

3. Outils

Pour ce qui est des outils que nous avons utilisés afin de réaliser ce projet, ceci peut être divisé en trois parties.

La première partie, communication. Nous avions utilisé différentes plateformes. Principalement Messenger, facile d'utilisation, il nous a permis de maintenir une communication stable de nos avons respective. Discord, utilisé pour communiquer de vive voix et ainsi être plus directe dans nos échanges.

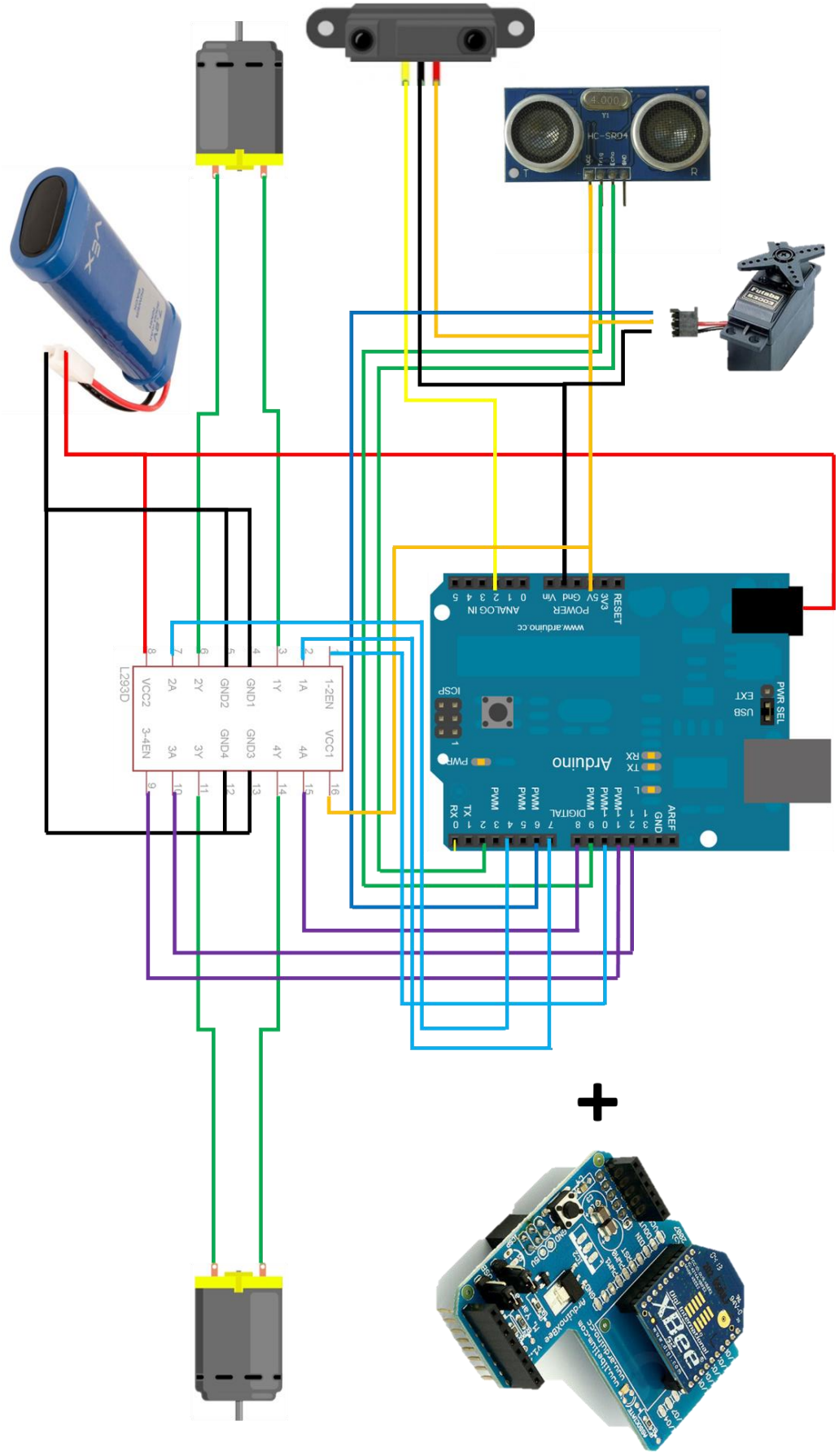
La deuxième partie, Maintenabilité du code et des documents divers. Pour les documents, l'utilisation de Google Drive c'est trouvé fort utile pour mettre en commun la réalisation écrite de chacune de nos parties. En ce qui concerne le code, GitHub reste l'un des meilleurs outils pour la gestion de version et le maintien du code au sein d'une équipe.

Et la dernière partie, les outils de conception. Pour la partie développement du moteur physique, calibration et IHM, celle-ci ont été codé en Java via l'IDE Eclipse et l'utilisation de la bibliothèque LIBGDX. Côté bibliothèque Arduino du robot, celle-ci a été développé en C++ en utilisant Sublime TEXT pour la saisie et L'IDE Arduino pour la compilation.

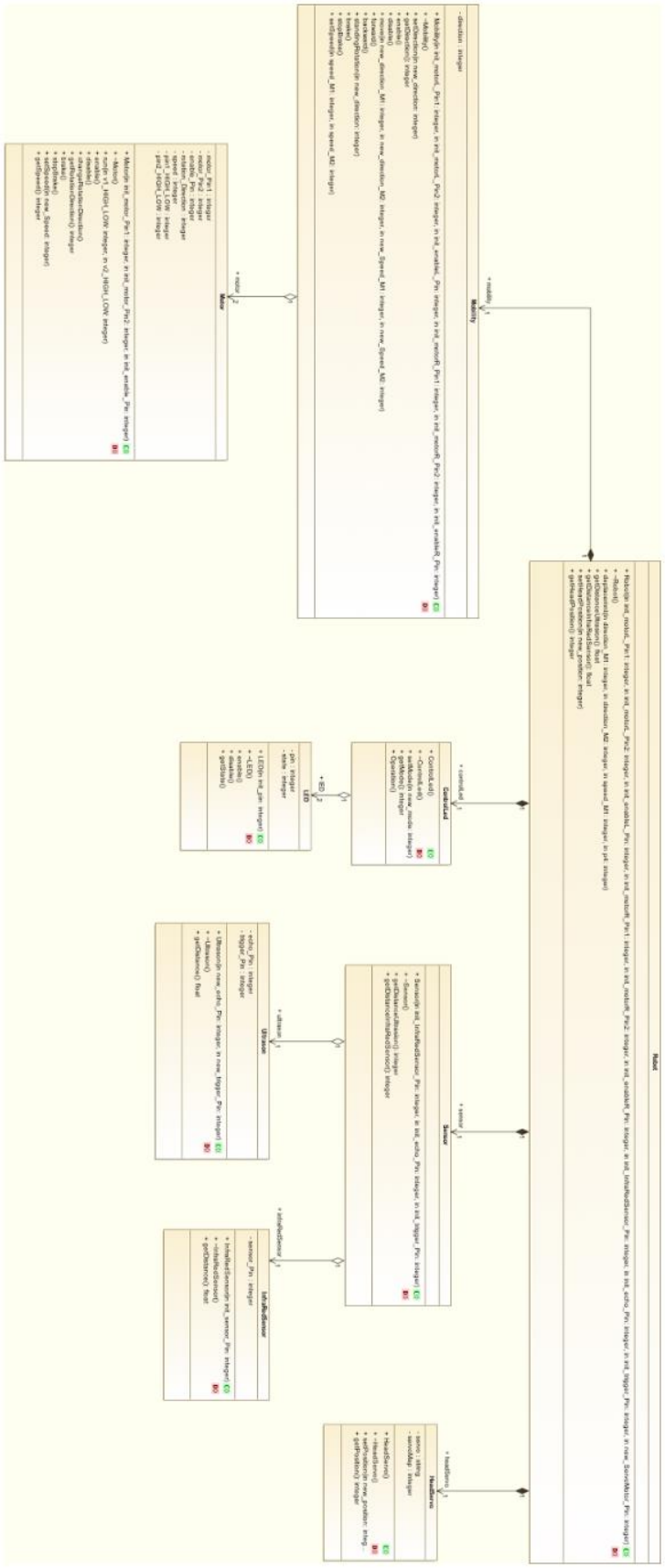
Pour ce qui est de la partie communication, côté PC, elle a été codé en Java via Processing et pour la partie Arduino, en C/C++.

VIII. Bilans

ANNEXE 1 : Schéma de câblage



ANNEXE 2 : Diagramme des classes participantes embarqué sur le robot mobile



ANNEXE 3 : Machine à état du système décisionnel