*Carval Amaury*
*Preud'homme Geoffrey*

*Polytech Lille - 4ème année*
*Informatique, Microélectronique, Automatique*
*2017-2018*

# Rapport de projet P66
# « Coupe de France de robotique »


Responsables :

- Mr Redon Xavier
- Mr Boé Alexandre
- Mr Vantroys Thomas

en partenariat avec Robotech, club de robotique de Polytech Lille

## Remerciements

Nous adressons nos remerciements à Planètes sciences et Oryon, qui proposent chaque année un défi et un événement ludique permettant de mettre en œuvre nos connaissances, ainsi qu'aux autres équipes participant à la coupe pour leur fair-play et leurs conseils.

Nous souhaitons ensuite remercier Robotech. D'une part ses anciens membres qui ont permis de construire un héritage non négligeable nous ayant permis de financer la Coupe de France de robotique 2018, et par extension ce projet. D'autre part ses membres actuels, nous ayant pour certains conseillé dans certains choix et pour d'autres accompagné tout au long de la préparation de la coupe. On pense notamment à Stephen Andriambolisoa, Gaëlle Bernard, Mathis Dupre, Brandon Elemva, Pierre Frison, Anas Ilou, Juliette Obled et Valentin Pitre.

Remercions également le Fabricarium, pour nous prêter ses locaux, ses machines et nous fournir du matériel ; ainsi que l'entreprise Bulles 2 Services pour avoir été partenaire de Robotech pour cette coupe, notamment Nathalie Preud'homme pour avoir été notre coach et conductrice.

Nous tenons également à remercier les professeurs qui nous ont conseillé, même rapidement au détour d'un couloir, notamment Rochdi Merzouki, Emmanuelle Pichonat, et Rodolphe Astori. Un remerciement spécial à Thierry Flamen, pour la réalisation des PCB mais surtout pour ses conseils et sa patience.

Enfin, nous remercions tout particulièrement messieurs Alexandre Boé, Xavier Redon et Thomas Vantroys pour nous avoir accompagné durant ce projet mais aussi pour nous avoir permis de réaliser une partie cet évènement en tant que projet école, nous permettant de le traiter avec beaucoup plus de professionnalisme et de profondeur qu'un simple projet associatif.

## Sommaire

# 1. Introduction

Lors de ce projet, notre objectif a été de répondre au besoin émis par le club Robotech Lille d'obtenir un robot offrant à la fois un déplacement aussi fiable que possible et une électronique embarquée modulable afin de subvenir aux besoins évoluants au fil des années. Le but final est de permettre au club de participer dans les meilleurs conditions à la Coupe de France de Robotique qui a lieu chaque année à la Roche-sur-Yon. Dans ce rapport nous présenterons donc le contexte et ses exigences pour ensuite développer le travail qui a été réalisé. Nous terminerons ce rapport par une critique du travail accompli et ses perspectives d'évolutions.

# 2. Présentation du projet et état de l'art

## *2.1. La compétition*

La coupe de france de Robotique est une compétition qui s'adresse à toute équipe d'étudiants souhaitant relever un défi ludique et scientifique de robotique. Il s'agit aussi du 1er rassemblement d'écoles d'ingénieurs d'Europe. Chacune des 200 équipes présentes cette année doit réaliser un robot autonome respectant un cahier des charges précis afin de participer.

Un match se déroule sur un terrain avec deux équipes qui doivent dans un délai de 100 secondes valider le plus d'actions proposées dans le cahier des charges de façon à accumuler des points. L'équipe qui a validé le plus de points remporte le match.

Chaque année un nouveau cahier des charges est donné aux environs du mois de novembre avec de nouvelles épreuves et de nouvelles spécifications. Cette année le thème est "Robot Cities, Build a Better World" et nécessite de réaliser les tâches suivantes :

- Récupération,  tri et propulsion de balles
- Empilement de cubes de couleurs
- Lancement d'un engin mécanique
- Appui sur un interrupteur pour allumer un panneau domotique

*Présentation du terrain sur lequel la compétition aura lieu*

### 2.2. Le client

Depuis maintenant 5 ans Robotech, club associatif de Polytech Lille regroupe une dizaine de personnes afin de participer à cette compétition. Il doit faire face aux changements de cahier des charges qui ajoutent à la contrainte d'un budget serré la difficulté de réutiliser le matériel d'une année à l'autre.

### 2.3. Le projet

### 2.4. Introduction

Notre projet ne sera pas de construire un robot complet pour la compétition mais de fournir à notre "client" une base mobile avec un asservissement permettant le meilleur compromis fiabilité/vitesse ainsi que le dimensionnement et la conception de toute l'électronique embarquée en mettant une priorité sur la taille qui doit être la plus petite possible.

Plus spécifiquement, nous devons réutiliser au maximum ce qui existe déjà dans le club en terme d'ordinateur de bord, de moteurs et autres composants tout en garantissant que les investissements réalisés se révéleront utiles pour l'avenir du club. De plus, les solutions doivent être compréhensible par les années à venir surtout pour l'aspect développement où une nécessité de modularité des différents éléments constitutifs du projet devra être réalisées.

### 2.5. Organisation du projet

Nous avons découpé le projet en de nombreuses tâches que nous nous sommes répartis en fonction de nos compétences respectives. La plupart des tâches nécessitent la réalisation de plusieurs tâches au préalables, l'ordre de réalisation des tâches a été fixé par cette contrainte.

Vis à vis du client nous avons participé à chaque réunion du club, les jeudis midis, afin de suivre et de discuter les besoins sur le robot et les changements pouvant impacter notre projet.

De plus en dehors des séances de projets nous avons régulièrement remis à plat les difficultés rencontrées individuellement et les solutions trouvées.

Le Wiki nous a également permis d'avoir un journal de bord sur les étapes validées et informations trouvées et nous a fait gagner beaucoup de temps en nous permettant de ne pas reproduire une expérience déjà réalisée ou de rechercher une information à nouveau.

En outre, nous avons utilisé d’autres outils pour gérer nos tâches de façon approfondie et prendre note des heures de travail passées de façon. Ces outils exportant directement les informations sur le Wiki.

## *2.6. Etat de l'art*

Robotech n'en est pas à sa première participation à la Coupe de France de robotique. Cependant, très peu de composants ont survécus aux années, et peu de code reste trouvable. Faisons état de ce qui nous est légué de la participation de l'année passée.

### 2.6.1. Estimation de la position et asservissement

Jusqu'à lors, l'estimation de la position était réalisée à l'aide des roues codeuses des moteurs dont nous disposons et d'un Arduino récupérant les signaux générés par ces codeuses. Cependant l'Arduino n'étant pas assez performant pour enregistrer tous les signaux, il était seulement possible d'enregistrer jusqu'à une certaine vitesse - assez faible pour l'utilisation.

De plus, seule une codeuse des deux moteurs était lue, il était considéré que les deux roues se comportaient de la même façon. Enfin, l'asservissement en position réalisé sur le même Arduino était constitué d'un PID sur chaque roue, ce qui fonctionnerait dans le cas où les deux roues atteignent leur consigne avec la même évolution, mais ce n'était pas le cas.

Le déplacement était en conséquence de ces choix techniques assez approximatif.

### 2.6.2. Conversions de tensions pour l'alimentation

Le stockage de l'énergie sur un système embarqué se fait à l'aide de batteries électrochimiques qui peuvent être à base de lithium (LiPo), Nickel hybride (NiMH), plomb.. L'avantage de ce type de stockage est qu'il permet le stockage d'une grande quantité d'énergie pour un faible volume et offre des capacités de courants très importantes.

Cependant elles ont le défaut d'avoir une tension de sortie variant selon la charge de façon non linéaire et non équivalente d'une batterie à une autre. C'est pour cela que la conception d'une carte de puissance devient nécessaire.

Il existe trois technologies permettant de réaliser cette conversion de tensions :

#### *2.6.2.1. Step up converter*

Ce type de conversion revient à obtenir d'une tension A une tension B supérieure à la précédente ce qui est impossible avec un système de régulation linéaire (comme une PWM). Deux solutions existent :

- Passer par un transformateur ce qui signifie convertir le courant en alternatif puis le redresser après passage dans un transformateur.
- Utiliser les propriétés des bobines qui vont stocker l'énergie et la restituer à une valeure plus élevée cependant la difficulté consistera à suffisamment bien filtrer la

sortie qui se révèlent être à haute fréquence. Cette méthode est plus complexe à mettre en oeuvre si on souhaite une bonne stabilité du système notamment à cause du filtrage haute fréquence.

Quelle que soit la méthode utilisée, une boucle de contre réaction est nécessaire afin de stabiliser le système et de permettre un tension de sortie stable quelque soit l'appel de courant que le système va devoir fournir. Sinon nous aurions un rapport direct entre la croissance de la charge et la décroissance de la tension de sortie.

Il existe aujourd'hui assez peu de convertisseur UP offrant à la fois une tension de 24V et un courant suffisant pour supporter une charge élevée.

#### *2.6.2.2. Step down converter*

Une conversion de tension de ce type est plus simple à mettre en oeuvre que la précédente, en effet ici le découpage est envisageable. On parle de convertisseur buck ou hacheur série et plus généralement on parle d'alimentation à découpage. Cette technologie très bien maîtrisée et répandue est bien moins chère que la conversion UP évoquée précédemment.

Le principe des alimentations à découpage (à opposer aux alimentations linéaires) est d'utiliser les composants en mode commutation afin de découper la tension, on obtient ainsi un signal créneau qu'il suffit de lisser pour obtenir notre tension de sortie.

#### *2.6.2.3. Fly-back et Buck-boost converter*

Ces deux modes de conversions sont assez proches l'un de l'autre dans leur fonctionnement général. Ce système permet de fournir une tension inférieure ou supérieure à l'entrée. Le principe est de fournir un découpage de la tension d'entrée comme dans un step down converter, mais cette fois de jouer également sur le potentiel négatif. De cette façon, la différence de potentiel peut-être plus grande que l'entrée mais de polarité inverse

La technologie Buck-Boost utilise pour cela une inductance alors que le flyback utilise un transformateur pour réaliser la conversion de tension.

## 3. Établissement du cahier des charges

Le cahier des charges pour le robot principal de la coupe de robotique est défini par notre client Robotech et s'articule succinctement autour de quatre points :

1. Faire un robot conforme au règlement de la Coupe de France de robotique (annexe : 9.2.1.1)
2. Faire en sorte que le robot marque un maximum de points selon le précédent règlement
3. Réutiliser à cette fin un maximum d'éléments déjà disponibles dans les réserves de Robotech
4. Si des achats doivent être effectués, choisir dans la limite de l'imaginable des composants pouvant être réutilisés pour les années futures

Notre projet ne porte pas sur l'intégralité de la conception de ce robot : en effet sa conception prendrait beaucoup trop de temps pour un projet d'IMA sur une année. Nous avons donc séparé la partie traitée en tant que projet de la partie traitée par Robotech, car cette dernière est réalisée avec des autres personnes et leur travail ne doit pas être pris en compte dans l'évaluation du présent projet.

Le cahier des charges du projet, prenant toutefois en compte certaines spécificités demandées par le client est le suivant :

Voici le cahier des charges auquel le projet répond dans sa forme finale :

1. **Se déplace en suivant un parcours programmable par le client.**
   Sur une piste de 2m×3m en PVC (type bâche adhésive décorative pour sol).
2. **Est capable de connaître sa propre position.**
   On s'autorise une dérive de quelques centimètres pour 5 m de déplacements. Ces 5 mètres correspondent à la distance que parcourra le robot durant un match.
3. **Évite les obstacles à l'avant et à l'arrière.**
   Le parcours à réaliser est statique, cependant une protection basique contre les imprévus (autres robots) est nécessaire.
4. **Résiste aux interférences causées par d'autres capteurs de distance de même type.**
   Il y aura d'autres robots présents sur la piste, et certains d'entre eux pourront utiliser les mêmes capteurs que nous utilisons. Il faudra donc faire en sorte que les interférences ne perturbent en rien le bon fonctionnement de notre robot.
5. **Est facilement debuggable.**
   Après la réalisation du robot dans le cadre du projet, il passera dans une longue phase de test pour s'assurer du bon fonctionnement des actions qu'il est censé

réaliser. Afin de faciliter et accélerer cette phase, on aimerait plusieurs choses du système de contrôle du robot. Il devra démarrer rapidement, utiliser le minimum de ressources nécessaires, et pouvoir être mis à jour et débuggé via une communication sans fil.

6. **Peut interagir avec l'utilisateur.**
   Une fois sur le chemin de compétition, le robot n'aura plus l'aide d'un ordinateur pour pouvoir communiquer avec l'utilisateur. L'utilisateur doit toutefois pouvoir effectuer des tâches simples (saisir la zone de départ, l'équipe, la stratégie...) et contrôler l'état du robot (batterie, calibrage...).
7. **Est conforme au cahier des charges de la Coupe de France de Robotique 2018 en tant que robot principal.**
   Il fixe entre autres des conditions sur les dimensions du robot et sur le type de matériel qu'il peut embarquer.
8. **Réutilise le matériel disponible à Robotech.**
   La création depuis zéro d'un tel robot est largement hors-budget pour un projet IMA4. On utilisera au maximum le matériel mis à disposition par le client pour minimiser les dépenses.
9. **Dispose une alimentation 9 V.**
   C'est une demande spécifique du client.

Ce cahier des charges est identique à celui écrit en début de projet, à l'exception du point numéro 2. En effet la précision initialement voulue (1 mm) est en fait assez irréaliste pour le matériel dont nous disposons.

# 4. Choix des technologies

## *4.1. La base mécanique du robot*

Elle est déjà existante, et possède les caractéristiques suivantes :

- Une structure en aluminium offrant un bon compromis solidité / légèreté
- Deux moteurs Faulauber de tension nominale 24V pour 5500 tr/min et une puissance de sortie de 25.4W
- Un système de réduction réalisé de rapport 1:4 c'est à dire que pour une révolution du moteur pour une rotation des roues de 90°. Cela permet une augmentation du couple disponible en échange d'une vitesse maximale réduite, ce qui ne nous importe peu dans notre cas.
- Des codeuses HEDS-5500 directement branchées sur chaque moteur

## *4.2. Les capteurs et les actionneurs*

Le robot doit savoir faire toutes sortes de tâches de part ses actionneurs, et leurs actions sont dictées par des contrôleurs, qui prennent connaissance du monde réel par des capteurs. Les actionneurs et les capteurs sont déterminés par les actions que doit faire le robot et ce qu'il doit savoir. Le modèle exact de ces composants est déterminé par le stock dont dispose le client. L'établissement d'une liste de composants est donc très rapide :

### 4.2.1. Actionneurs

- Moteurs (par l'intermédiaire d'un contrôleur moteur, cf ci-dessus)
- Ecran LCD (écran HD44780 + contrôleur I²C PCF8574) : pour afficher les informations

Le client nous informe qu'il aura d'autres actionneurs embarqués sur le robot, et que ces derniers seront tous gérés par un contrôleur Arduino (afin de pouvoir travailler sur des bases de codes séparées). À titre informatif, les actionneurs sont les suivants : servo-moteur d'ouverture de loquet pour récupérer les balles, moteur dynamixel AX12A utilisé pour faire tourner le barillet stockant les balles, servo-moteur pour pousser les balles hors du barillet, servo-moteur pour sélectionner si la balle doit être évacuée ou propulsée, et relai permettant d'activer/désactiver le moteur autorisant la propulsion des balles.

### 4.2.2. Capteurs

- Capteurs de distance (HCSR04) : afin d'éviter les obstacles
- Encodeurs (HEDS-5500 C06) : afin de connaître à tout moment la rotation des roues du robot
- Boutons : Afin de fournir au robot quelques informations

- IMU / Centrale inertielle (MinIMU-9 v3) : moyen alternatif de connaître la position du robot

### 4.3. Les contrôleurs

Le choix des contrôleurs sera dicté dans un premier temps par leur capacité à contrôler ces capteurs et actionneurs. Nous disposons de 3 contrôleurs différents, établissons un tableau de compatibilité.

| Contrôleur | FPGA MicroNova Mercury | Arduino Mega | Raspberry Pi 3 B+ |
|---|---|---|---|
| Contrôleur moteur | PWM 3.3 V possible | PWM 5 V possible | PWM 3.3 V possible, fréquence fixée |
| Écran LCD | Module I²C disponible | Communication software I²C limitée | Communication I²C native |
| Capteur de distance | Parfait | Fonctionne très bien avec interruptions | Trop lent pour avoir des valeurs fiables |
| Encodeurs | Parfait | Vitesse limite imposée. Sens indisponible. | Trop lent pour avoir des valeurs fiables |
| Boutons | Parfait | Parfait | Parfait |
| IMU | Module I²C disponible | Communication software I²C limitée | Communication I²C native |

Le FPGA pourrait suffire à lui tout seul. Cependant, la programmation sur FPGA est assez peu flexible ce qui ne convient pas au client. De plus, embarquer toute la logique de contrôle, position et enregistrement des valeurs n'est peut-être pas possible avec la "mémoire" limitée du FPGA.

On utilisera alors en complément son parfait opposé, le Raspberry Pi, qui le déchargera des tâches qu'il est capable de faire aisément. Il est d'ailleurs bien plus adapté pour les calculs de position et d'asservissement que l'Arduino de part son processeur plus puissant. Il permettra aussi de simplifier la procédure de débogage du programme demandée par le client, car il est très simple de s'y connecter avec des protocoles standards (Wi-Fi + SSH). On pourra d'ailleurs l'utiliser pour mettre à jour le programme du Raspberry Pi et celui du FPGA.

On utilisera une liaison série pour la communication entre le FPGA et le Raspberry Pi pour différentes raisons :

- simplicité d'implémentation sur le FPGA
- non congestion du bus I²C existant

- meilleures performances liées à une plus grande flexibilité du protocole (pas besoin d'adressage des composants ni des variables à lire)

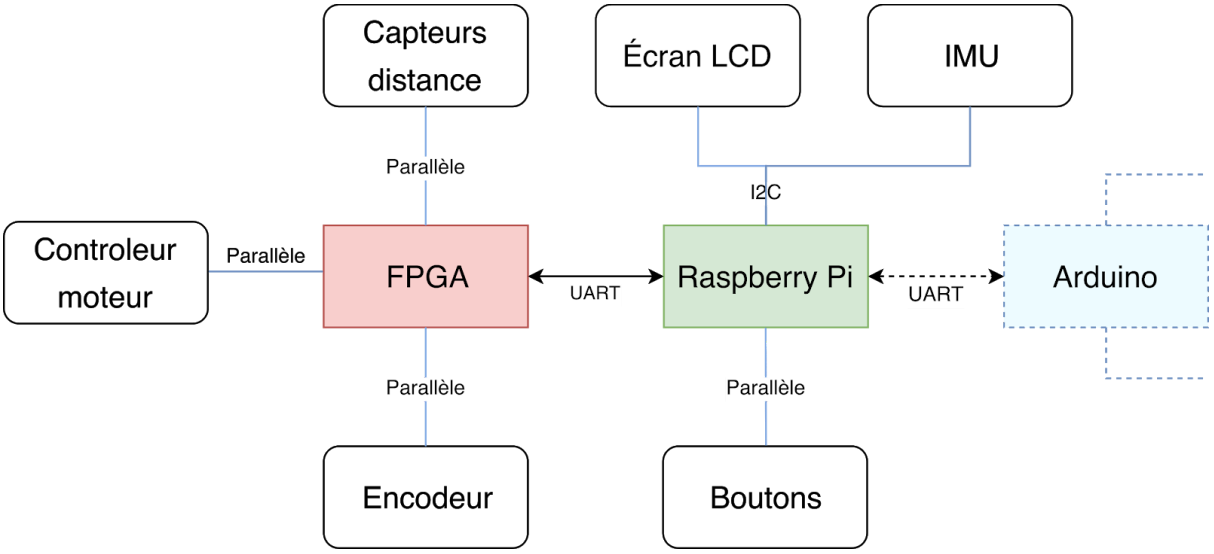



*Schéma fonctionnel des actionneurs, capteurs et contrôleurs utilisés par le robot ainsi que leur méthode de communication*

### 4.4. Contrôleur moteur

Pour le contrôle des moteurs nous utiliserons des composants LMD18200T qui embarquent des ponts en H complets permettant le contrôle à l'aide d'une PWM, d'une entrée sélectionnant la direction des moteurs et une entrée autorisant le passage en freinage (court-circuit du moteur). Afin de déterminer ce qu'il est aussi nécessaire d'utiliser, observons son schéma bloc :

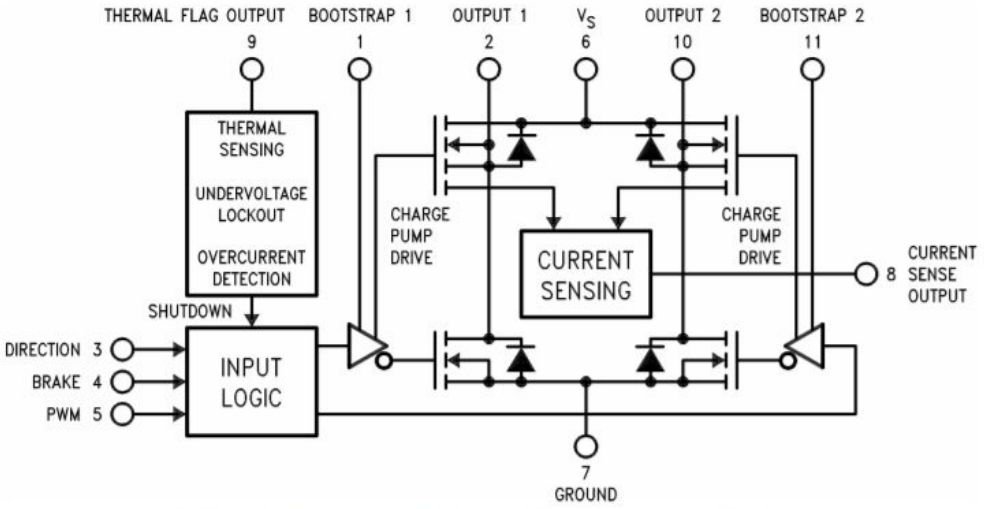



*Schéma bloc du LMD18200T*

Les sorties nommées **bootstrap** qui seront reliées à la sortie du pont en H déterminent la fréquence de fonctionnement du contrôleur moteur. L'entrée du contrôleur moteur est

une PWM, alternant entre les valeurs 0 V (GND) et 5 V (VCC) avec une proportion variable fournissant une information assimilable à une tension continue analogique.

Afin de ne pas avoir un effet de créneau sur notre moteur, il faut placer entre le bootstrap et la sortie une capacité pour lisser le signal. Cette capacité va influencer sur la fréquence maximum de notre PWM. Nous utiliserons la capacité conseillée par le corollaire, à savoir 10 nF. Théoriquement, cela offre un temps de montée de 100 ns soit une fréquence de PWM d'un minimum de 500 kHz[1].

## 4.5. Électronique de puissance

### 4.5.1. Bilan des consommations

L'électronique de puissance a été dictée par les besoins nécessaires pour notre projet et par le reste des besoins du client. Ces derniers ont d'ailleurs été assez difficile à obtenir, certains paramètres ayant été fixés à la dernière minute. Voici un récapitulatif des éléments consommateurs constituant le robot.

| Consommateur | Alimentation choisie | Courant maximum | Quantité |
|---|---|---|---|
| Arduino Mega | 9 V | 0,5 A | 1 |
| Raspberry Pi 3 B | 5 V | 0,82 A | 1 |
| Moteur Faulhaber | 24 V | 2,5 A | 2 |
| FPGA Mercury | 5 V | 0,275 A | 1 |
| Dynamixel AX-12A | 9 V | 0,2 A | 1 |
| Autres actionneurs | 5 V, 10 V | 0,6 A | N/A |

Au total nous obtenons donc un maximum de courant consommé théorique de 7,575 A. Cependant, en test de stress, nous n'avons pas nécessité un courant dépassant les 3 A. Il faudra donc convertir la tension fournie par la batterie (imposée) LiPo de 4 cellules, fournissant une tension variant autour de 14,4 à 16,8 V en fonction de sa charge vers les différentes tensions décrite ci-dessus en respectant les courants maximums.

L'impact que ces éléments ont eu sur notre projet réside dans la nécessité à penser la conception des cartes et surtout leurs formes. De plus, la disposition dans le robot des différents éléments que nous ne maîtrisons pas dans le cadre du projet implique des

[1] C'est d'ailleurs la raison pour laquelle la génération des PWM n'est pas faite sur Raspberry Pi mais sur FPGA : la fréquence des PWM est fixée mais n'est pas adaptée à la configuration de notre composant.

rendez-vous avec les designers des tâches pour la compétition afin de vérifier que les solutions trouvées soient mutuellement compatibles avec notre design.

### 4.5.2. Choix des technologies

Pour répondre à toute cette demande cela nous utiliserons deux technologies :

- **Down-converter** (également appelé buck converter) à tension réglable. Ils sont relativement peu chers et offrent une modularité en permettant de facilement changer la tension de sortie en modifiant la valeur de la résistance.
- **Convertisseurs CC-CC** isolés permettant de fournir une tension de 24V.

### 4.5.3. Choix des composants

#### *4.5.3.1. Sortie 10 V et 5 V*

Nous avons choisi d'utiliser composant down-converter PTN78020W car il permet, à l'aide d'une simple résistance, de choisir la tension de sortie, tout en offrant un courant disponible de 6A - ce qui est largement suffisant pour cette année et laisse de bonnes perspectives pour les années futures.

Il existe de nombreuses alimentations à découpage sur le marché, notre choix à été déterminé par sa taille très compacte, la simplicité de réglage de la tension et son adaptabilité. Le tableau utilisé pour calculer la valeur de cette résistance est disponible en annexe 9.2.1.3. Pour notre usage, nous utiliserons une résistance de 22kΩ pour le 5V et une de 3,2kΩ pour le 9V.

#### *4.5.3.2. Sortie 24 V*

Le marché nous a imposé l'utilisation du composant UWE-24/3-Q12N-C.. En effet il a été très difficile de trouver un convertisseur fournissant la tension requise avec un courant acceptable pour des tensions d'entrée inférieures à la tension de sortie.

Le coût onéreux de ce composant nous a fait réfléchir sur l'utilisation d'une solution alternative : l'utilisation de deux batteries en série générant ainsi environ 30 V et permettant d'utiliser un buck converter. C'est une solution plus économique, modulable dans le temps, offrant un meilleur courant. Le défaut de cette technique réside dans l'encombrement : en effet deux batteries prennent beaucoup plus de volume. De plus des tensions plus élevées entraînent une perte d'efficacité des composants de puissances donc un effet joule plus important - impliquant une baisse d'autonomie, et plus grave un échauffement des composants qui est toujours à éviter quand on le peut tant pour le composant lui même que pour son environnement.

Il a été toutefois été choisi d'acheter le composant. En effet avec le client le constat a été fait que le volume de deux batteries était trop contraignant voire impossible pour la

réalisation du robot actuel. De plus, une seule batterie fournit un courant suffisant même lors des pics de consommation du moteur.

#### 4.5.4. Alimentation des contrôleur moteur

Le robot aura besoin de deux tensions continues de 24 V les plus stables possibles permettant de fournir la puissance à chacun des moteurs. Cette tension se révélera surdimensionnée par rapport à l'usage actuel du robot, Cependant il permet d'assurer une utilisation pérenne de notre carte dans les années à venir (par exemple, l'utilisation d'un réducteur avec un ratio dans le but de fournir plus de couple, ou l'augmentation du poids du robot).

### *4.6. Programme du Raspberry Pi*

#### 4.6.1. Système d'exploitation

Nous avons choisi d'utiliser un système d'exploitation minimal, afin de réduire l'empreinte mémoire et l'utilisation du CPU par ce dernier au minimum, afin de maximiser les performances pour le calcul de la position et de l'asservissement. On utilisera pour cela Buildroot, qui est un utilitaire qui nous permet de choisir quels programme installer et génère une image du système d'exploitation prêt à être gravée sur carte SD. Le système est reproductible, on peut donc sans problème l'effacer. Il est d'ailleurs très léger, pas besoin d'une carte SD de grande capacité.

Ce programme nous permet également de rajouter nos propres programmes, et gère lui-même la compilation croisée, permettant de compiler le programme pour une cible ARM sur un ordinateur avec une autre architecture, permettant ainsi de diminuer les temps de compilation.

Ces caractéristiques auraient aussi été disponibles avec un système d'exploitation temps-réel tel que FreeRTOS, mais nous préférons garder un minimum de compatibilité avec différents protocoles (Ethernet, Wi-Fi, SSH, USB…) afin de faciliter le débogage.

#### 4.6.2. Langage de programmation

Le Raspberry Pi supporte la quasi-totalité des langages de programmation, cependant, nous nous tournerons vers un langage compilé classique : le langage C. Les performances d'un tel langage sont bien meilleures qu'un langage compilé et permettent un meilleur asservissement du robot.

Pour obtenir les avantages du langage tout en minimisant les inconvénients vis à vis d'un langage interprété dynamique, on prendra soin de bien séparer les différents modules qui seront contrôlés par ce programme dans des fichiers différents avec des méthodes standardisées afin de pouvoir rapidement créer des programmes de test et les exécuter.

Outre les bibliothèques standard, on utilisera la bibliothèque **wiringPi**, offrant un accès simplifié au GPIO du Raspberry Pi, ainsi qu'à sa liaison I²C.

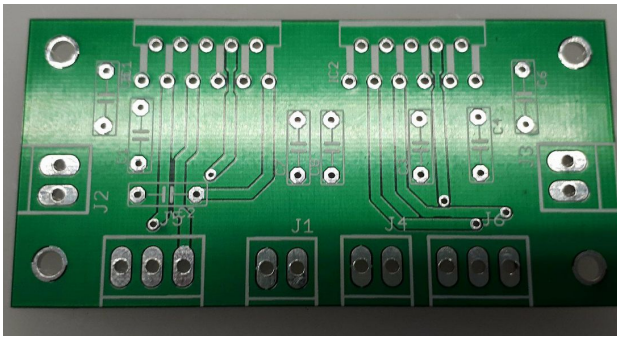
# 5. Réalisation technique

## *5.1. Routage des cartes*

Nous sommes partis du constat que les 2 principaux défauts des précédentes cartes conçues par le client étaient :

- Une durée de vie assez courte.
- Une forte tendance au court-circuit. Ils peuvent en effet arriver très facilement dans un espace aussi dense qu'un robot comme le nôtre.

L'usinage à Polytech se faisant sur des plaques de cuivre sans vernis celui-ci s'oxyde dans le temps et la qualité du circuit s'en ressent. Ce phénomène a surtout été observé sur les contrôleurs moteurs, plus sensibles que les cartes de puissances. C'est pourquoi il a été décidé de faire usiner les contrôleurs moteurs à l'extérieur afin d'avoir une carte de qualité professionnelle.C'est ce que nous avons fait pour la carte du LMD18200T. Sur cette image on voit le PCB livré avec une couche de vernis[2]




*Carte nue du contrôleur moteur sortie directement de l'usine.*

Pour le design, nous avons également prêté attention aux détails notamment en ce qui concerne la CEM (Compatibilité Électro-Magnétique) afin de réduire la sensibilité aux ondes extérieures, et de réduire le rayonnement d'ondes par le circuit lui-même. Ce n'est pas vital pour nos besoins, mais préférables. De plus les bonnes pratiques de CEM permettent une meilleure gestion du flux des électrons ce qui réduit les risques d'échauffements des pistes et donc des pertes joules.

Pour résumer le processus de la création d'un circuit imprimé est le suivant :

1. Création du circuit sur feuille avec dimensionnement des composants
2. Intégration du schematic dans le logiciel de conception

[2] Le pré-étamage des pistes offre également une qualité de soudure et donc une résistance aux chocs et aux vibrations.

3. Placement des composants en bonne intelligence entre leurs caractéristiques et la fonctionnalité du rendu finale.
4. Test du design rule check qui vérifie que le circuit respecte les spécifications de la machine qui usinera le circuit
5. Impression sur papier du circuit imprimé pour vérifier la taille des trous, visualiser les erreurs de type composants miroirs, la praticité du circuit etc.
6. Usinage final du PCB

Le respect de ces consignes permet d'éviter les erreurs nécessitant un nouvel usinage et assure la durabilité des PCB.

## *5.2. Branchements*

### 5.2.1. Organisation du câblage

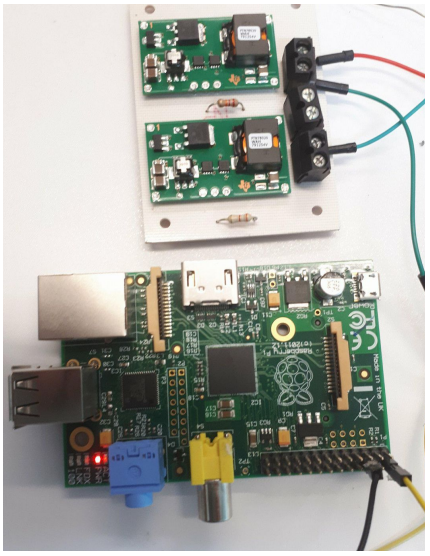
Le nombre de composants importants dans un espace restreint pose quelques contraintes pour ce qui est de relier tous les modules entre eux. Par souci de flexibilité, et afin de pouvoir travailler pour et avec le client, nous avons choisi de garder un câblage à base de borniers, dominos, et breadboard en dernier recours.

Ces derniers, bien que très pratiques lorsqu'il s'agit d'implémenter des modifications de dernière minute pour le client, ne sont pas d'une robustesse exemplaire, et il a pu arriver que certains fils s'arrachent des borniers ou de la breadboards, surtout durant les phases de transport.

Pour remédier à ce problème et sur conseil de nos encadrants nous avons essayé de déporter la contrainte mécanique jusqu'alors transmise directement sur les connectiques. Pour cela nous avons utilisé des gaines récupérées sur une alimentation de PC en panne afin d'isoler les câbles correspondant à des fonctions figés (codeuses, contrôleurs moteurs..) et nous avons utilisé des colliers de serrage pour bloquer les câbles sur les parois du robot. De cette façon les traction involontaire sur un câble sont absorbées par ces fixations.

De plus, un programme d'auto-diagnostic pour le Raspberry Pi a été écrit. Il est chargé de vérifier automatiquement un maximum de connectique en quelques secondes, et de reporter quels sont les liens qui ne fonctionnent pas correctement.

### 5.2.2. Alimentation des contrôleurs



*Utilisation d'un ancien Raspberry Pi pour vérifier le bon comportement des cartes de puissance 5 V.*

Dans un premier montage, nous avions choisi d'alimenter les broches de la breadboard directement depuis la carte de puissance, et de brancher le Raspberry Pi sur ce même lien. Cependant il s'avère que -même avec seul le Raspberry Pi de branché sur la breadboard- le comportement du contrôleur devient aléatoire, certains composants ne fonctionnent pas, et parfois le système refuse complètement de démarrer. Nous avons donc récupéré un câble micro-USB dont nous avons connecté le câble à l'alimentation 5 V, ce qui offre un comportement beaucoup plus stable.

Les autres contrôleurs, en revanche, ne souffrent pas de ce problème d'instabilité lorsqu'ils sont alimentés par leurs broches VCC et GND. Il faut cependant séparer la sortie de la carte de puissance en plusieurs points d'entrée proches des entrées des composants : en effet la breadboard possède une résistance interne non négligeable et la tension baisse rapidement en la parcourant, provoquant quelques instabilités aux niveau des contrôleurs. On remarque par ailleurs que les proches VCC et GND du Raspberry Pi ne peuvent être utilisées en tant qu'entrée en alimentation d'un autre contrôleur, celui-ci ne fournissant pas assez de courant.

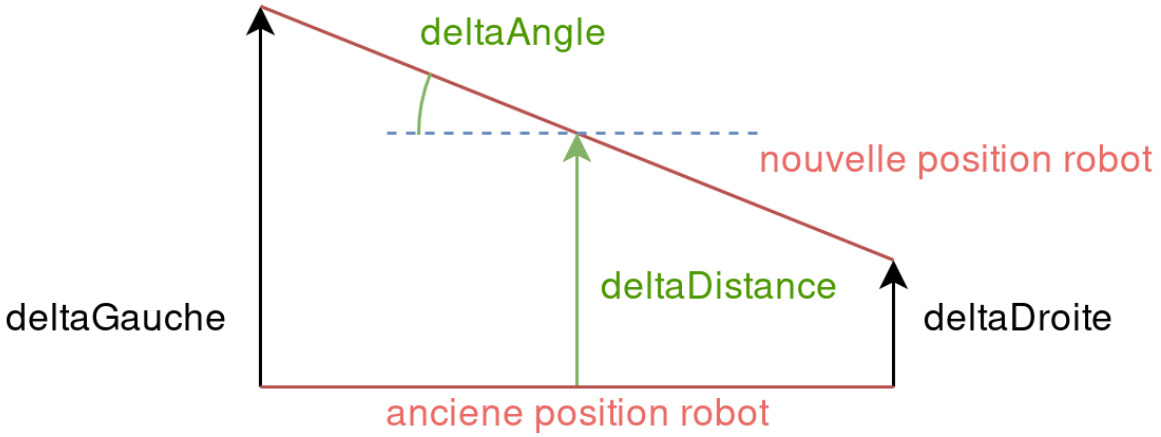
## *5.3. Position et déplacement*

### 5.3.1. Calcul de position

Les valeurs de rotation des deux encodeurs gauche et droite sont envoyés régulièrement depuis le FPGA au Raspberry Pi. Plus précisément c'est la différence de valeur depuis la dernière lecture qui est envoyée, car c'est de cette différence en rotation dont nous avons besoin[3] pour mettre à jour la position dans le système $(x, y, \theta)$. $x$ et $y$

[3] Il aurait été tout à fait possible d'envoyer une valeur absolue et de calculer le delta algorithmiquement (afin de prendre en compte les dépassements). C'est d'ailleurs ce qui

correspondent à la position du robot dans le système cartésien, $\theta$ (appelé **o** dans le code) est la rotation par rapport au vecteur directeur de l'axe $x$. Un simple calcul de trigonométrie est effectué pour calculer la nouvelle position.




*Représentation géométrique d'un changement de position du robot.*

### 5.3.2. Calcul de destination

Le robot se déplace en suivant un parcours prédéfinis par des points. Il s'y déplace en ligne droite, éventuellement en pivotant pour s'orienter dans la direction voulue (les roues n'étant pas omni-directionnelles). À l'arrivée d'un nouveau point, on peut découper le mouvement en trois parties :

1. Orientation vers le point de destination
2. Avancement vers le point de destination
3. Orientation vers l'angle de destination (si voulu par la consigne)

La consigne sera recalculée à chaque instant pour être un ensemble de deux valeurs : une distance à parcourir et un angle à pivoter, qui sont facilement convertibles en consigne de position sur les deux moteurs. Lors de la réception d'une nouvelle consigne, on fixe d'abord la distance à parcourir à 0, et on met à jour dynamiquement l'angle à pivoter de manière à ce que le robot soit face (ou dos, il peut aussi aller en arrière) au point de destination. Une fois orienté, la distance à parcourir est mise à jour pour suivre l'écartement entre la position actuelle et le point de destination. L'angle à pivoter est toujours mis à jour, afin de compenser les éventuelles différences entre les roues et imprécisions lors de l'orientation initiale. Une fois arrivée sur la position, et si la consigne le demande, on met à jour l'angle à pivoter pour suivre l'orientation de destination. Une fois cela fait, le robot est à destination.

---

a été fait lorsque nous avons remplacé la liaison série par une liaison I²C : la détection d'erreurs de transmission est alors plus facile.

#### 5.3.3. Asservissement

L'algorithme de déplacement serait aussi simple que celui évoqué ci-dessus si le système réel n'avait aucune inertie. En réalité, il est nécessaire d'utiliser un asservissement pour atteindre la consigne sans pour autant la dépasser ou se retrouver en état d'instabilité. On utilise pour cela un régulateur PID (proportionnel, intégral, dérivé) sur les deux valeurs de consigne : la distance à parcourir et l'angle à pivoter.

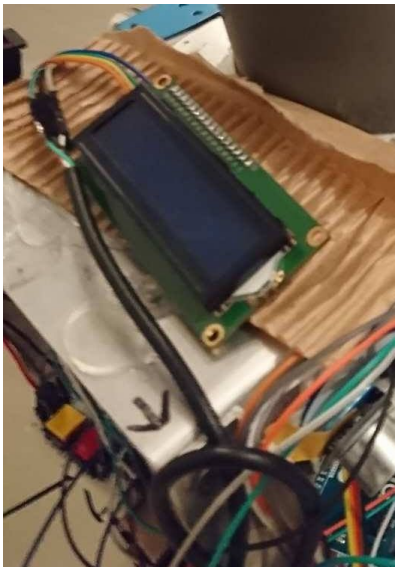
La régulation peut engendrer des dépassements de consigne. Il faut pour gérer ce cas autoriser que le robot puisse reculer (et non faire demi-tour pour revenir au point voulu comme dans les premières versions). Il est aussi nécessaire d'attendre que le robot se soit stabilisé autour de la consigne voulue (qu'elle soit en direction et/ou en angle) avant de freiner ou de passer à la consigne suivante pour éviter les glissements. Pour cela, on considère que si la vitesse du robot est nulle, et que la tension qui lui est fournie sur les roues est en dessous d'un certain seuil en dessous lequel le robot ne peut démarrer, faute de couple, on peut passer à la consigne suivante.

La régulation en position pose cependant quelques problèmes. En effet, à gain constant une petite consigne provoquera une petite accélération, mais une grande consigne provoquera une grande accélération. Pour palier à ce problème, il est conseillé de générer un signal rampe, afin d'obtenir une accélération identique indépendante de la consigne, et arriver à une vitesse constante (il s'agit donc de recréer un asservissement en vitesse). Une alternative créative tout aussi efficace consiste à limiter la consigne à un rayon autour de la position actuelle afin d'éviter des accélérations trop hautes et limiter la vitesse facilement.

### *5.4. Interaction avec les humains*

Le robot est à destination du client, et il aura besoin de modifier le parcours du robot facilement. Pour cela, plusieurs choses ont été mises en place.

### 5.4.1. Exécution des programme : l'interface homme-machine embarquée



*Écran LCD et boutons*

Le robot, à son allumage, lance automatiquement un programme permettant de faire différentes choses sans avoir à se connecter depuis un ordinateur. Afin de pouvoir régler quelques paramètres facilement, un afficheur LCD connecté par une liaison $I^2C$ accompagné de deux boutons montés en pull-up permettent de sélectionner entre plusieurs actions (réinitialiser la position, lancer un auto-diagnostic, lancer le programme) et modifier plusieurs paramètres (position de départ, programme à lancer, mode debug).

### 5.4.2. Modification des programme : la connexion au système du robot

#### *5.4.2.1. Établissement de la connexion*

Le Raspberry Pi est l'élément central du robot. Il est possible de s'y connecter comme l'on se connecte à un ordinateur distant. On peut utiliser pour cela une connexion ethernet en pair à pair, qui est activable depuis n'importe quel ordinateur avec une copie du programme et le fichier de configuration associé au robot, une simple commande suffit (**`make sshconf CON_MODE=eth -B`**). Mais il est plus utile de se connecter au robot par liaison sans fil (**`make sshconf CON_MODE=wifi -B`**), pour cela il faut connecter l'ordinateur et le robot (toujours à l'aide du fichier de configuration) sur le même réseau Wi-Fi (l'utilisation d'un partage de connexion mobile s'est révélé être la solution la plus pratique).

#### *5.4.2.2. Mise à jour des programmes*

Une fois le lien établi entre l'ordinateur du client et le robot, il lui est possible grâce à une simple commande de recompiler le programme embarquant le parcours (**`make upgrade-chef`**), et de mettre à jour si besoin le système même du Raspberry Pi (**`make`**

**upgrade-filesystem**), ou les programmes du FPGA (**make upgrade-fpga**) et de l'Arduino (**make upgrade-arduino**).

#### *5.4.2.3. Lancement des programmes*

Une fois connecté au robot via SSH (**make ssh**), différents programmes sont disponibles dans le répertoire de test (accessible avec l'alias **c**). **bin/premier** est le programme lancé automatiquement lors de l'allumage du robot et permet la sélection par l'interface humain-machine embarquée des différentes actions. **bin/testXXX** sont des programmes permettant de tester individuellement différents aspects du robot (récupération des données des capteurs, test des actionneurs…). Ils ont été notamment utiles pour valider nos théories au fur et à mesure de nos avancées.

Il est d'ailleurs possible d'accéder à l'affichage de l'écran et aux boutons de l'IHM embarquée lorsqu'un programme est lancé dans ce mode, via l'entrée et la sortie standard.

La commande par SSH ne permet pas un contrôle complet des actionneurs du robot (et ce n'est ni adapté ni le but recherché), mais quelques alias d'urgence sont fournis tels que **s** pour bloquer les roues et arrêter les actionneurs et **f** pour libérer les roues.

### 5.4.3. Amélioration des programmes

#### *5.4.3.1. Débogage à distance*

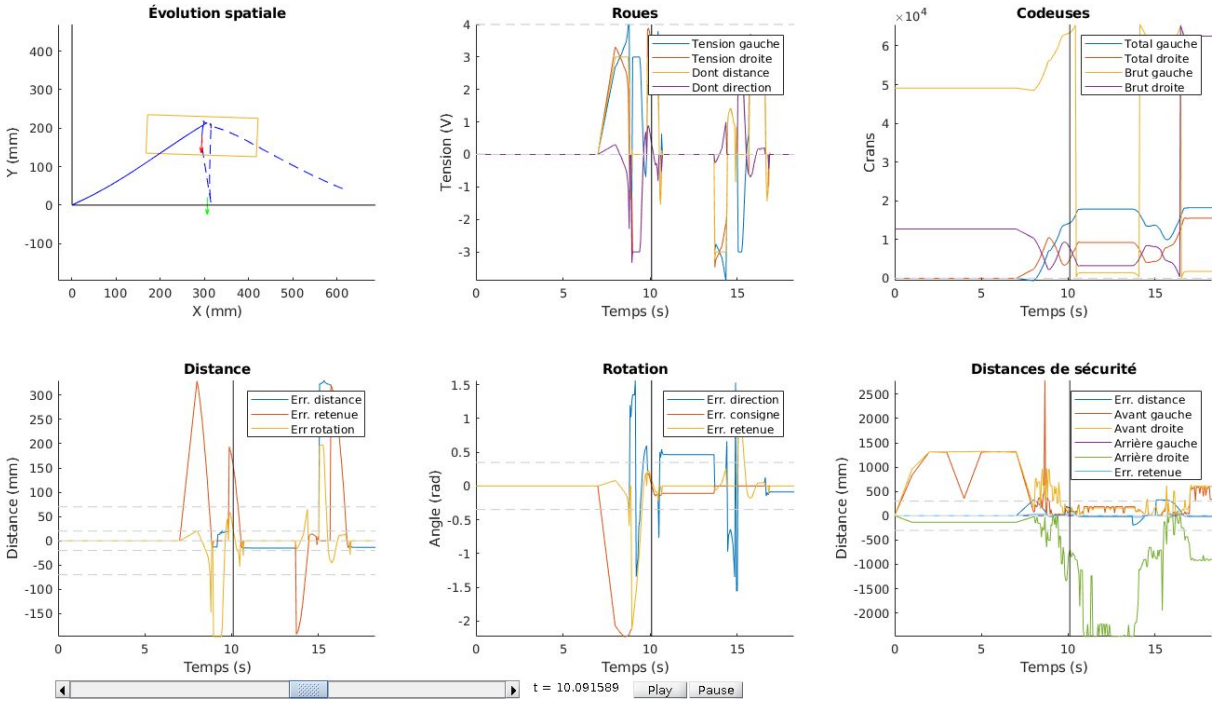
Les outils de débogage usuels tels que **gdb** ou **ddd** ne sont pas installés sur le système d'exploitation cible pour des raisons de place, cependant il est toujours possible de déboguer à distance à l'aide de **gdb-server** sur le Raspberry Pi et de gdb sur un ordinateur configuré en mode télécommande. Il suffit simplement de taper la commande **make debug**.

#### *5.4.3.2. Consultation des journaux*

Chaque programme (excepté certains programmes de tests) enregistre des journaux sur les différentes valeurs qui sont reçues par les capteurs, utilisées pendant des calculs intermédiaires, ou envoyées aux actionneurs durant l'intégralité de son fonctionnement. Elles sont stockées dans des fichiers **.csv** et peuvent être récupérées sur l'ordinateur (**make get-logs**).

Il est ensuite possible de visualiser l'évolution de ces valeurs au cours du temps à l'aide du logiciel Matlab[4] et du script fourni.

[4] Matlab n'est pas forcément le logiciel le plus adapté, la raison de ce choix réside dans les pistes de travail abandonnées.




*Exemple de visualisation de journaux d'une session d'un parcours de test*

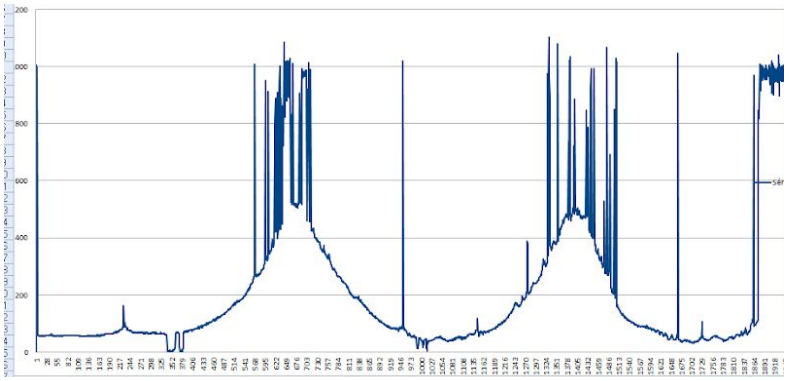
## 5.5. Gestion des capteurs

### 5.5.1. Nécessité du filtre

Lors de la compétition, le robot doit dans ses déplacements assurer un système d'anti-collision. Cependant, les autres robots présents sur la pistes disposent d'un système similaire, et pouvant être exactement le même que le nôtre. Cela peut donc engendrer des interférences. Notre rôle est de filtrer le signal, afin de se débarrasser de ces interférences, en en profitant pour réduire la sensibilité des capteurs aux bruits ultrasonores ambiants et et à leur propre bruit caractéristique.

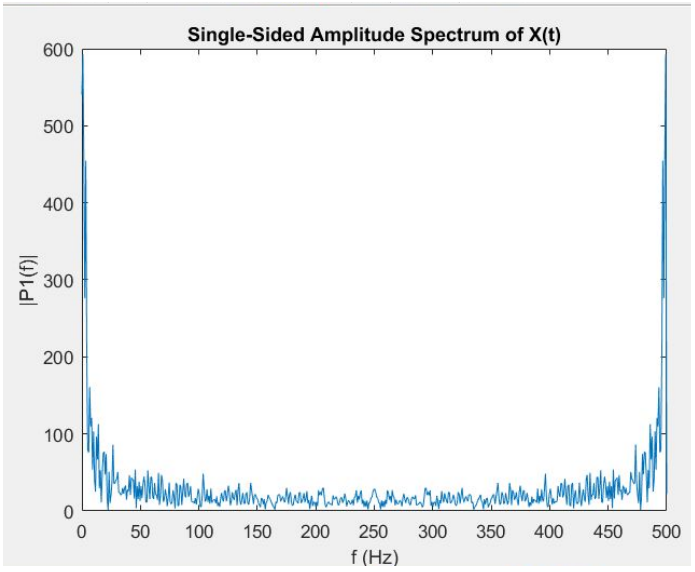
### 5.5.2. Traitement des données

Observons les valeurs obtenues par un capteur HCSR04 en l'approchant d'un objet et d'un autre capteur en fonctionnement :



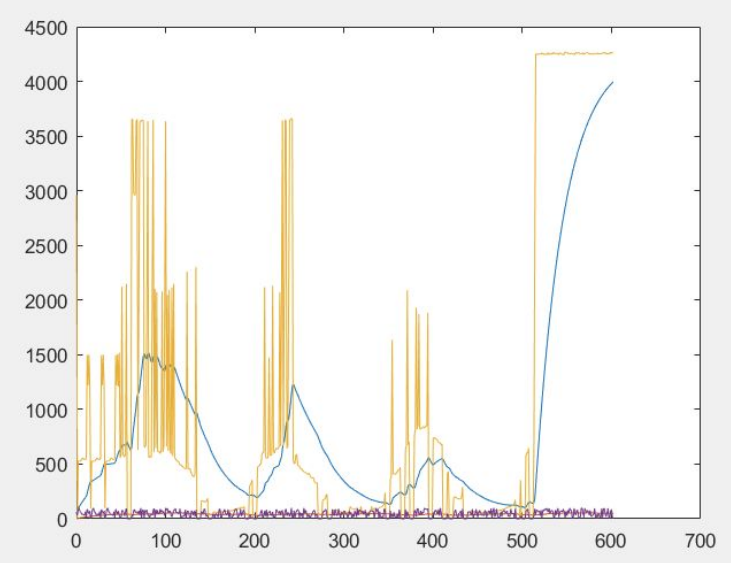
*Valeurs obtenues en approchant puis éloignant un objet à deux reprises d'un capteur HCSR04*

Une transformée de fourier rapide sur le signal obtenue nous montre que le signal d'intérêt se trouve à très basse fréquence autour de quelques Hz. Il faut donc trouver un compromis entre un filtre trop basse fréquence créant un moyennage trop important un filtre trop haute fréquence n'ayant pas d'effet.




*FFT d'un signal de capteur HCSR04 en fonctionnement*

En choisissant un filtre passe-bas d'une fréquence de coupure d'environ 5 Hz, on trouve un bon compromis entre l'efficacité du filtre et sa réactivité, ce qui est une composante importante pour l'évitement d'obstacle.




*Comparatif d'entrée et de sortie du filtre*

#### 5.5.3. Implémentation du filtre

Elle sera faite directement sur le FPGA, le Raspberry Pi n'aura qu'à récupérer les valeurs filtrées directement. À chaque lecture d'une nouvelle valeur sur le capteur, la valeur filtrée sera mise à jour à l'aide d'un filtre à réponse impulsionnelle finie, dont les coefficients ont été calculés à l'aide d'un logiciel dédié à partir des caractéristiques déterminées ci-dessus.

Le temps de calcul du filtre est toutefois assez important, en effet 120 ms sont nécessaires pour avoir deux échantillons de capteurs et ainsi récupérer une différence sur la sortie du filtre. Le Raspberry Pi anticipe cependant l'obstacle, en ralentissant en le voyant au loin (cela est réalisé en limitant la distance cible), mais dans le cas d'un scénario queue de poisson, à sa vitesse maximale le robot a le temps de se déplacer de 16 cm avant de commencer à ralentir, ce qui rend l'impact inévitable.[5]

### *5.6. Protocole de communication*

L'asservissement en position repose en grande partie sur la communication entre le FPGA et le Raspberry Pi pour faire l'aller-retour entre les données des encodeurs et les consignes pour les contrôleurs moteur. Nous avons donc décidé de créer un protocole de transfert d'information bi-directionnel optimisé pour notre usage. Ce dernier sépare les messages en deux types de messages : ceux dont l'information transitent du FPGA au Raspberry Pi et ceux qui font le chemin inverse. Chaque message peut être réparti en trois catégories :

[5] En compétition on a préféré désactiver partiellement le filtre. Le robot ralentissait inutilement mais la sécurité était assurée.

1. Les messages **directs** : Le contrôleur possédant l'information l'envoie à chaque fois qu'elle est mise à jour (ex : mise à jour de l'ordre à appliquer sur les contrôleurs moteurs). Le contrôleur recevant l'information peut ré-envoyer un message de même type mais vide indiquant la bonne réception / la fin du traitement.
2. Les messages **indirects** : Le contrôleur nécessitant l'information envoie un message vide pour demander la récupération de la valeur. Le contrôleur possédant l'information renvoie un message du même type. Cela permet de ne pas envoyer d'informations quand le contrôleur n'est pas prêt à la recevoir (ex : données des encodeurs).
3. Les messages sur **seuil** : Le contrôleur récupérant l'information transmet à celui qui la possède un niveau de seuil. Dès que ce seuil est dépassé, le contrôleur possédant l'information envoie un message informant du franchissement. Cela permet de ne pas envoyer d'information quand elle n'est pas utile (ex : données des capteurs de distance[6]).

Chaque message est préfixé par un caractère indiquant son type ce qui permet, à partir des définitions des messages au préalable et du sens dans lequel le message est envoyé, de déterminer sa taille et l'information qu'il cherche à envoyer.

# 6. Pistes de travail abandonnées

Lors de notre travail, nous avons exploré différentes pistes qui se sont avérées non-viables ou qui ont été supplantées par une solution plus simple. Nous en parlerons brièvement dans cette partie.

## *6.1. Utilisation d'un Arduino comme intermédiaire entre le FPGA et le Raspberry Pi*

Doutant de la réactivité du Raspberry Pi entre le moment où il reçoit une mise à jour des données des codeurs et le moment où il renvoie une instruction asservie pour les contrôleurs moteurs, il était initialement prévu d'utiliser un Arduino comme moyen de calculer la position du robot et de générer les consignes asservies. Cependant, il s'avère que le calcul de nombres en virgule flottante sur un Arduino est extrêmement long, bien plus que la réactivité du Raspberry Pi, qui a été d'ailleurs grandement améliorée par l'utilisation d'un système d'exploitation personnalisé.

On pensait par ailleurs que l'Arduino serait indispensable pour contrôler les moteurs, car tous les contrôleurs ont une référence sur 5 V, et seul l'Arduino peut fournir une PWM

[6] Dans le cas des capteurs HCSR04, la fréquence d'échantillonnage fixée à 60 ms n'aurait que peu d'impact sur la liaison série. Cependant cela aurait pu être le cas pour des capteurs avec un plus grand taux de rafraîchissement (ex : capteurs infrarouge).

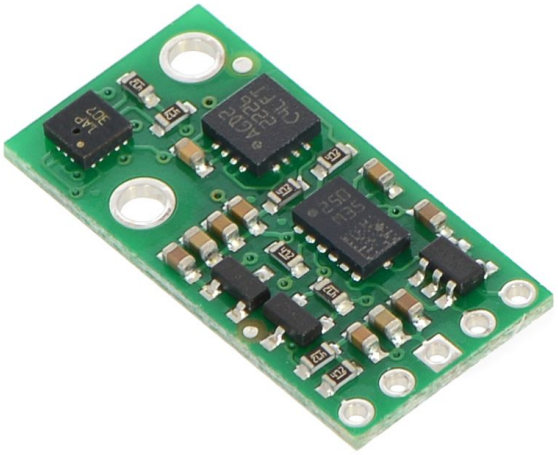
allant jusqu'à 5 V. Cela n'a cependant pas été un problème, les contrôleurs ayant pour tension de base du 24 V et l'utilisation que nous avons finalement faite des moteurs ne dépassant pas les 6 V, la tension maximale de 3,3 V comme PWM pour les autres contrôleurs était déjà suffisante.

Une confusion avait d'ailleurs été commise, l'IMU aurait été dans ce cas connecté au FPGA, à l'aide du protocole de communication SPI, pour lequel l'Arduino possède un accélérateur matériel. Sauf que cette dernière communique en fait via I²C, protocole impossible à mettre en place sur un Arduino aussi chargé.

Le seul avantage vraiment perdu avec l'abandon de cette technique est que le changement de programme sur le Raspberry Pi n'induit pas forcément un recalibrage de la position, cette dernière étant stocké sur l'Arduino. Mais si on le souhaitait cette fonctionnalité aurait pu être ré-implémentée sur le Raspberry Pi logiciellement.

### *6.2. Utilisation d'une centrale inertielle*

Dès la compétition de l'an dernier, il était acquis que se baser uniquement sur les encodeurs intégrés au moteur n'était pas suffisant pour se repérer correctement sur la piste. En effet, il est très commun d'obtenir des dérapages de la roue sur la piste suite à des accélérations / décélérations trop brutales[7]. C'est pour cela qu'il avait été fait l'acquisition d'une centrale inertielle (ou IMU) embarquant un accéléromètre et un gyroscope afin d'avoir une autre source sur la position du robot sur la piste.



*MinIMU-9 v3*

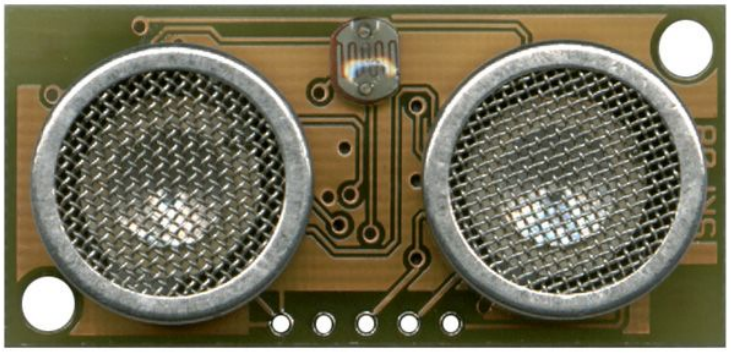
Ce type de capteurs est normalement plus utilisé pour des grandes valeurs d'accélération, ce que nous ne produisons pas. Différents tests de récupération de

[7] En revanche, l'utilisation de roues codeuses externes en plus des roues alimentées est plus approprié. En effet, il devient possible d'appliquer une pression à l'aide d'un ressort afin que la roue ne glisse que très peu sur la piste *(source : équipe Arfit de Polytech Tours)*

valeurs le confirment : il y a un bruit assez conséquent, et estimer la position du robot à partir de ces valeurs prises au repos font croire que le robot oscille dans tous les sens alors qu'il est en réalité statique.

L'idée n'était pas de se servir de l'IMU comme moyen de récupérer les valeurs de position au long du trajet du robot, mais de s'en servir comme moyen de détecter les incohérences avec les valeurs des encodeurs, à l'aide d'un outil mathématique appelé filtre de Kalman, permettant de sélectionner l'utilisation des valeurs du capteur le plus cohérent à l'instant $t$. Cependant, l'implémentation d'un tel filtre dans notre cas était assez complexe et chronophage, et nous n'étions pas sûr que cela améliorerait nos résultats, nous avons donc préféré limiter les accélérations / décélérations de notre robot. En revanche, la récupération des données de l'IMU et son traitement est parfaitement fonctionnelle, ce qui laisse la possibilité à un futur automaticien d'implémenter ce fameux filtre.

### 6.3. Utilisation des capteurs SRF08

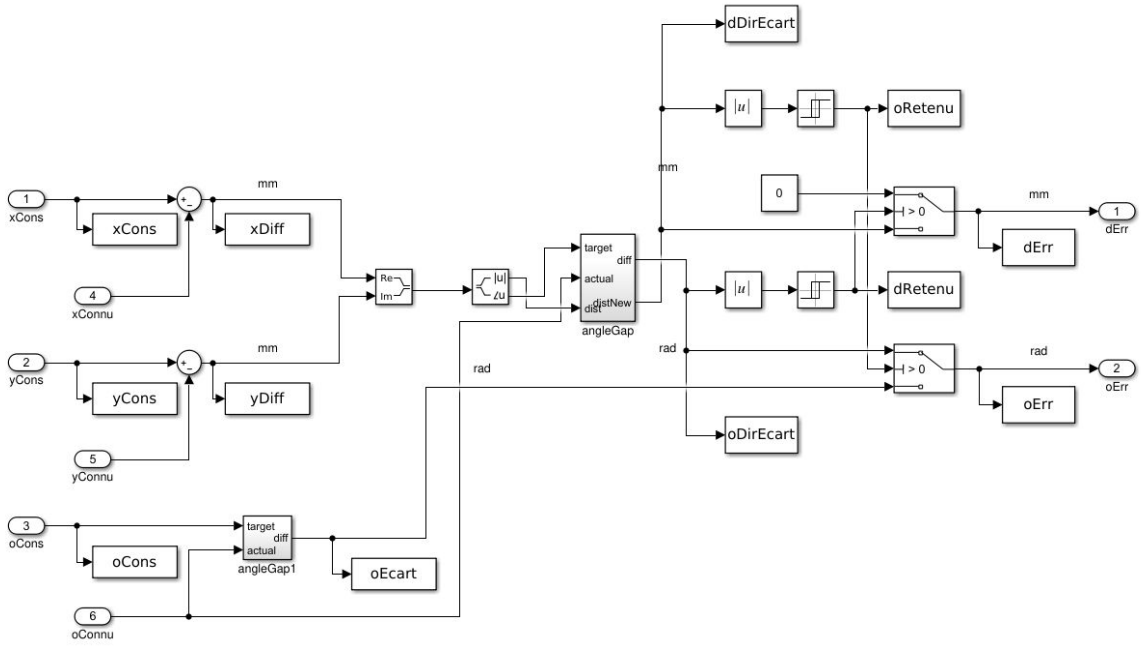


*Capteur SRF08*

Nous disposions de capteurs à ultrasons de type SRF08. Ces derniers se distinguent des HCSR04 de part leur communication par protocole I²C permettant l'accès aux temps de captation des différents rebonds d'écho. Là où un HCSR04 serait capable de nous donner la distance à l'obstacle le plus proche, le SRF08 serait capable de nous donner la distance au premier, second, troisième jusqu'à 5ème obstacle (ou alors les rebonds d'aller-retour des premiers obstacles, facilement filtrable car ce sont des multiples de la première captation). Ces valeurs nous auraient probablement permis après étude de mieux détecter les interférences en analysant le motif des différents rebonds, et ainsi éviter l'utilisation d'un filtre augmentant le délai de transfert de l'information. Cette piste a été mise de côté car la détection d'obstacle à base de filtre était déjà très adaptée à nos besoins.

### 6.4. Simulation du système

Afin d'éviter quelques tests en conditions réelles, plus complexes à réaliser, nous avons commencé à réaliser une simulation du robot à l'aide d'un programme dédié à base de schéma blocs. Le but ultime était de réaliser une simulation suffisamment complète pouvant permettre de trouver des paramètres optimaux pour le réglage de l'asservissement. Cependant un grand nombre d'équation et de variables rentre en compte dans la réalisation d'un tel système : bilan des forces, transmission de couple, conversion électromécanique, résistance au frottements… La réalisation d'une telle simulation viable constituerait un projet à lui tout seul.




*Algorithme de génération de consigne sous forme de schéma blocs*

Les efforts ont donc été rapidement abandonnés, mais le peu qui a été fait a permis de valider sur une portée limitée les algorithmes de calcul de la position et de génération de la consigne. Le code utilisé pour générer les graphiques de la simulation a été modifié pour afficher les valeurs réelles enregistrées par le robot.

### 6.5. Utilisation d'un autre contrôleur moteur

#### 6.5.1. Utilisation du TLE5206

L'utilisation d'un contrôleur moteur de référence TLE5206 était envisagée. En effet, en théorie il avait tout l'air d'un bon rival pour le LMD18200 :

| Composant | LMD18200 | TLE5206 |
|---|---|---|
| Courant délivré | 3A | 5A |
| Courant délivré (pic) | 6 A sur 200 ms | 6 A sur 100 ms |

| | | |
|---|---|---|
| Tension maximale | 60 V | 40 V |
| Température de fonctionnement maximale | 125°C | 150°C |
| Technologie | Pont en H complet CMOS DMOS | Pont en H complet CMOS DMOS |
| Bootstrap | Inclus | Non inclus |
| Dimension (projection verticale en mm) | 20 × 4.5 | 10 × 2.4 |
| Prix (fournisseur : farnell) | 14,68 € | 3,87 € |

Nous avons réalisé le contrôleur moteur à base de TLE5206 et il s'est révélé brièvement fonctionnel avec le robot principal. Cependant nous avons observé des incohérences dans son comportement. Malgré nos efforts, nous n'avons jamais pu obtenir un comportement constant de celui-ci même avec l'usage de composants parfaitement neufs et en respectant à la lettre les composants conseillés dans la datasheet.

Nous avons pris la précaution de commander les deux types composants en même temps, et que si les deux étaient toutefois fonctionnels, l'un pourrait servir de remplacement en cas de problème. Cette précaution s'est révélée utile. Nous avons pu constater que les composants dont le prix et les caractéristiques sont alléchantes ne sont pas forcément de bonne facture[8].

### 6.5.2. Utilisation d'un transistor de puissance

Il était également question de réaliser un contrôleur moteur avec un pont en H réalisé par nos soins, cependant nous avons dû revoir nos objectifs pour nous concentrer sur l'essentiel nécessaire au fonctionnement du robot, non sans regrets car cela aurait sans aucun doute une expérience enrichissante.

[8] Dans la même veine, nous avons commandé trois câbles USB ↔ Série auprès d'un même vendeur chinois en début d'année. Les trois sont tombés en panne avant la fin du projet. Notre système n'est cependant pas à mettre en cause, l'un n'ayant servi qu'avec un Arduino branché nul part.

# 7. Bilan et perspectives

## *7.1. Apprentissage reçu*

Ce projet proposé par les étudiants et accepté par nos professeurs nous a permis de relever le défi de réaliser un système avec un objectif extérieur : réaliser un robot pour une participation la Coupe de France de robotique.

La conclusion qui nous vient en premier est l'apprentissage que nous tirons de ce projet, tant sur l'aspect technique que sur l'aspect humain.

D'un point de vue technique, une majeure partie des tâches prévues avaient déjà été au moins partiellement traitées durant notre enseignement. Cependant, la réalité de l'application à un cas concret nous a mis face à des difficultés que nous n'étions pas habituées à rencontrer lors des TP et TD conditionnés. La duretée d'un travail concret ne pardonne pas les approximations et les erreurs, et nous avons été poussé à revoir nos objectifs en fonction de cette réalité.

Du point de vue humain et de gestion de projet, il s'agissait d'un projet d'envergure nécessitant un travail de groupe et la bonne gestion de paramètres extérieurs. Nous avons compris l'importance d'une organisation solide et rigoureuse qui doit être maintenue tout au long du projet si on souhaite que celui-ci aboutisse.

Nous pouvons aujourd'hui voir à quel point nous avions sous estimé le temps que nous prendraient les inconnues lié aux actionneurs sur le robot. Cela a nécessité de nombreuses réunions, discussions et parfois débats avec le client, et a pu nous ralentir pour le choix des composants ou la réalisation de tests en conditions réelles (le robot étant inutilisable car en cours de construction pour les actionneurs par exemple) mais c'est un apprentissage à part entière du métier d'ingénieur.

## *7.2. Gestion de la démarche de projet*

Lors de ce projet nous avons énormément progressé dans la démarche d'ingénieur. En effet, au fil des heures de travail et des échanges avec les encadrants, nous avons pu constater la rigueur et l'exigence que requiert une vraie démarche ingénieur. Nous avons pris la juste mesure de ces éléments et avec le recul que nous permet la rédaction de ce rapport, nous pouvons voir que la démarche que nous avons adoptée n'a pas toujours été optimale.

Une structuration de l'avancement du projet en plusieurs sous-étapes, avec chacune une phase de conception, réalisation et validation aurait été plus judicieuse. Cela nous aurait

permis de gagner du temps en évitant de partir sur des pistes non viables qui auraient pu être évitées par non-validation d'une étape précédente.

Ici, nous étions parti sur une longue phase de conception, suivi d'une longue phase de réalisation, et enfin d'une longue phase de validation, qui se serait transformée en phase de débogage et re-conception sans l'intervention en mi-parcours de nos encadrants pour nous faire comprendre cet aspect de la démarche ingénieur. Sans cette intervention, le projet n'aurait probablement pas abouti au même résultat.

### *7.3. Essence de la compétition*

La participation à la compétition nous a permis de se remettre en question sur la pertinence de la participation à la Coupe de France de robotique. En effet, nous avons pu constater deux différences majeures entre l'équipe de Robotech et les équipes de têtes.

On constate tout d'abord que les équipes sont constituées en majorité par des étudiants en mécanique. Ces équipes utilisent le plus souvent des composants électroniques déjà réalisés (ex : une carte complète possédant plus de 30 entrées pour capteurs avec carte de puissance intégrée ainsi que des contrôleurs moteurs avec encodeurs), et utilisent en conséquence des logiciels généralement propriétaires pour le contrôle, avec une programmation généralement graphique (à base de Labview le plus souvent). Cela va de pair avec des moyens financiers généralement beaucoup plus importants.


*Robot de l'équipe ESEO, finaliste à la coupe 2018*

Une autre grande différence réside dans le suivi du projet d'année en année, en effet les équipes ont au moins un professeur ou un ancien qui survit au renouvellement des équipes et fait ainsi profiter de son expérience de la compétition. De plus, la gestion de projet qui sous-tend la réussite de cette compétition est très importante, motiver et faire avancer un groupe d'étudiants qui travaillent bénévolement en dehors des cours et des

projets est un travail à part entière qui joue sur la capacité du club étudiant à avancer durant l'année.

Il est donc de façon réaliste assez peu envisageable que Robotech remporte un jour la compétition en suivant la même démarche : nous concentrons nos efforts sur certains aspects (électronique, informatique), là où les autres équipes utilisent des solutions clefs en main, et nous négligeons certains aspects (mécanique) qui semblent primordiaux pour la réussite de la coupe.

L'objectif n'était cependant pas la victoire, mais la mise en place de solutions que nous pouvons réaliser à l'aide de nos compétences acquises durant notre enseignement, et la participation à cette coupe est un moyen intéressant d'y parvenir, même si nous ne jouons pas dans la même cour que les autres équipes.

## 8. Conclusion

Au cours de ce semestre, nous avons répondu au cahier des charges demandé par notre client en concevant un système modulable, simple d'utilisation et fonctionnel permettant l'alimentation et le déplacement d'un robot pour participer à la Coupe de France de Robotique. Nous avons également appris à appliquer dans un cas concret la démarche ingénieur qui nous servira indubitablement lors de nos expériences professionnelles futures et de la poursuite de nos études.

# 9. Annexes

## *9.1. Sources*

Les sources pour la réalisation du projet (PCBs, FPGA, système d'exploitation du Raspberry Pi et programme principal), ainsi que celui pour la simulation / affichage est disponible sur le dépôt git suivant : https://archives.plil.fr/gbontoux/cdf2018-principal/

Il est à noter que ce code contient par nécessité des parties de code réalisées par des membres de Robotech autre que nous, ou par nous mais hors du cadre du projet. Dans ce cas, c'est indiqué en en-tête des commentaires de fichier. De manière générale, la provenance des parties de codes n'étant pas écrite par nous est indiquée à l'intérieur des fichiers, à l'exceptions des sous-modules git.

Suite à des problèmes matériels pendant la Coupe de France de robotique, la liaison série reliant normalement le Raspberry Pi au FPGA a été remplacée par une liaison I²C, ce qui est visible dans le code après le commit « *fe3ae8ef* Serial → I2C ». Cependant, le code de la liaison série étant plus intéressant que celui de la liaison I²C (ce dernier ayant été programmé en urgence et majoritairement récupéré depuis internet), nous souhaiterions être de préférence évalué sur la première (le lien vers les fichiers concernés se trouve sur le Wiki). En revanche, le reste du code est parfaitement utilisable avec le dernier commit, et est d'ailleurs mieux commenté.

## *9.2. Liens*

### 9.2.1. Utile

1. Cahier des charges de la compétition : https://www.coupederobotique.fr/wp-content/uploads/C2018_Rules_final_FR.pdf
2. Datasheet des moteurs avec traduction des éléments importants: https://projets-ima.plil.fr/mediawiki/images/8/87/P66_Moteurs_principal.pdf
3. Calculateur de résistance pour alimentation à découpage : https://docs.google.com/spreadsheets/d/12o-t0-J8r7Gmm0sNj0GpuK4PqRz_7LqPpdcwkLS9ias/edit?usp=sharing

### 9.2.2. Datasheet des composants utilisés

1. Datasheet des codeuses : https://www.infineon.com/dgdl/Infineon-Encoder_HEDS-5540-A14-AP-v01_00-EN.pdf?fileId=5546d46147a9c2e40147d3d593970357
2. Datasheet des composants de l'IMU : https://www.pololu.com/product/2468/resources

3. Datasheet du TLE5206 : https://www.infineon.com/dgdl/Infineon-TLE5206_2-DS-v01_01-en.pdf?fileId=db3a30431f848401011fc753a71779a7
4. Datasheet du LMD18200T : http://www.ti.com/lit/ds/symlink/lmd18200.pdf
5. Datasheet de l'alimentation à découpage : http://www.ti.com/lit/ds/symlink/ptn78020w.pdf
6. Datasheet du convertisseur 24V : https://www.mouser.fr/datasheet/2/281/uwe-57928.pdf

#### 9.2.3. Manuel des programmes / bibliothèques utilisées

1. Manuel de Buildroot : https://buildroot.org/downloads/manual/manual.html
2. Manuel de WiringPi : http://wiringpi.com/reference/setup/
3. Implémentation d'une liaison série en VHDL : https://github.com/pabennett/uart (sous license Apache 2.0)

### *9.3. Information sur les batteries*

Un match lors de la compétition ne dure que 100 secondes donc la problématique d'autonomie ne se pose pas pour nos batteries vu les consommations mesurées par le robot.

Nous avons 3 types de batteries :

- LiPo 4S 2400Mah 20C
- LiPo 4S 3800Mah 20C
- LiPo 4S 4000Mah 25C

#### 9.3.1. Que signifie 20C ou 25C ?

Ce chiffre indique la quantité de courant qui peut-être tirée de l'accumulateur, c'est la **C**apacité de décharge. Pour connaître la charge maximum que peut supporter une batterie, on multiplie la taille de celle-ci par sa capacité de décharge ainsi nos batteries délivrent respectivement :

- 48A (=2.4*20)
- 76A
- 100A

Notre capacité de décharge est donc amplement suffisante pour la compétition et cela permet au robot de ne transporter qu'une batterie LiPo (qu'on veillera à changer régulièrement) de 2400mAh et ainsi gagner une place considérable sur le robot.

#### 9.3.2. Notions autour de la décharge des batteries LiPo

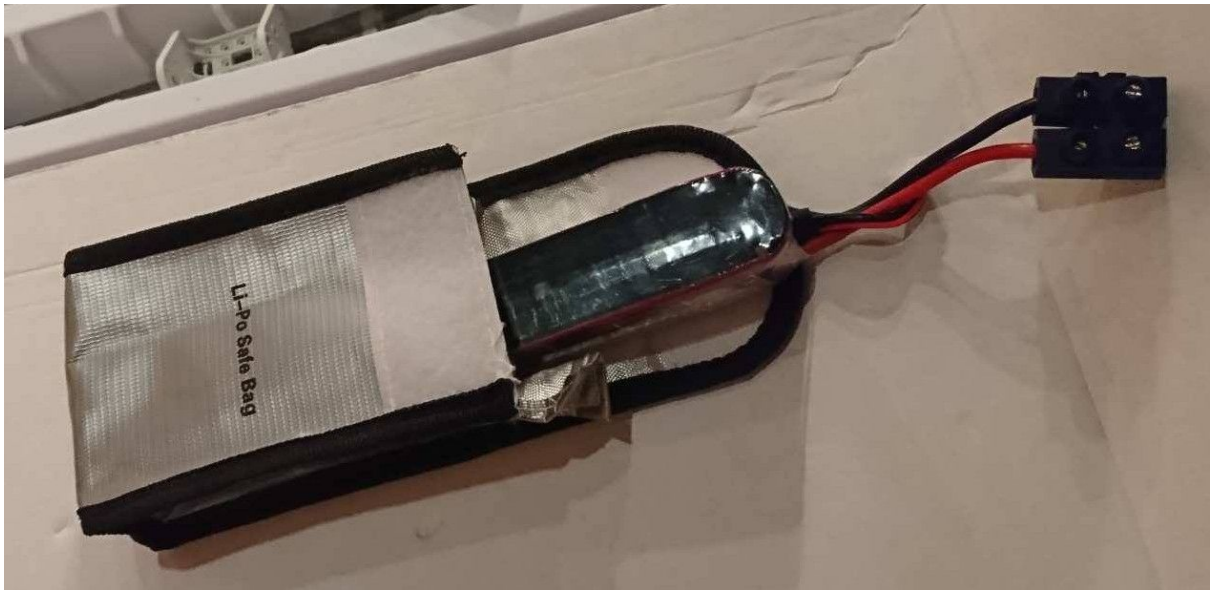
Une batterie LiPo, pour être réutilisable, ne doit pas descendre en dessous d'un niveau de charge qui se mesure simplement à l'aide de la tension de sortie de la batterie. En

effet, passé un certain cap les cellules ne sont plus rechargeables et la batterie est irrécupérable.

Si certaines batteries sont pourvus d'un indicateur de niveau de charge ce n'est pas le cas de toutes et il est toujours prudent de garder en tête le tableau suivant

| | 1 él | 2 él | 3 él | 4 él |
|---|---|---|---|---|
| 0% | 3,00V | 6,00V | 9,00V | 12,00V |
| 5% | 3,30V | 6,60V | 9,90V | 13,20V |
| 10% | 3,60V | 7,20V | 10,80V | 14,40V |
| 20% | 3,70V | 7,40V | 11,10V | 14,80V |
| 30% | 3,75V | 7,50V | 11,25V | 15,00V |
| 40% | 3,79V | 7,58V | 11,37V | 15,16V |
| 50% | 3,83V | 7,66V | 11,49V | 15,32V |
| 60% | 3,87V | 7,74V | 11,61V | 15,48V |
| 70% | 3,92V | 7,84V | 11,76V | 15,68V |
| 80% | 3,97V | 7,94V | 11,91V | 15,88V |
| 90% | 4,10V | 8,20V | 12,30V | 16,40V |
| 100% | 4,20V | 8,40V | 12,60V | 16,80V |

### 9.3.3. Notions de sécurités autour de l'utilisation des batteries LiPo




*Batterie LiPo dans son sac ignifugé*

Ces batteries très populaires dans le secteur du modélisme et de la robotique amateur possèdent un défaut majeur : leur instabilité. En effet, à tout moment une batterie LiPo est susceptible d'entrer en combustion spontanée impliquant gazs toxiques, flammes, températures très élevées et explosion. Se présente ainsi la nécessité de posséder des sacs de protections ignifugés labellisés "LiPo Safe".

La probabilité de combustion de la batterie augmente également en cas d'impact violent et lors de la recharge de la batterie.

Ces notions bien qu'indirectement liées au projet nous semblent importantes à préciser de part leurs caractères critiques. Un départ de feu sur le robot n'est clairement pas souhaitable.