Rapport de projet – Semestre 8

P13 – Robot à système de propulsion déformable

Vincent Dubois
Damien Tillaux IMA-SC2020

Table des matières

Introduction

Sur le marché actuel, on trouve très peu de robots amphibies. En effet, les systèmes de propulsion traditionnellement utilisés sont développés pour des tâches bien précises et ne sont pas adaptés pour fonctionner dans différents environnements. De nouvelles techniques voient le jour, notamment avec l'émergence de la robotique déformable (Soft Robotics). Ce domaine s'inspire fortement de la manière dont les organismes vivants se déplacent et s'adaptent à leur milieu pour accroître la mobilité des systèmes électromécaniques en utilisant des matériaux à la fois rigides et déformables.

Une grande majorité des animaux amphibiens utilise un mode de déplacement très particulier propre à leur anatomie. On retrouve notamment le mouvement d'ondulation caractéristique de certains serpents aquatiques. Ce même mouvement est également repris par la raie manta, lui permettant de se déplacer dans l'eau. Seul l'inclinaison de l'ondulation (sur un plan horizontal ou vertical) différencie le mode de déplacement de ces deux animaux. L'objectif principal de ce projet est donc de développer un robot doté d'un système de propulsion par ondulation lui permettant d'évoluer aussi bien dans un milieu aquatique que sur la terre ferme.

Ce rapport relate les avancées du projet au semestre 8. Après un bref rappel des objectifs et des réalisations des précédents semestres, nous analyserons les solutions développées sur les aspects propulsion et navigation du robot. Enfin, nous conclurons sur les potentielles limites et améliorations du projet.

I. Présentation générale du projet

1.1. Redéfinition des objectifs

Le projet initialement choisi au semestre 6 avait comme intitulé « Réalisation d'un manipulateur déformable ». Il avait pour objectif de travailler sur la création d'un bras robotisé déformable avec une infinité de degrés de liberté. Cependant, nous avons été plus attirés par l'un des projets présentés par l'équipe de recherche du tuteur : un petit robot se déplaçant en se contorsionnant grâce à sa structure souple. Nous nous sommes donc tournés vers la problématique du déplacement grâce à la robotique molle.

En plus de développer le système de propulsion, nous avons fait le choix de concevoir la partie navigation du robot. Ce dernier doit pouvoir être contrôlable à distance par un pilote mais également se déplacer de manière autonome et s'adapter à son environnement. Nous nous sommes fixés comme contrainte d'utiliser le maximum de ressources « Open source » afin que le projet puisse être réutilisable par n'importe qui et dans n'importe quel cadre.

1.2. Axes de travail

Au semestre 8, le projet a été organisé selon 2 axes principaux : la propulsion et la navigation.

La première partie vise à finir la réalisation du système de propulsion par ondulation afin que le robot puisse se déplacer dans n'importe quel milieu. Nous discuterons du dimensionnement des nageoires ainsi que de l'algorithme qui permet l'ondulation.

La seconde partie concerne le contrôle du robot. Ayant déjà réalisé le mode de navigation manuelle (avec une télécommande) au semestre 7 nous nous sommes attardés sur le déplacement autonome. Nous verrons ainsi les différentes pratiques pour reconnaitre l'environnement et s'y déplacer.

II. Propulsion

L'un des enjeux importants du sujet est de reproduire la propulsion par ondulation. Les problématiques que nous nous sommes posées au début du sujet étaient :

- Quel type de moteur utilisé pour faire onduler les pales.
- Combien de moteur utilisé ?
- Comment contrôler le robot ?
- Quelle forme donnée au robot ?
- Comment concevoir les pales ?

Toutes ces problématiques nous ont alors mené à travailler sur la forme physique et la conception du châssis, l'électronique embarquées que nous avons sélectionné et leurs utilisations, la conception d'algorithme de d'ondulation des pales pour rendre le déplacement possible.

2.1. Conception des pales

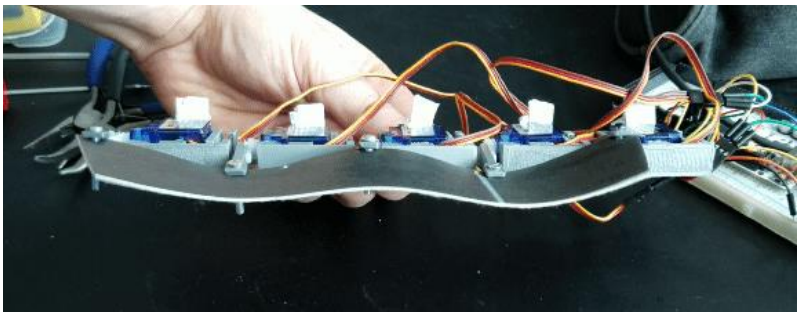
Après le châssis, la conception des pales était un point important du projet. Au début du projet plusieurs hypothèse sur le matériau et la fabrication des pales ont été posées.

Au niveau de la méthode de fabrication des pâles nous avions comme idée de soit les découper directement dans un matériau ou d'usiner un matériau sous forme d'une mésostructure ce qui permettrait d'utiliser un matériau rigide et de le rendre plus souple sans trop affecter sa solidité.

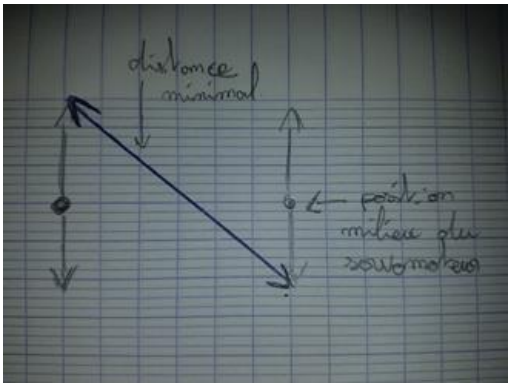
Au niveau du matériau utilisé nous avions pensé à du PLA et imprimer, via imprimante 3D, une pale en mésostructure. Cette idée a été très vite écarté car très peu d'information existe à ce sujet et que faire cela nous même aurait été une perte de temps et hors de nos compétences enseignées en IMA. L'utilisation de caoutchouc ou d'un matériau du même type ou avec les mêmes caractéristiques de souplesse avec un aspect solide nous a alors paru idéal pour concevoir les pales.

Nous avons alors choisi d'utiliser du caoutchouc pour réaliser les pales. Nous avons alors commandé un rouleau lors du semestre 5 et en attendant de le recevoir nous avons réalisé les premières pales en lino pour déjà appréhender la forme que nous donnerons aux pales et savoir si notre hypothèse sur le nombre servomoteur était faisable.

Nous avons alors réalisé différents tests sur la forme des pales en lino pour voir quelle méthode de découpe été la plus optimal pour obtenir des ondulations permettant le déplacement du robot.



La première pale réalisée était sans courbure avec une largeur de 4cm, donc respectant la largeur maximum de 8 cm. Le problème sur ce type de pale c'est que nous ne pouvons pas mettre que la distance entre chaque fixation sur la pale ne peut pas être égale à la distance entre les servomoteurs. La distance entre chaque fixation doit alors être au minimum la distance entre la position haute et la position basse des servomoteurs.



Nous avons réussi à obtenir une ondulation. Mais nous voulions l'amplitude de l'ondulation plus importante sans modifier la différence d'angle maximum des servomoteurs. Pour cela nous avons courber la découpe des pales pour que quand elles se retrouvent fixé robot, la longueur intérieure soit plus petite que la longueur extérieure de la pale et permettra de ce fait une amplitude d'ondulation plus importante sur l'extérieur de la pale qu'en intérieur.

Mais après plusieurs tests, nous avons déduit que mettre une forte amplitude sur les ondulations, imposait de trop forte contrainte sur les servomoteurs. Nous avons aussi remarqué que nous avions besoin de 6 servomoteurs par flanc pour obtenir une ondulation mieux contrôlée.

Nous devons alors maintenant réaliser des pales dans un matériau plus rigide car le lino ne permet pas une rigidité assez forte pour supporter le poids du robot.

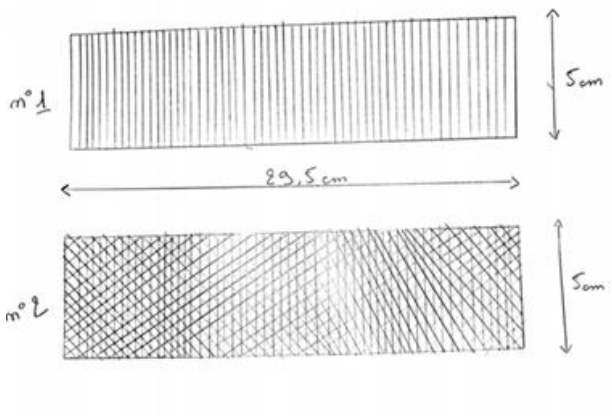
Après réception du rouleau de caoutchouc nous avons pu réaliser les pales dans un matériau permettant la résistance du poids de robot. La première pale réalisée fut juste une pale droite sans courbe. Le problème rencontré avec le caoutchouc est que le matériau est trop rigide pour que le couple des servomoteurs puisse plier le caoutchouc. La force engendrée par la rigidité du matériau est un paramètre que nous n'avion pas pris en compte lors du dimensionnement des moteur. Ce qui fait que les SG90 sont sous dimensionnés pour le caoutchouc que nous avions commandé.

Nous devons alors trouver un moyen de rendre le caoutchouc plus souple pour que les servomoteurs puissent créer une ondulation.

Pour rendre le caoutchouc plus souple, le moyen le plus simple et rapide était d'entailler le caoutchouc. 3 types d'entaille sont alors envisageable :

- L'entaille sur la largeur : ceci permettrait de rendre la pale plus souple sur la déformation des ondulations sans perdre la rigidité sur l'axe des appuis véhicule/sol.
- L'entaille sur la longueur : non envisageable car l'opposé de la précédente donc inutile dans notre utilisation.
- L'entaille en diagonale : cela peut être combiné à la première proposition pour améliorer encore plus la souplesse du caoutchouc sur des ondulations si celle-ci n'est pas suffisante avec seulement les entailles sur la largeur. Mais attention les entailles en diagonale vont aussi faire perdre de la rigidité au caoutchouc sur l'axe des appuis véhicule/sol.

Nous avons réalisé 2 types de pales : une avec seulement des entailles sur la largeur et une avec des entailles sur la largeur et en diagonale.



Plusieurs remarques sont alors observées lors des tests de ces 2 types de pale.

- La pâle avec les entailles sur la largeur et en diagonale possède une meilleure déformation pour les ondulations.
- Les ondulations sont très approximatives car la déformation du caoutchouc au milieux de la pâle demande trop de force aux servomoteurs. Ils n'arrivent pas à réaliser une ondulation maximale contrairement aux servomoteurs en extrémité de la pale.

Au niveau du type de pale, la pale n°2 semble la plus approprié au niveau de la souplesse de la pale. Mais au niveau des servomoteurs du milieu de la pale, il reste compliqué pour les servomoteurs d'aller à l'angle demandé car le caoutchouc impose encore trop de résistance au servomoteur. Pour empêcher cette résistance sur les servomoteurs, il faut essayer d'obtenir plus de liberté au niveau des servomoteurs.

Pour cela 2 solutions sont envisagées :

- Retirer de la matière entre les accroches des servomoteurs pour laisser plus de liberté aux servomoteurs.

- Couper complètement la liaison entre tous les servomoteurs. Pour cela partir plus sur un système de pagaie d'aviron.

Comme nous voulons rester sur un système de pale nous avons pris la première solution, de plus cette solution fonctionne parfaitement et permet à tous les servomoteurs d'aller aux positions demandées.

Nous arrivons maintenant à avoir de très bonnes ondulations avec ce type de pale mais les servomoteurs restent encore sous dimensionné pour soulever le robot et réaliser les ondulations en même temps.

2.2. Électroniques embarquées - Shield pour Arduino

Suite à la réalisation de plusieurs tests pour le programme de contrôle au semestre 6, nous nous sommes rendus compte qu'il n'était pas pratique de travailler avec une *Breadboard* et des *Jumper* volants dans tous les sens. Pour des questions purement pratiques et esthétiques nous avons donc décidé de concevoir et fabriquer un *Shield* pour accueillir les connecteurs des servomoteurs et autres composants sur la carte Arduino. Ce prototype a été réalisé rapidement au début du semestre 8 pour pouvoir effectuer de nouveaux tests de déplacement.

La carte possède les caractéristiques suivantes :

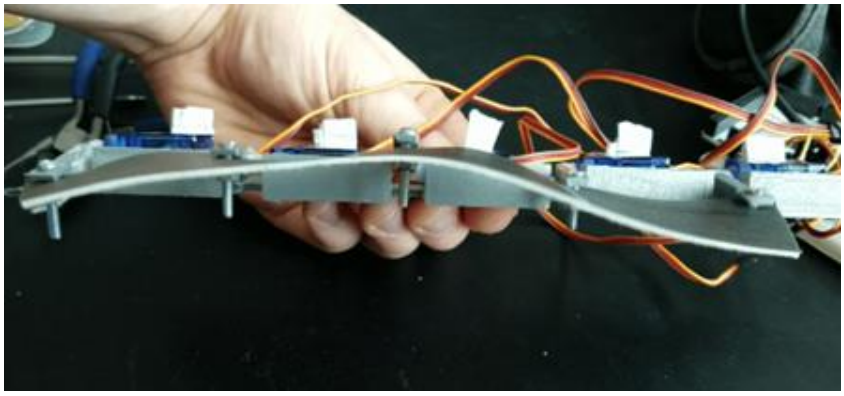
- 12 connecteurs pour servomoteur (GND + alimentation 5V + PWM)
- 8 emplacements pour LED de test (LED + résistance adéquat)
- Un connecteur pour l'alimentation générale de la carte (entrée 5V + GND)

Etant donné la simplicité du circuit (cf. Annexe 1) et le besoin pressant du *shield* pour réaliser des essais, ce dernier a été conçu avec le logiciel *Fritzing* puis imprimé directement avec la graveuse CNC de l'école.

2.3. Algorithme de propulsion

Semestres précédents

Au cours des deux derniers semestres, nous n'avons pu tester le déplacement qu'avec des pales en Lino. Un premier algorithme pour 5 servomoteurs à d'abord été testé avant de passe au modèle a 6 servomoteurs. Le principe de l'algorithme est de mettre en phase certain servomoteur, ici celui en position 1 avec le 3, celui en 2 avec le 4 et le 3 avec le 6, et de mettre un décalage entre chaque phase. Ici à l'initialisation nous mettons la première phase en position milieu, la deuxième phase à 66% de la position maximum haute et la troisième phase à 66% de la position maximum basse. Cette initialisation nous permet alors d'obtenir une vague avec 2 appuis au sol. Lors du fonctionnement de l'algorithme nous allons alors incrémenter ou décrémenter l'inclinaison des servomoteurs vers le haut ou vers le bas mais toujours en gardant les mêmes servomoteurs en phase et en gardant le même écart de phase entre chaque phase.



Ce premier test nous a permis de montrer que nous arrivons bien à créer une ondulation sur une pale en lino. Cette ondulation est encore une onde stationnaire et nous ne savions pas encore que cela ne fonctionnerait pas pour faire déplacer le robot.

Semestre 8

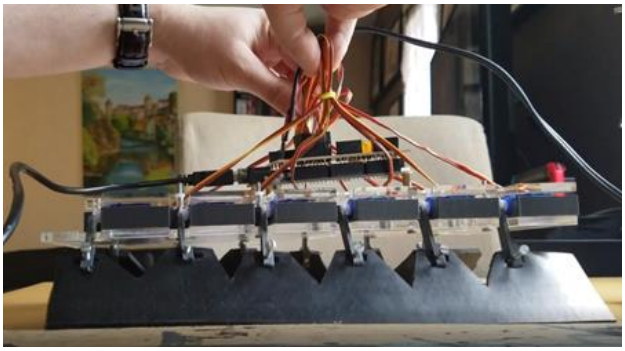
Le semestre 8 est le semestre où nous avons pu faire les premiers tests avec des pale en caoutchouc. Après avoir réalisé la conception des pales comme décrit dans la partie 2)1), nous avons directement essayé le code réalisé lors du semestre 7. Nous en avons déjà parlé précédemment mais le dimensionnement des servomoteurs étant trop faible, pour voir si l'algorithme réalisé fonctionne, nous devons soulever le robot à la main pour que moins de force soit appliqué sur les servomoteurs.

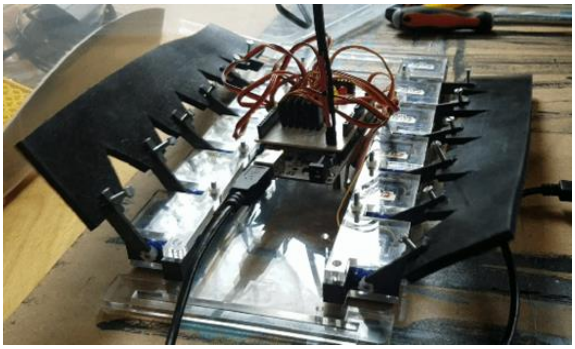
Malencontreusement l'algorithme crée lors des semestres précédents ne fonctionnait pas car l'algorithme crée une onde stationnaire. Une onde stationnaire ne peut pas permettre au robot d'être propulsé car aucune direction n'est donnée par l'ondulation d'une onde stationnaire. Nous devons alors réaliser un nouvel algorithme qui nous permettrait de créer une ondulation non stationnaire.

Vous pouvez alors trouver le code réalisé pour créer une onde non stationnaire sur le GitHub [7] dans Firmware/code_Arduino_S8/main.c.

Cet algorithme permet de créer une onde stationnaire. Le principe utilisé ici n'est plus de mettre des servomoteurs en phase mais de gérer seulement la vitesse d'ondulation d'un servomoteur et que les

servomoteurs qui suivent reproduise le même mouvement avec un retard. Nous pouvons alors paramétrer l'ondulation de la pale avec la vitesse d'ondulation du premier servomoteur, le retard entre chaque servomoteur et le temps entre chaque ondulation. Cela permet alors de choisir, à l'aide des paramètres, une ondulation non stationnaire si on ne prend pas des paramètres trop spécifiques et de ce fait réaliser une propulsion dans une direction pour le robot.





III. Navigation

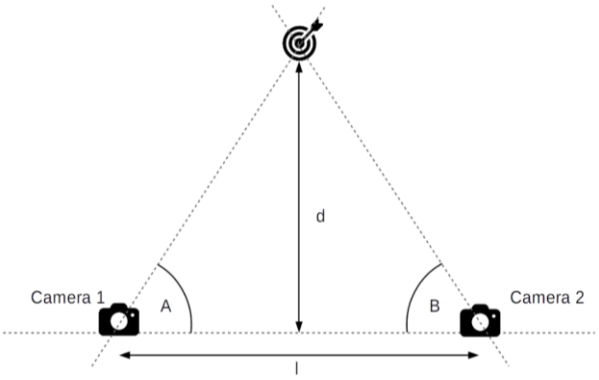
En plus du mode manuel régi par une télécommande (réalisation du semestre 7) nous souhaitions que le robot puisse se déplacer dans n'importe quel type d'environnement de manière autonome. Pour ce faire, nous avons besoin de collecter des informations précises, en temps réel, sur la nature de l'environnement avec des capteurs. Il nous faut plus précisément connaitre la position de la cible à atteindre et des différents obstacles rencontrés sur le parcours pour la rejoindre ainsi que la position du robot dans son environnement.

3.1. Stéréovision

La première solution envisagée pour analyser l'environnement a été d'utiliser la triangulation à partir des images : c'est ce qu'on appelle la Stéréovision. On utilise deux caméras séparées d'une certaine distance (comme deux yeux) pour mesurer la distance d'un objet. Les images permettent d'identifier des éléments précis que l'on peut caractériser d'obstacle ou de cible. A chaque point identifié on peut associer une distance par triangulation des données des deux images.

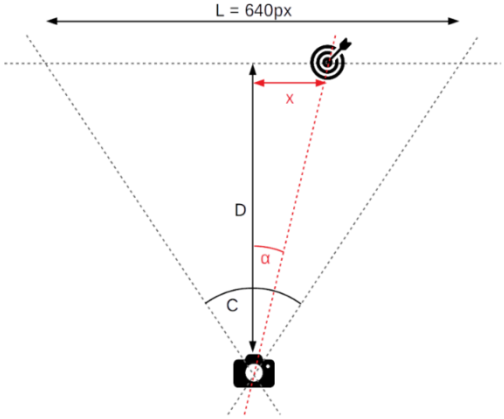
Principe et implémentation de la triangulation

Les deux caméras sont sur le même plan et ont la même inclinaison verticale/horizontale. On connaît la distance qui les sépare (l). Si on connaît les angles A et B alors on peut utiliser le principe de triangulation pour connaître la distance de l'objet (d).



- $l = \frac{d}{tan\,(A)} + \frac{d}{tantan\,(B)}$
- $d = l\,\frac{tan\,(A)tan\,(B)}{tantan\,(A) + tan\,(B)}$

On peut calculer les angles A et B car on connaît l'ouverture du champ de vision de la caméra (angle C) et on connaît également la largeur de l'image (L en pixel). On peut donc calculer la distance D et en déduire l'angle α.



- $D = \frac{\frac{L}{2}}{tan\,(\frac{C}{2})}$
- $tan\,(\propto) = \frac{x}{D} \Leftrightarrow \propto = arctan\,(\frac{x}{D})$
- $A = 90° - \propto$

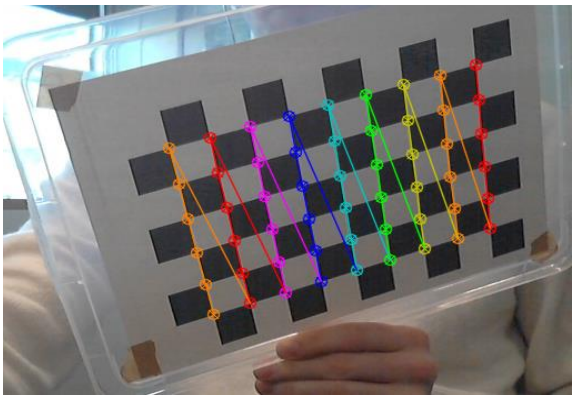
$$= 90° - arctan\,(\frac{x}{\frac{\frac{L}{2}}{tan\,(\frac{C}{2})}})$$

Il suffit désormais de trouver un point similaire dans chacune des images pour calculer sa distance par triangulation. A chaque point de l'image on peut désormais associer une notion de profondeur et donc on peut reconstituer un nuage de point 3D représentant l'environnement extérieur.

Pour trouver les similarités entre deux images, l'algorithme *Speeded Up Robust Features* (SURF) est utilisé. Nous n'avons pas eu à implémenter directement cet algorithme puisque la plupart des fonctions nécessaire au traitement des images et la construction du nuage de points sont présente dans la librairie OpenCV.

Calibrage des caméras

Pour que la triangulation fonctionne correctement, il faut traiter les images pour compenser la distorsion (radiale et tangentielle) introduite par la caméra (notamment à cause du positionnement de la lentille). Cette distorsion modifie les formes réelles des objets et donc influe sur le résultat de la triangulation. Pour rectifier cela, on prend une série de photo d'un damier avec différentes inclinaisons et on lance le programme de calibration (P13/StereoVision/CalibCam/calibRight.py). Ce dernier détecte les coins du damier et mesure les distances (en pixel) entre chacun. Ces dimensions sont comparées aux dimensions réelles et permette de calculer les coefficients de distorsion.



Limites

Cette technique possède des limites notamment en termes de précision puisque la qualité des mesures dépend fortement de l'environnement (luminosité, contraste, etc…). Le *Matcher* utilisé (StereoSGBM) est calibré par un certain nombre de paramètres déterminés expérimentalement (avec le programme P13/StereoVision/CalibSBM/calibReconstruction.py). Cependant, nous avons remarqué que selon l'environnement et la prise de vue des caméras, ces paramètres devaient constamment être redéfinis pour obtenir un résultat exploitable.

Nous rencontrons également un problème pour calculer la matrice de projection Q (cf. `P13/StereoVision/reconstruction.py`). Cette dernière est saisie manuellement car la fonction de calcul *stereoRectify* d'OpenCV ne semble pas fonctionner. Elle pourrait influencer la qualité du nuage de point.

La fiabilité des résultats obtenus par cette technique nous a poussé à nous tourner vers d'autres solutions. Un LIDAR avait initialement été prévu pour étalonner les distances mesurées par les caméras. Ce dernier possède une précision bien supérieure à la stéréovision et c'est pourquoi nous avons décidé de l'utiliser non plus comme simple étalon mais également comme outil de mesure principal pour l'observation lors de la cartographie.

3.2. Mapping – SLAM

L'observation de l'environnement est une chose, mais comment relier deux points similaires d'une position à l'autre ? La solution envisagée pour repérer le robot dans l'espace est de créer une cartographie de l'environnement extérieur. Si on connaît la position du robot, on peut aisément construire la carte à partir des données d'observation. Or dans des environnements contraignants, l'utilisation de systèmes de positionnement absolu (balises, GPS…) est impossible. De plus, si on connaît la cartographie, on peut calculer la position du robot. Dans notre cas, on ne connait ni l'un ni l'autre donc on utilise un algorithme SLAM (Simultanous Localization And Mapping).

SLAM basé sur un filtre de Kalman étendu

L'algorithme se base sur deux paramètres pour définir la carte. D'une part, on a les données d'observation qui sont, dans notre cas, fournies par le LIDAR. A chaque emplacement *t* on a donc un vecteur (z_t) de 360 valeurs qui associe à un angle donné, la distance de l'élément de l'environnement mesurée. D'autre part, on a les données de mouvement (*ut*) du robot. Ces dernières représentent une estimation du déplacement du robot mesuré par un codeur optique (*odometry motion modele*) ou un gyroscope (*velocity motion modele*). Dans notre cas, seul l'utilisation d'un gyroscope est envisageable puisque le robot ne possède pas de roues. Une fois ces deux informations recueillies, l'algorithme va estimer l'état du robot, à savoir la position et la cartographie :

$P(x_t, m \mid z_{1:t}, u_{1:t})$ avec x_t : la position du robot
m : la cartographie
$z_{1:t}$: Les données d'observation
$u_{1:t}$: Les données de mouvement

Cette estimation établie à l'aide d'un filtre de Bayes et plus précisément sur un filtre de Kalman (Kalman étendu puisque répartition non linéaire).

Implémentation

La plupart des recherches sur les moyens de mettre en place le SLAM nous ont mené à *ROS* (*Robot Operating System*). Ce dernier propose une implémentation toute faite de l'algorithme que l'on peut directement utiliser sur un jeu de données pré-enregistré. Cependant, ce système est lourd à mettre en place et ne nous permet pas de contrôler tous les aspects de son fonctionnement. Par exemple, le format de stockage des données du LIDAR est un format propre à *ROS* (.bag) et nous avions du mal à adapter le programme du LIDAR réalisé en amont aux contraintes de *ROS*.

Nous sommes donc passé à un système beaucoup plus transparent en exploitant le code de *PythonRobotics* [2]. Néanmoins, nous n'avions pas prévu de système de mesure du déplacement (codeur incrémental ou gyroscope) dans le cahier des charges et le confinement nous empêchait d'en commander un. Il fallait donc trouver un moyen de faire fonctionner l'algorithme sans retour sur la position du robot (ut). Là encore, les recherches nous ont principalement conduits vers ROS avec le module *GoogleCartographer* [3]. Finalement, la solution optimale que nous avons adoptée fut de reprendre des fonctions du *BreezSLAM* [4] trouvées sur le dépôt git de *Simon D. Levy*. Ce dernier propose une implémentation en python adaptée à nos besoins et nos contraintes (sans odométrie).

Limites

Dans les scénarios d'usage envisagés, seules des méthodes de positionnement relatives sont envisageables car la fiabilité des méthodes absolues (satellite, balise, …) n'est pas compatible avec la précision demandée. Cependant, elles produisent en général une dérive dans le positionnement du fait de l'accumulation de petites erreurs au fur et à mesure de l'investigation du robot. Cette

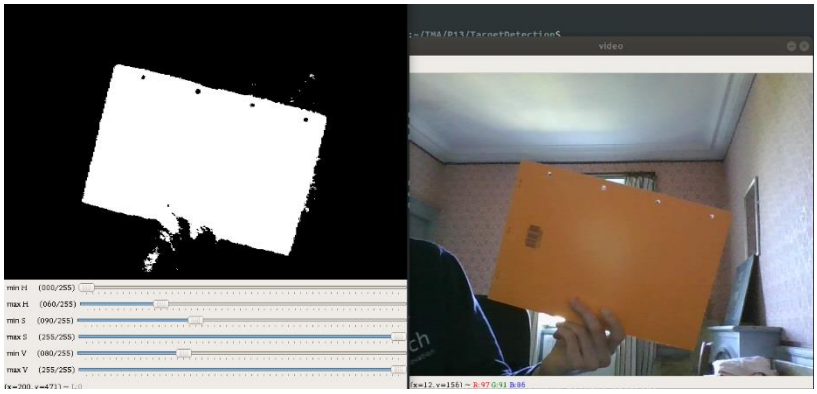
accumulation d'incertitude est principalement engendrée, dans notre cas, par le manque d'un système odométrique.

De plus, un autre problème majeur se pose dans notre cas ; La puissance de calcul demandé par l'algorithme SLAM peut parfois détériorer les performances de la carte *RaspberryPI* et ainsi pénaliser les autres programmes fonctionnants en même temps (cf. partie 3.4). Il en est de même pour l'autonomie du robot qui serait dégradée par la demande énergétique de la carte de contrôle.

3.3. Détection de la cible et calcul de trajectoire

Maintenant que le robot est capable de connaitre sa position et la cartographie de l'environnement (SLAM) ainsi que de mesure la distance qui le sépare d'un élément extérieur (Stéréovision), il faut qu'il puisse détecter sa cible et se déplacer dans sa direction. Nous avons envisagé deux solutions quant à la détection de la cible, toutes deux utilisant les images recueillies par la caméra.

L'idée ici est d'analyser les images produites par une des deux caméras et détecter la cible par sa couleur. Cette méthode ne fonctionne que si l'objet choisi possède une couleur très caractéristique (exemple : vert) qui la distingue des autres éléments de l'environnement. Par exemple nous avons réalisé nos tests un objet de couleur orange :



Cette méthode a été développée principalement à but expérimental car il est inenvisageable, dans un réel scénario, d'appréhender la cible uniquement par sa couleur.

La deuxième solution envisagée est de faire de la reconnaissance d'objet par apprentissage. Cela consiste à entraîner un modèle avec un jeu de données pour lui apprendre à reconnaître la cible.

Une fois la cible repérée, on peut aisément repérer son centre en effectuant la moyenne des coordonnées des pixels sélectionnés. On peut également mesurer la profondeur de ce point central en utilisant la stéréovision comme expliqué dans la partie 3.1. Nous connaissons donc la position du robot (grâce au SLAM) ainsi que celle de la cible, il ne nous reste plus qu'à calculer le meilleur itinéraire pour l'atteindre avec l'algorithme a*.

3.4 Architecture globale

Au démarrage, l'utilisateur a le choix entre commande manuelle et automatique (Cf Annexe 2 : Schéma architecture). Dans le cas d'une commande manuelle, le programme lit en boucle les instructions de la télécommande sur le serveur web de la Raspberry et les envoie par liaison série à la carte Arduino pour effectuer le déplacement.

Pour le déplacement autonome, le robot va d'abord explorer son environnement et constituer une cartographie avec SLAM. L'image issue d'une caméra va, elle, permettre de déterminer si la cible se

situe dans l'environnement (visible par le robot). Si la cible n'est pas détectée, le robot se déplace et recommence le même procédé d'analyse. Si la cible est détectée, on utilise alors la stéréovision pour récupérer la distance à laquelle se trouve l'objet. On peut ensuite déterminer la trajectoire à suivre pour atteindre l'objet. Le robot se déplace ainsi jusqu'à atteindre sa cible.

Dans un souci de temps et de ressources, nous n'avons pas pu développer entièrement cette architecture. En effet, les différentes parties sont fonctionnelles indépendamment les unes des autres. Il faudrait créer un programme général qui régit les actions du robot en faisant appel aux différentes fonctions (stéréovision, SLAM, trajectoire, …). De plus, il faudrait pouvoir exploiter d'avantage la cartographie dans le calcul de trajectoire ainsi qu'établir un protocole de communication entre la RaspberryPI et l'Arduino pour l'envoi des commandes de déplacement.

IV. Perspective : Limites et améliorations

La majorité des difficultés rencontrées viennent du fait qu'il s'agit d'un projet de recherche et que nous devions donc explorer plusieurs solutions avant d'aboutir à un résultat. Contrairement à la plupart des projets proposés, nous sommes parties de zéro et il a fallu construire brique par brique le projet en s'inspirant de plusieurs travaux trouvés sur internet.

Nous avons également rencontré des difficultés du point de vue des ressources et du temps à notre disposition. A titre comparatif, l'entreprise qui travaille sur le même type de robot (Pliant Energy System) n'en n'est qu'au stade de prototype alors qu'elle compte plusieurs dizaines d'employés.

Concernant le travail fournis dans sa globalité, nous pouvons dire que le système a été sous-dimensionné d'un point de vue matériel, rendant la tâche plus compliquée concernant la conception de la propulsion notamment. Cette partie requiert des compétences en mécanique qui dépassent le spectre de la formation IMA, ce qui nous a fortement limité. Il aurait été préférable de scinder le projet entre les deux spécialités (IMA-CM) afin d'avoir plus de temps pour développer la partie navigation et communication du projet.

Nous nous étions également fixés pour objectif de réaliser un robot low-cost et d'utiliser des ressources open-source. Cela a été respecté mais nous a aussi contraint d'utiliser du matériel moins performant. Par exemple, nous aurions pu d'emblée choisir de travailler avec une caméra 3D ou une *Kinect* au lieu de développer notre propre système de stéréovision.

Enfin, faute de support mobile, nous n'avons pas eu l'occasion de tester le système de navigation dans sa globalité. De même, nous n'avons pu tester le déplacement du robot que dans un seul type d'environnement.

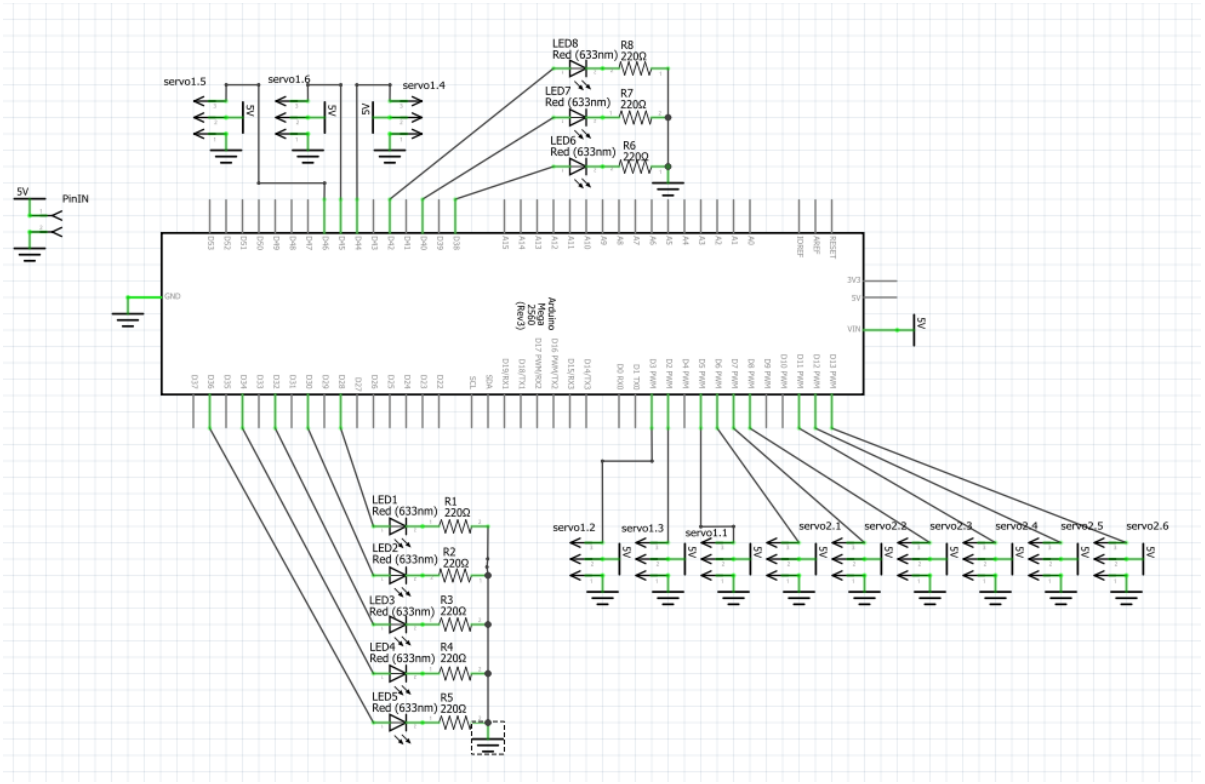
Malgré les problèmes rencontrés, nous pensons qu'une année de plus nous aurait permis d'aboutir à un prototype fonctionnel.

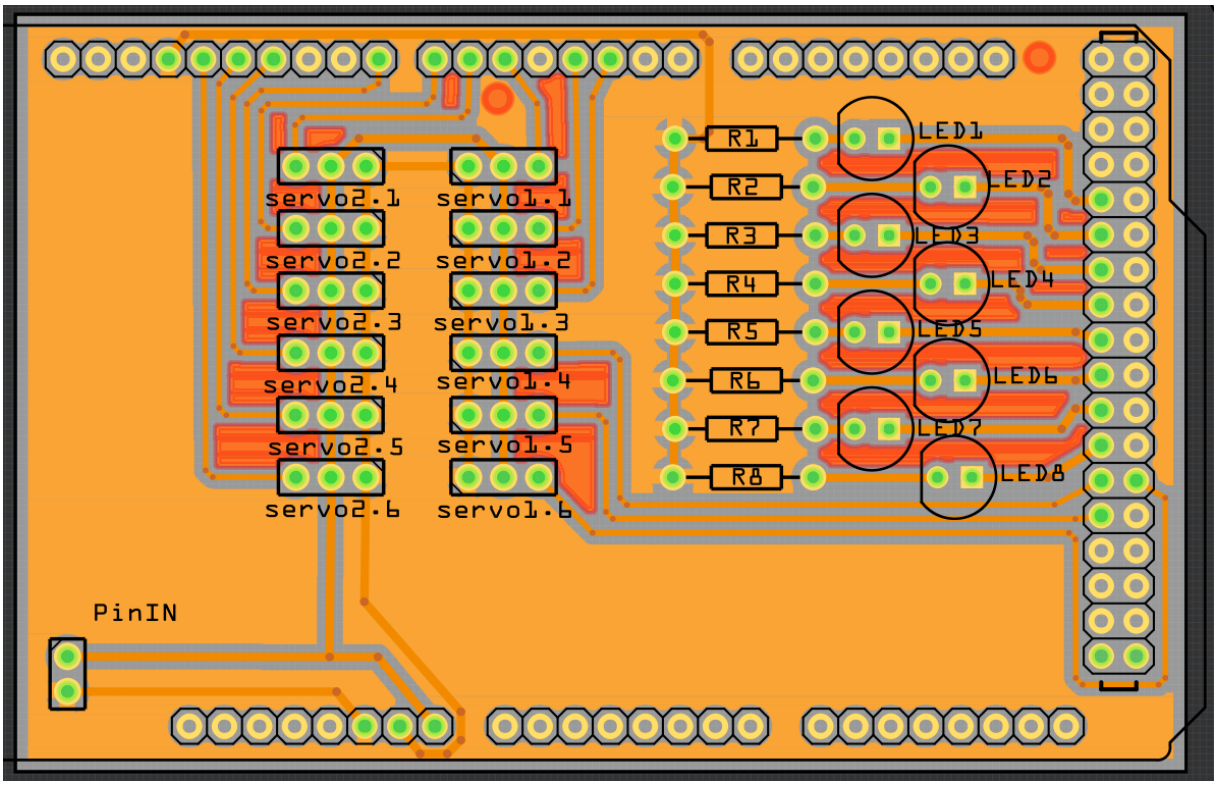
Conclusion

Au cours de ce dernier semestre, nous sommes passés à une phase de réalisation plus concrète du projet, comparé aux semestres précédents. Ces derniers nous ont permis d'exploiter différentes solutions et d'en retenir les meilleures concernant la propulsion du robot. Nous avons par exemple pu tester différents matériaux, dimensions et algorithmes pour les nageoires et comparer les résultats afin d'en tirer le meilleur compromis. Concernant la navigation, nous avons exploité différentes solutions techniques pour répondre à la problématique du déplacement autonome. Il s'est avéré que la solution optimale combinait plusieurs technologies.

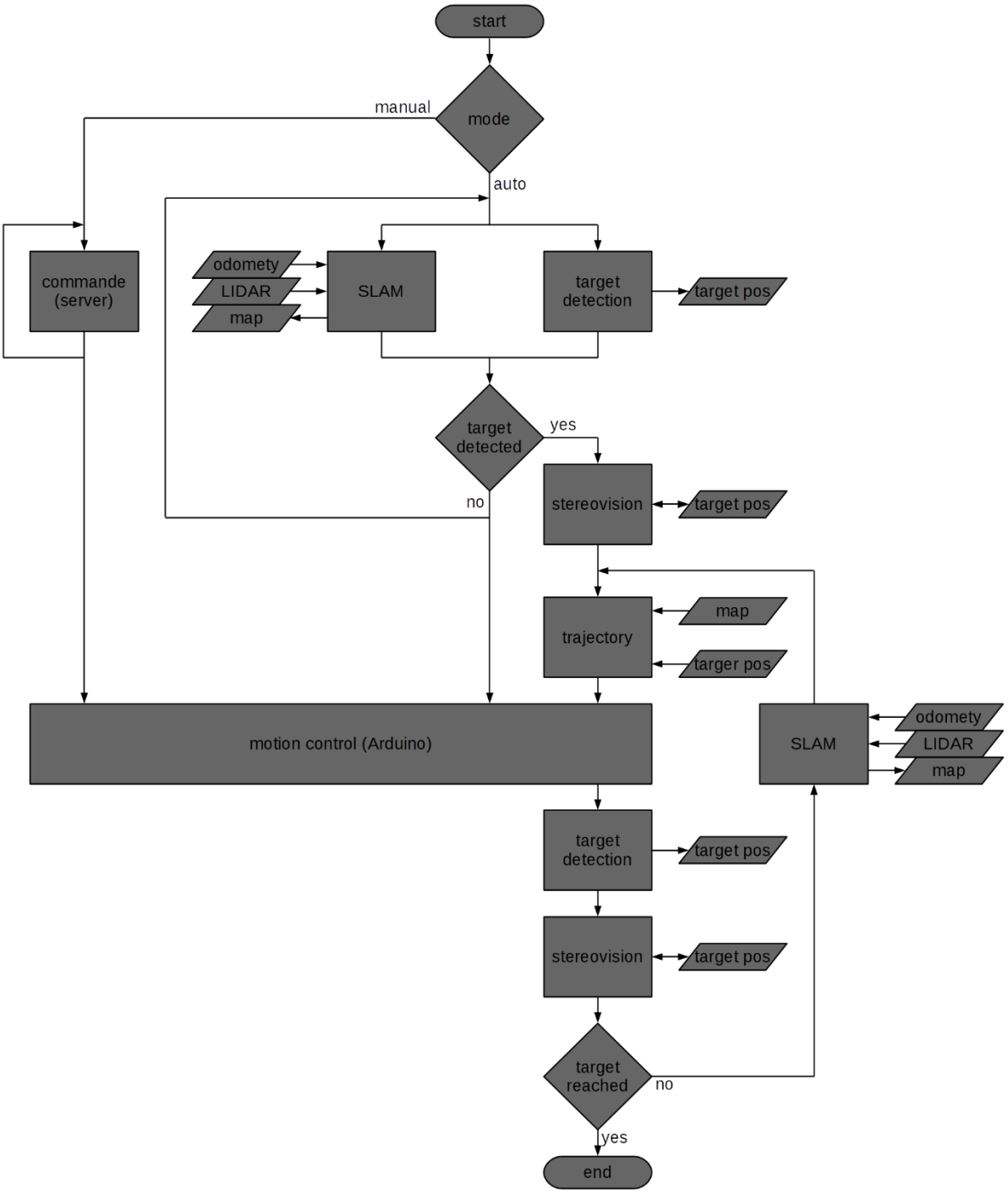
Dans sa globalité, le projet nous a permis de découvrir et de nous perfectionner dans des domaines que nous n'aurions pas eu l'occasion de voir en classe. Malgré les ressources matérielles et humaines à notre disposition, nous sommes plutôt fiers du travail réalisé et souhaitons que le projet soit repris par les futurs promotions.

Annexe 1 – Shield Arduino





Annexe 2 – Schéma architecture



Références

[1] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

[2] https://github.com/AtsushiSakai/PythonRobotics/tree/master/SLAM

[3] https://google-cartographer-ros.readthedocs.io/en/latest/

[4] https://github.com/simondlevy/BreezySLAM

[5] https://www.researchgate.net/publication/260634709_Salamandra_Robotica_II_An_Amphibious_Robot_to_Study_Salamander-Like_Swimming_and_Walking_Gaits

[6] https://www.pliantenergy.com/home-1

[7] https://github.com/vincentduboisdlc/P13