ECAM Robotic

Eurobot 2016

Aiglon Benjamin, Ben Hammou Wassim, Didouh Mohamed, Leroy Alexis, Linard Gauthier, Verfaille Guillaume

Table des matières

1. Introduction

Les machines envahissent de plus en plus la vie de tous les jours. Étant tous curieux et avides d'aventures, nous sommes une petite équipe de 6 personnes ayant décidé de participer au concours Eurobot et à sa première étape, la coupe de Belgique de robotique. Cette année, nous préparons notre canne à pêche, un seau pour construire des châteaux de sable, un essuie de bain et surtout de quoi se protéger du soleil… Le thème, la plage. Notre but, 7 mois pour réaliser un robot autonome.

Aller c'est parti, ne perdons pas de temps. Pour alléger un peu notre tâche, nous avons récupéré le robot de l'année précédente afin de disposer d'une base matérielle correcte. Après avoir pris connaissance de l'état du robot et suite à plusieurs réunions de groupe, nous nous sommes fixés de :

- ➔ Recâbler le robot
- ➔ Améliorer sa régulation
- ➔ Mettre en place en mécanisme pour attraper des poissons
- ➔ Réaliser un montage capable d'ouvrir un parasol

Après de longues heures de travail et de périlleux périples, nous avons réussi à finaliser notre robot. Il a ainsi été homologué au concours et nous avons pu participer au 5 matchs prévus nous permettant de totaliser 224 points. Une première pour l'ECAM !

2. Gestion de projet

Afin de gérer les différentes tâches, échéances et délivrables relatifs à notre projet, la gestion de l'avancement du robot s'est fait sur base de la méthode agile SCRUM.
Il s'agit d'une méthodologie qui consiste à découper le projet en différentes tâches et à établir un planning selon la disponibilité et l'expérience des membres de l'équipe.

Pendant la totalité du projet, l'équipe est solidaire, s'entraide et chacun expose son travail et les difficultés qu'il a rencontré lors de "stand up meetings" quotidiens animés par un SCRUM master. C'est également lors de ceux-ci que les tâches sont mises à jour et que l'équipe définit de nouveaux objectifs sous forme de :

- Tickets : objectifs à court terme réalisés par 1 ou 2 membres de l'équipe et pour lesquels on mentionne la tâche, la durée de celle-ci et les membres qui la réalisent

- Délivrables : objectifs intermédiaires sur du plus long terme

En procédant de la sorte, l'ensemble de l'équipe se fixe des objectifs à atteindre et est apte à prendre plus rapidement des mesures en cas de retard. D'autre part, les "stand up meetings" permettent à tous les membres de l'équipe de rester informés de l'avancement

des tâches qui ne sont pas les leurs et de faire bénéficier les autres de leurs connaissances en cas de problèmes relatifs à celles-ci.

3. Restructuration du projet Git

Un des problèmes que nous avons rencontrés lors de la lecture des codes réalisés les années précédentes est la compréhension du programme. Il n'y avait, en effet, presque pas de commentaires. Les tests réalisés et proposés par les élèves des années précédentes ne fonctionnaient pas ou n'étaient pas complets. Nous avons donc perdu un temps non négligeable en essayant de comprendre leur code.

Nous avons voulu simplifier la lecture et la compréhension du programme pour les années suivantes. Nous donc créé une documentation afin de guider ces nouveaux utilisateurs dans le dossier et dans les différents fichiers de programmation. Cette documentation permet à l'utilisateur d'avoir une idée globale et claire du programme principal. Il peut ainsi déjà avoir un aperçu du fonctionnement général du robot tout en pouvant faire rouler le robot avec le débogage afin de visualiser par où passe le script.

Les dossiers ainsi que les fichiers ont également été réarrangés afin d'obtenir un dossier racine le plus propre possible. Seulement l'essentiel y réside, sa compréhension est donc augmentée et l'utilisateur ne s'y perd plus. Les dossiers qui s'y trouvent sont les suivants

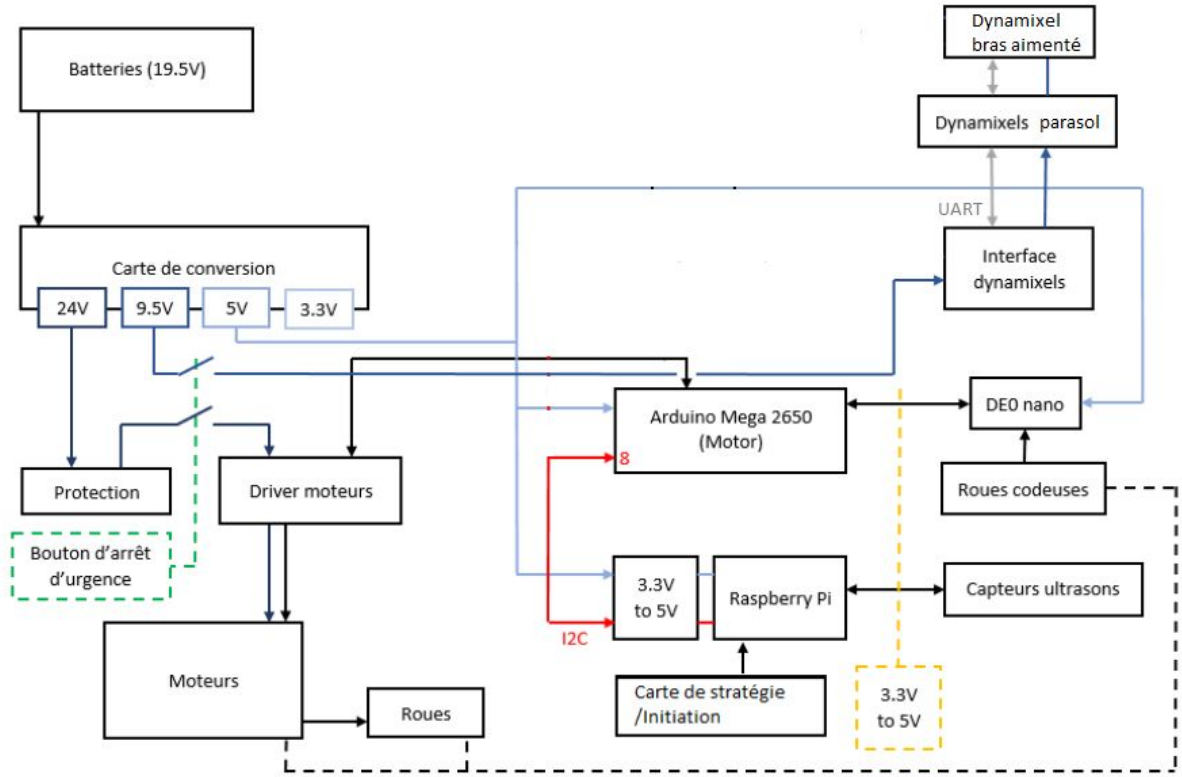
- **DEO Nano** : contient les codes de la DEO Nano
- **Main** : reprenant les codes qui nous ont permis de participer au Concours Eurobot 2016
- **Tests** : qui regroupe tous les tests que nous avons réalisés pour aboutir au code final. Chaque test comprend sa propre documentation afin de pouvoir les refaire
- **Tutoriels** : reprenant un bon nombre de tutoriels sur les différents éléments utilisés dans ce projet robot

Cette documentation est disponible sur le Git de l'ECAM =>
http://git.ecam.be/12073/ecam-robotic/tree/master

4. La structure hardware

Dans notre structure hardware, nous pouvons distinguer deux parties :

- L'électronique de puissance
- L'électronique embarquée



4.1. L'électronique de puissance

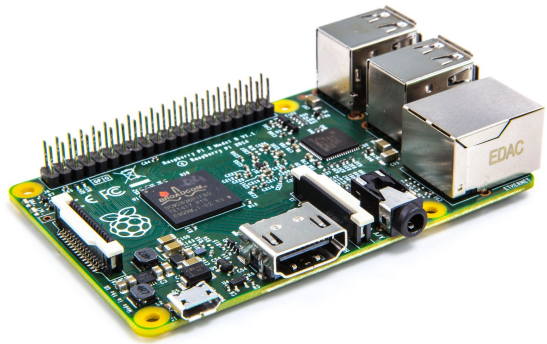
La carte d'alimentation est une carte qui a été récupérée des années précédentes, elle nous fournit 24, 9.5, 5 et 3.3 V à partir d'une batterie au lithium-fer de 19.5 V :

- 24 V pour alimentation des moteurs
- 9.5 V pour l'alimentation des Dynamixels
- 5 V pour les cartes de commande : Raspberry Pi, Arduino, DE0 nano, capteurs US
- 3.3 V pour la Raspberry et la DE0 nano qui envoient des signaux à cette tension

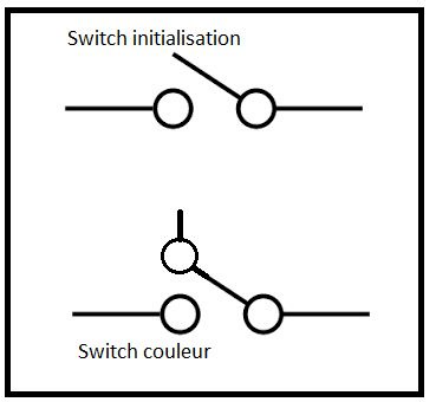
Le bouton d'arrêt d'urgence agit seulement sur l'alimentation des éléments mécaniques du robot (dynamixels et moteurs). Ainsi, nous évitons le reset complet en cas d'arrêt d'urgence.

4.2. L’électronique embarquée

4.2.1. Raspberry Pi 2

	La raspberry constitue notre carte principale qui joue le rôle d’intelligence principale de l’ensemble du robot. C’est elle qui gère toutes les autres cartes. En effet, elle communique en I2C avec les autres cartes du robot. C’est donc la Raspberry qui décide et détermine la séquence de déplacement et d’action à réaliser. Elle gère également les capteurs ultrasons.

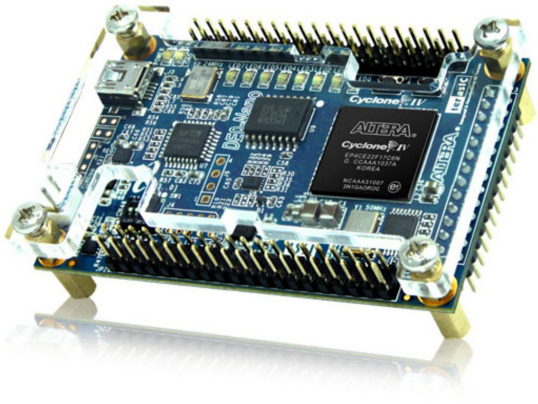
4.2.2. Carte de stratégie

	La carte de stratégie / initialisation permet d’initier le programme et d’activer le switch qui détermine dans quelle zone le robot joue (verte ou mauve) et ainsi sélectionner un des deux parcours possibles.

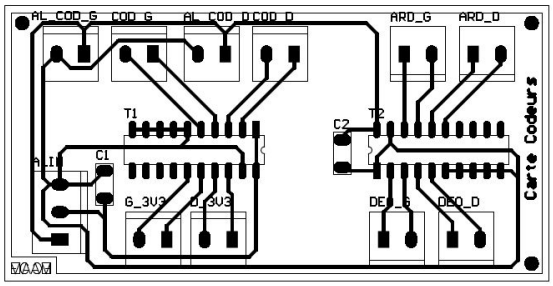
4.2.3. Arduino MEGA

	L’arduino Mega contrôle les moteurs et reçoit des informations de la part des roues codeuses. Le contrôle des moteurs se fait au moyen de signaux modulés PWM envoyés vers le driver moteur que nous verrons plus loin.

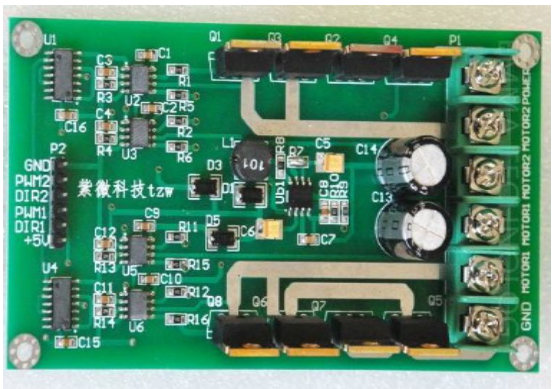
4.2.4. DE0-Nano (FPGA)

	Concernant les roues codeuses, ces dernières envoient beaucoup trop d'interruptions par seconde pour l'Arduino qui ne sait pas gérer une telle cadence de signaux. C'est pourquoi la DE0-nano est là pour permettre de diviser le nombre d'impulsions des roues codeuses par un rapport bien défini.

4.2.5. Carte relais de signaux

	Voici notre nouvelle rajoute. Il s'agit d'une petite carte simple qui permet de faire le relais des signaux entre les roues codeuses, la DE0-Nano et l'Arduino. Cette carte contient deux convertisseurs de niveau logique alimentés respectivement en 5V et 3.3V. Selon une pin de direction sur le circuit intégré (74LVC245), nous pouvons alors transformer un signal logique dans le sens 3.3V > 5V pour aller vers l'Arduino et dans le sens 5V > 3.3V pour aller vers la DE0-Nano. L'atout qu'apporte la carte est ses connecteurs qui permettent une solidité de système de câblage (cf. Fonctionnement des roues codeuses).

4.2.6. Driver moteur

	Le driver moteur se compose de transistors de puissances permettant la conversion de signaux de commande en signaux de puissance pour faire tourner les moteurs. Cette carte, en provenance de Chine, est protégée contre les surtensions et les surintensités grâce notamment à des diodes de roues libres intrinsèques aux transistors.

5. Structure software

5.1. Raspberry Pi

Les codes qui ont permis de participer au concours Eurobot sont stockés sur le Git de l'ECAM dans le projet **ECAM Robotic**.
Nous allons vous expliquer comment fonctionne globalement le robot. Si vous voulez plus de détails, nous vous conseillons de lire la documentation des différents test qui ont été réalisés. Ces tests se trouvent ici.

5.1.1. Le déroulement du programme

Le programme s'exécute en 3 étapes :

1. Initialisation des GPIOs, de la couleur et des capteurs US
2. Attend que la ficelle soit enlevée
3. Dès que la ficelle est enlevée, si la stratégie sélectionnée est
    - La stratégie de l'homologation exécute simplement les actions les unes après les autres. Il n'y a pas de timer. Cette stratégie a été utilisée à l'homologation du robot.
    - Sinon, lance un processus secondaire qui exécutera toutes les actions demandées pour le match. Le programme principal quant à lui attend la fin du match (90sec). Une fois les 90 secondes atteintes, arrêt des moteurs et exécution de la Funny Action (le parasol).

5.1.2. Les différents fichiers

- **`debug`** : permet d'afficher des messages à l'écran si le paramètre debug est mis à True
- **`I2CManager:`** gère la communication i2c entre la Pi et l'Arduino. Pour envoyer des données, il suffit d'exécuter la commande `send(action, data)`. Nous n'avons pas eut besoin de la méthode **`read()`** mais elle permet d'aller demander des informations à l'Arduino. Il suffit ensuite de lire les bytes renvoyés grâce à la méthode **`unPack()`**.
- **`initialisation`**: initialise tous les GPIOs ainsi que la couleur du match
- **`main:`** contient le programme principal
- **`myRobot:`** contient l'ensemble des actions que le robot peut réaliser
- **`param:`** regroupe tous les paramètres généraux du programme utilisé dans les différents fichiers
- **`us`**: gère les capteurs ultra-son

5.2. Arduino MEGA

Toutes les actions que doit gérer l'Arduino MEGA sont implémenté dans un seul fichier dans laquelle on importe la librairie dynamixel :

- **`Intégration.ino`** : Réceptionne par I2C, envoie les commandes aux moteurs et aux dynamixels
- **`DynamixelSerial.h`** : Librairie de toutes les fonctions du Dynamixels. Cette librairie (open source) contient donc toute la routine de communication UART entre l'Arduino et le Dynamixel.

6. Régulation

6.1. Fonctionnement

Pour pouvoir réaliser des actions, il faut évidemment que le robot ait une notion de distance. Pour se faire, il faut utiliser la régulation afin de pouvoir réaliser des distances d'avance ou des angles de rotations, et cela de façon précise. L'outil indispensable est la paire de roues codeuses (une pour la roue de gauche et une pour la roue de droite). Chaque roue codeuse est reliée à un codeur incrémental (ITD 01 B14) qui va transmettre les impulsions à la DE0-Nano. Cette dernière, après un traitement de ces informations pourra communiquer le sens et la cadence des tics d'avance à l'Arduino MEGA (moteur). L'Arduino pourra alors s'occuper de réguler l'avance et la rotation du robot sur base de ces informations.

6.2. Roues codeuses

Comme cité au paragraphe précédent, nous avons travaillé avec des codeurs incrémentaux pour connaître l'état d'avance du robot. Ces informations sont cruciales pour la régulation. Nous avons donc pris soin de recâbler correctement les codeurs pour avoir quelque chose de solide et fiable. Voici les caractéristiques principales du codeur :

ITD 01 B14

ITD 01 B14 avec bride synchro

Points forts

- Mini codeur axe sortant ø4 mm
- Résolution max. 1024 impulsions/tour
- Détection optique
- Boîtier ø24 mm
- Signaux de sortie TTL ou HTL
- Sortie câble radiale ou axiale

Option

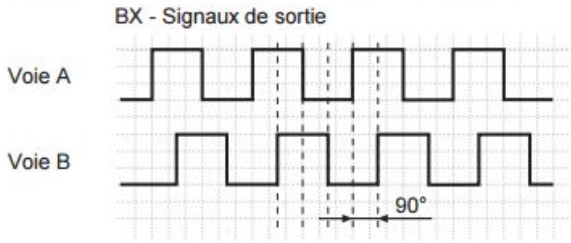
- Version avec connecteur au bout du câble

Alimenté en 5V, le codeur fonctionne sur base de deux signaux de sortie A et B qui selon l'avance de l'un ou l'autre signal en ce qui concerne le déphasage, permet de facilement savoir si le moteur concerné par le codeur tourne dans un sens ou dans l'autre. Voici, selon la datasheet, comment sont organisées les affectations des bornes.

ITD 01 B14

Signaux de sortie

Pour une rotation en sens horaire et vue côte montage.



Affectation des bornes

Signaux de sortie BX-/NX

Câble	Désignation
vert	Voie A
jaune	Voie B
gris	Voie 0
brun	+U alimentation
blanc	0 V alimentation
transparent	Blindage/boîtier

6.3. Traitement par le FPGA

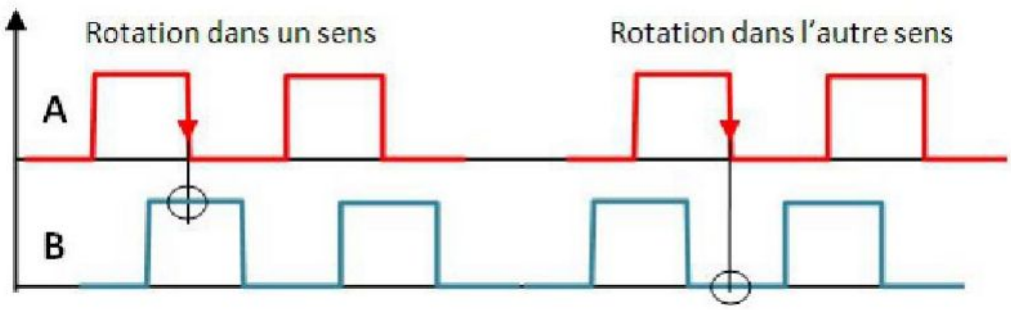
L'objectif de la DE0 nano est de récupérer les informations provenant des roues codeuses et de les transformer de manière à ce que l'arduino puisse les interpréter correctement. De plus, il faut veiller à ce que le nombre d'impulsions renvoyées vers l'arduino ne soit pas trop élevé pour ne pas dépasser la vitesse de traitement de l'arduino.

6.3.1. FSM d'une roue codeuse

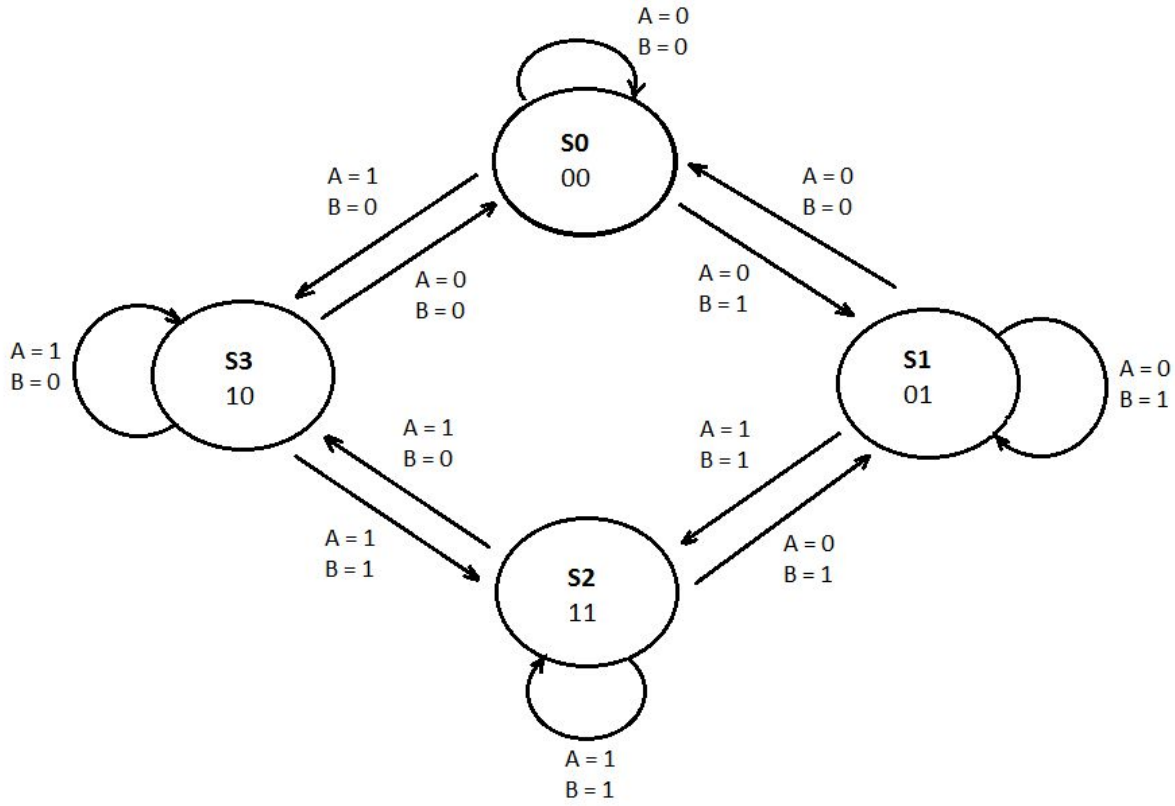
Le but de cette FSM est de renvoyer une impulsion lorsqu'une impulsion provient de la roue codeuse et le sens dans lequel va le robot. Pour détecter le sens de rotation, il suffit de travailler avec des états. Nous sommes en présence de quatre états qui s'enchaîneront d'en un sens ou dans l'autre selon la direction des roues, soit les états :

- S0 = 00
- S1 = 01
- S2 = 11
- S3 = 10

Soient les signaux A et B.



L'état de ces deux signaux fera donc l'objet de conditions de passage d'un état à un autre dans la FSM (cf. figure suivante)



Un fois que nous savons l'état dans lequel nous nous trouvons et l'état précédent, nous pouvons donc déterminer le sens de marche.

6.3.2. Diviseur de clock

Pour ne pas surcharger l'Arduino, le nombre d'impulsions en provenance des roues codeuses est réduit. Pour ce faire, il suffit de faire passer le signal d'impulsion en sortie des FSM (une par roue codeuse) dans un diviseur de clock. Celui-ci divise par un facteur 5 le nombre d'impulsions.

6.4. Système de régulation

C'est ici que vient l'essentiel. Une fois la cadence et le sens de marche connus, nous pouvons incrémenter ou décrémenter un compteur d'impulsion, respectivement pour une avance dans le sens horlogique de la roue et pour une avance dans le sens anti-horlogique. Ce compteur permettra ainsi de savoir quelle distance ou quel angle le robot a parcouru.

6.4.1. Initialisation de données constantes

```
// Roues codeuses
#define PI 3.14159265358979323846       // Nombre PI
#define Imp 409.6                       // nbr d'impulsions par tours
#define Rayon 4.02                      // rayon de la roue codeuse
```

6.4.2. Conversion consigne/nombre d'impulsion

Dans un premier temps, nous recevons une information de distance (en cm) ou d'angle (en degré) par communication I2C provenant de la raspberry. Pour effectuer la régulation et ainsi initialiser l'erreur d'avance ou de rotation au départ de l'action, il faut convertir la consigne en nombre d'impulsion :

```
/*
* Conversion d'une distance en cm en nombre d'impusion
*/
int DistanceToPulse(uint16_t data, float rayon)
{
   float perim = rayon * 10 * 2 * PI; // pémimètre de la roue codeuse
   float dist_imp = perim / Imp; // distance d'une impusion
   float dist = data * 10.0;
   int nbr_imp = (int) dist / dist_imp;
   return nbr_imp;
}

/*
* Conversion d'un angle en degré en cm en nombre d'impusion
*/
int AngleToPulse(uint16_t data, float rayon)
{
   float perim = rayon * 10 * 2 * PI; // pémimètre de la roue codeuse
   float dist_imp = perim / Imp; // distance d'une impusion
   float corr_angle = 107.5/360; // on a 107.5 pour 360° donc 107.5/360 pour 1°
   float angle = data * 10.0 * corr_angle;
   int nbr_imp_angle = (int) angle / dist_imp;
   return nbr_imp_angle;
}
```

Désormais, connaissant la consigne sous forme de nombre d'impulsion, nous pouvons initialiser la régulation en mettant à 0 tous les paramètres liés à celle-ci.

6.4.3. Régulation PI

Comme l'année dernière, nous travaillons avec une régulation de type Proportionnel Intégrale (PI). La commande de ce régulateur est proportionnelle à l'erreur, mais aussi proportionnelle à l'intégrale de l'erreur. Nous rajoutons donc à la commande générée par le régulateur proportionnel, la somme des erreurs commises au cours du temps.

```
commande =  Kp * erreur + Ki * somme_erreurs
```

avec

- Kp, le coefficient de proportionnalité de l'erreur.
- Ki, le coefficient de proportionnalité de la somme des erreurs.

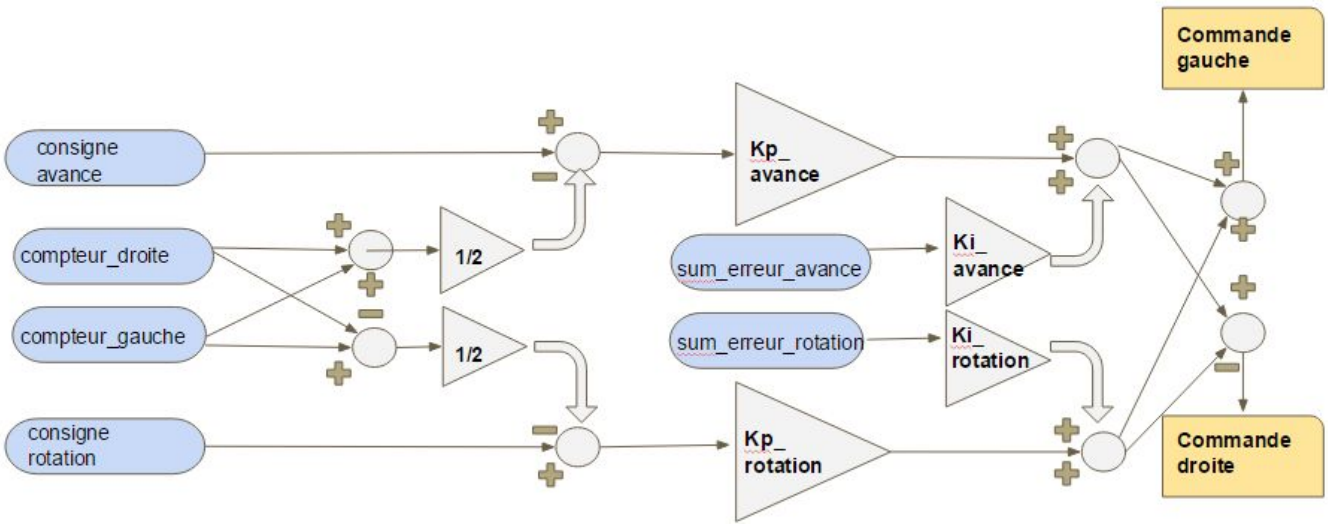
Le réglage des coefficients Kp, Ki d'un PI peut se faire manuellement par essais/erreurs. Voici la manière donc nous avons procédé pour régler ces paramètres :

- Tout d'abord, nous mettons en place un simple régulateur proportionnel (le coefficient Ki est donc nul). Par essais/erreurs, il faut régler le coefficient Kp afin d'améliorer le temps de réponse du système. C'est-à-dire qu'il faut trouver un Kp qui permette au système de se rapprocher très vite de la consigne tout en faisant attention de garder la stabilité du système. Autrement dit, il faut éviter des oscillations trop nombreuses autour de la consigne, due à un dépassement (cf. graphe ci-dessous)
- Une fois ce coefficient réglé, nous pouvons passer au coefficient Ki. Celui-là va permettre d'annuler l'erreur finale du système afin que celui-ci respecte exactement la consigne. Il faut donc régler Ki pour avoir une réponse exacte en peu de temps tout en essayant de minimiser les oscillations apportées par l'action intégrale.

Nous comprenons donc bien que le PI (voir PID) parfait n'existe pas.

6.4.4. Routine de régulation

Mettons en place le régulateur PI. Pour différencier une consigne d'avance ou de recule ainsi qu'une rotation horlogique ou anti-horlogique, nous utilisons respectivement des consignes positives ou négatives. Nous nous sommes inspiré de la routine utilisée l'année dernière suite à une longue réflexion. Voici le schéma bloc de régulation avec lequel nous avons travaillé pour gérer les commandes (gauche et droite) des moteurs.



Voici le code commenté qui s'exécute selon une cadence de 25ms.

```
/*
* Régulation PI (Proportionnelle - Intégrale)
*/
void regulate()
{
        DEBUG_PRINTLN("La regulation commence");
        // Récupération des commandes précédentes
        float prevCMDGauche = CMDGauche;
        float prevCMDDroit = CMDDroit;
        // ------------------------------------------
        // Calcul des erreurs d'avance et/ou rotation :
        // ------------------------------------------
        // Rotation
        ErreurRot = ConsigneRot - (compteurGauche - compteurDroit) / 2;
        if (ConsigneRot == 0 and abs(ErreurRot) > 200) // Si pour une AVANCE le nombre
        d'impulsion des compteur de gauche et de droite sont trop éloignés alors
        { // c'est que le robot dévie trop de sa trajectoire, il faut donc arrête
        d'urgence le moteur.
        EmergencyStop();
        }
        // Avance
        ErreurAvance = ConsigneAvance + ((compteurGauche) + (compteurDroit)) / 2;
        if (ConsigneAvance == 0 and abs(ErreurAvance) > 200) // Si pour une ROTATION le
        nombre d'impulsion des compteur de gauche et de droite sont trop éloignés
        (doivent normalement être opposé pour des nombre signés) alors c'est que le robot
        ne réalise pas une rotation assez "sur place", mais plutot en avançant ou en
        reculant, il faut donc arrête d'urgence le moteur.
        {
                EmergencyStop();
        }
        // ------------------------------------------------------------
        // Accumulation des erreurs d'avance ou rotation dans une somme :
        // ------------------------------------------------------------
        // Avance
        if (ErreurAvance < 300)
        {
                SumErreurAvance += ErreurAvance;
        }
        // Rotation
        if (ErreurRot < 200)
        {
                SumErreurRot += ErreurRot;
        }
        DEBUG_PRINT("ErreurRot : ");
        DEBUG_PRINTLN(ErreurRot);
        DEBUG_PRINT("ErreurAvance : ");
        DEBUG_PRINTLN(ErreurAvance);
        /*
        * Gain intermédiaire de l'action intégrale (I) sur l'accumulation des erreurs
        * -> On limite cette somme à une certaine valeur pour éviter que la régulation ne
        s'emballe (divergence)
        */
        RegulRot = kiRot * SumErreurRot;
        if (abs(RegulRot) > AntiEmballementRot)
        {
                if (RegulRot < 0)
                {
                        RegulRot = - AntiEmballementRot;
```

```
        }
        else
        {
                RegulRot = AntiEmballementRot;
        }
}
RegulAvance = kiAvance * SumErreurAvance;

if (abs(RegulAvance) > AntiEmballementAvance)
{
        if (RegulAvance < 0)
        {
                RegulAvance = - AntiEmballementAvance;
        }
        else
        {
                RegulAvance = AntiEmballementAvance;
        }
}
/*
* Gain intermédiaire de l'action proportionnel (P) sur l'accumulation des erreurs
*/
RegulRot += (kpRot * ErreurRot) + kdRot*(ErreurRot - prevErreurRot);
RegulAvance += (kpAvance * ErreurAvance) + kdAvance*(ErreurAvance -
prevErreurAvance);
/*
* Commande des moteurs gauches et droits pour ajuster l'avance et la rotation
régulée
*/
CMDDroit = RegulAvance - RegulRot;
CMDGauche = RegulAvance + RegulRot;
/*
* On empêche les commandes négatives
*/
if (CMDGauche < 0) // Le moteur gauche tourne en arrière
{
        CMDGauche = abs(CMDGauche);
        DIRG = false;
}
else
{
        DIRG = true; // Le moteur gauche tourne en avant
}
if (CMDDroit < 0) // Le moteur droit tourne en arrière
{
        CMDDroit = abs(CMDDroit);
        DIRD = false;
}
else
{
        DIRD = true; // Le moteur droit tourne en avant
}
/*
* Démarrage progressif des moteurs (On impose une augmentation de 3 du duty cycle
de la PWM pour les premières successions de commande de la régulation
*/
if ((CMDDroit - prevCMDDroit) > 3) CMDDroit = prevCMDDroit + 3;
if ((CMDGauche - prevCMDGauche) > 3) CMDGauche = prevCMDGauche + 3;
```

```
/*
* On empêche des commandes trop grande tout en n'empêchant pas la rotation
*/
int MaxSpeedGauche = 60;
int MaxSpeedDroit = 60;
if (deplacement == 0)
{
        MaxSpeedGauche = MaxSpeedAvance;
        MaxSpeedDroit = MaxSpeedAvance;
}
if (deplacement == 1)
{
        MaxSpeedGauche = MaxSpeedRot;
        MaxSpeedDroit = MaxSpeedRot;
}
/* Si on est en consigne d'avance, il faut autoriser des vitesses maximales plus
grande à gauche ou à droite
* au cas où les commandes à gauche et à droite sont trop distantes en valeur
l'une par rapport à l'autre
* dans ce cas, le robot a besoin de remettre sur la bonne trajectoire et pour
celà il faut soit autoriser une vitesse
* plus importante à gauche ou à droite en fonction de la déviation à rattraper.
*/
if (ConsigneRot == 0 and abs(CMDDroit - CMDGauche) > deltaCMD)
{
        if (CMDGauche > CMDDroit)
        {
                MaxSpeedGauche += 20;
        }
        else
        {
                MaxSpeedDroit += 20;
        }
}
// On plafonne les vitesses de commande
if (CMDDroit > MaxSpeedDroit)
{
        CMDDroit = MaxSpeedDroit;
}
if (CMDGauche > MaxSpeedGauche)
{
        CMDGauche = MaxSpeedGauche;
}
/*
* Si les erreurs sont assez faibles, c'est que le robot a réalisé la consigne
(distance ou rotation atteinte)
* Dans ce cas on prévient la RPi par la pin SUCCESS mise à l'état 1 et on attend
sa réponse pour réindiquer SUCCESS à 0,
* Ainsi la RPi peut lancer une nouvelle consigne
*/
if (abs(ErreurAvance) <= 15 and abs(ErreurRot) <= 15)
{
        if (!finished)
        {
                DEBUG_PRINTLN("SUCCES");
                digitalWrite(SUCCESS_PIN, 1);
                // On attend une réponse de la RPi
                while (true) if (digitalRead(RECOVER_PIN) == 1) break;
                digitalWrite(SUCCESS_PIN, 0);
                finished = true;
```

```
                    regulateInit(0.0, 0.0);
                    DEBUG_PRINTLN("Succes recover");
                }
                return;
        }
        // DEBUG_PRINTLN("Cmd gauche + dir : ");
        // DEBUG_PRINT(DIRG);
        // DEBUG_PRINT(" ; ");
        // DEBUG_PRINTLN(CMDGauche);
        // DEBUG_PRINTLN("Cmd droit + dir : ");
        // DEBUG_PRINT(DIRD);
        // DEBUG_PRINT(" ; ");
        // DEBUG_PRINTLN(CMDDroit);
        float prevErreurRot = ErreurRot;
        float prevErreurAvance = ErreurAvance;
        /*
        * Envoi des commandes moteurs
        */
        runMotors(DIRG, byte(CMDGauche), DIRD, byte(CMDDroit));
}
```

Comme nous pouvons le constater dans cette routine de régulation, nous pouvons passer en arrêt d'urgence au cas où une anomalie est repérée. De plus une fois, la consigne atteinte, nous lançons une communication via GPIO avec la Raspberry Pi. En effet, en plus de la communication I2C, 4 fils relient la Raspberry et l'arduino. Nous avons donc 4 signaux logiques (0 ou 1) que voici :

- *Success* : Détermine si la consigne est atteinte
- *Emergency* : Prévient de l'arrêt d'urgence lancée par l'Arduino
- *Recover* : Prévient de la bonne réception d'un message ou signal
- *Stop* : Stoppe les moteurs

En ce qui concerne la consigne atteinte, nous stoppons les moteurs. L'arduino met alors *Success* à l'état haut. La Raspberry, une fois au courant, répond en mettant *Recover* à l'état haut. C'est alors que l'Arduino peut réinitialiser les paramètres de régulations et attendre l'arrivée d'une nouvelle consigne.

7. Le bras à poissons

7.1. But

Une des actions qui étaient à réaliser cette année dans le cadre de nos matchs était d'être capable de pêcher des poissons. Pour cela, le règlement nous donnait les informations suivantes :

- Chaque équipe dispose de 4 poissons de sa couleur
- Les 4 poissons sont situés dans par 2 aquariums d'eau placés sur les côtés du plateau de jeu
- Chaque poisson possède des anneaux magnétisables
- Une fois pêchés, les poissons doivent être placés dans un filet situé à côté des aquariums

Autre information importante : saisir un poisson et l'extraire de l'eau rapporte 5 points tandis que le placé dans le filet rapporte 5 points de plus.

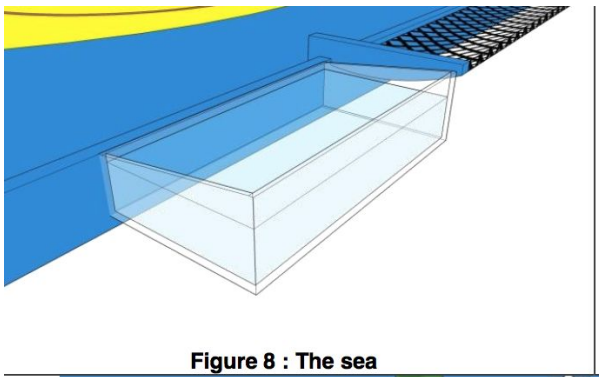


Figure 8 : The sea

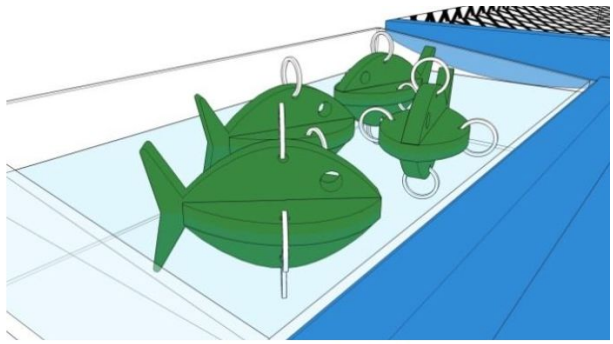
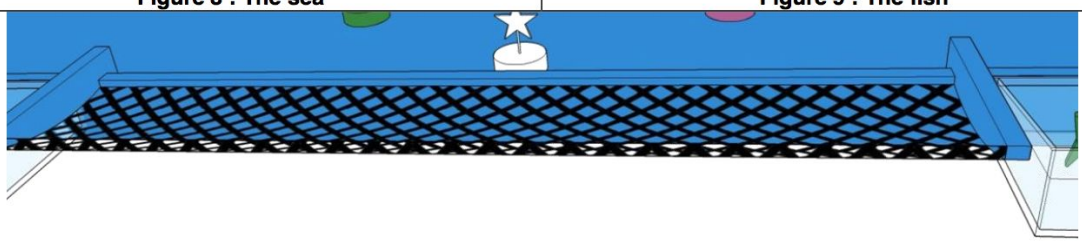


Figure 9 : The fish



7.2. Réflexion

Puisque le règlement de la compétition ne fixait aucune contrainte quant à la réalisation du bras à poisson, nous avons but laissé libre cours à notre imagination et tester plusieurs possibilités.

Dès le début, nous nous sommes mis d'accord sur l'utilisation d'aimants pour saisir les poissons en utilisant la propriété magnétisable de leurs anneaux.
Deux solutions étaient alors possibles :

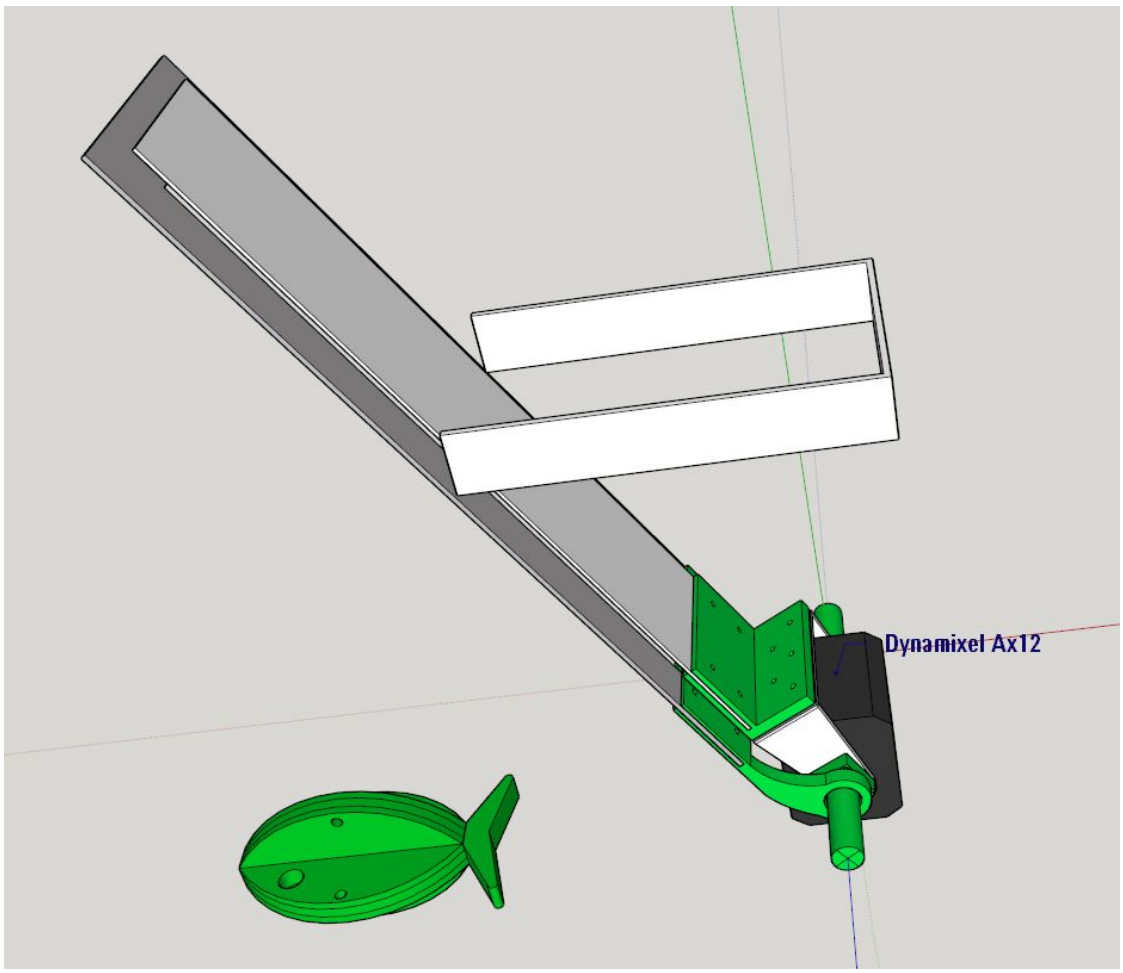
- Utilisation d’électroaimants
- Utilisation d’aimants permanents

Après réalisation d’un électroaimant “maison” dans le but de réaliser des tests, nous avons constaté qu’en utilisant cette méthode, le peu d’énergie dont nous disposions sur le robot ne nous permettrait pas d’obtenir facilement assez de puissance que pour procéder à la prise des poissons.

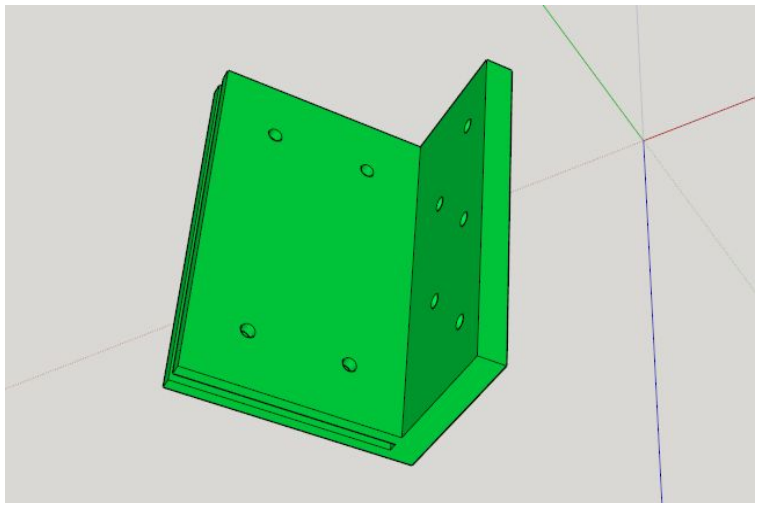
Du coup, nous nous sommes penchés sur l’autre solution, à savoir l’utilisation d’aimant permanent.

7.3. Mise en oeuvre

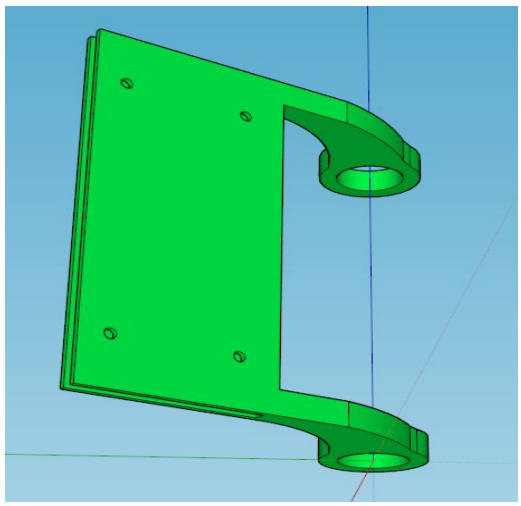
Une fois la solution des aimants permanents choisie, nous nous sommes penchés sur le mécanisme à mettre en place pour que les poissons puissent d’une part être saisis par le robot et d’autre part, relâchés dans le filet. Après réflexion, nous avons pu modéliser le système suivant :

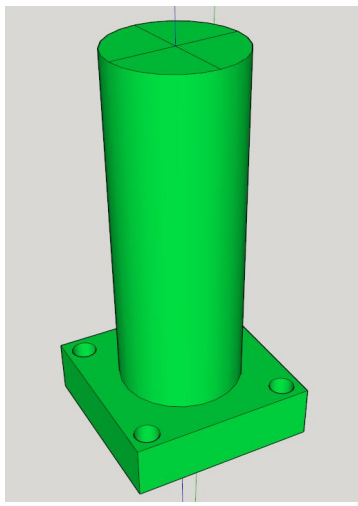


Le principe est le suivant :
Une première plaque aimantée (la rouge et bleu) est fixée sur un Dynamixel et est, pour débuter, en position verticale. En abaissant cette plaque à l'horizontale, celle-ci va venir aimanter les poissons au travers d'une seconde plaque située entre les aimants et les poissons. Une fois les poissons aimantés, le Dynamixel va se charger de remonter les 2 plaques dans une position intermédiaire pour se rendre jusqu'au filet. Une fois devant le filet, les 2 plaques remontent en position verticale et une petite pièce fixée au robot se charge de les séparer pour défaire l'aimantation et faire tomber les poissons dans le filet.



Support de la partie interne du bras qui permet de tenir la plaque aimantée.





La partie externe du bras est constituée d'un support pour le cadre métallique, et des extensions de l'arbre-moteur autour desquelles le support peut tourner.

Adaptation

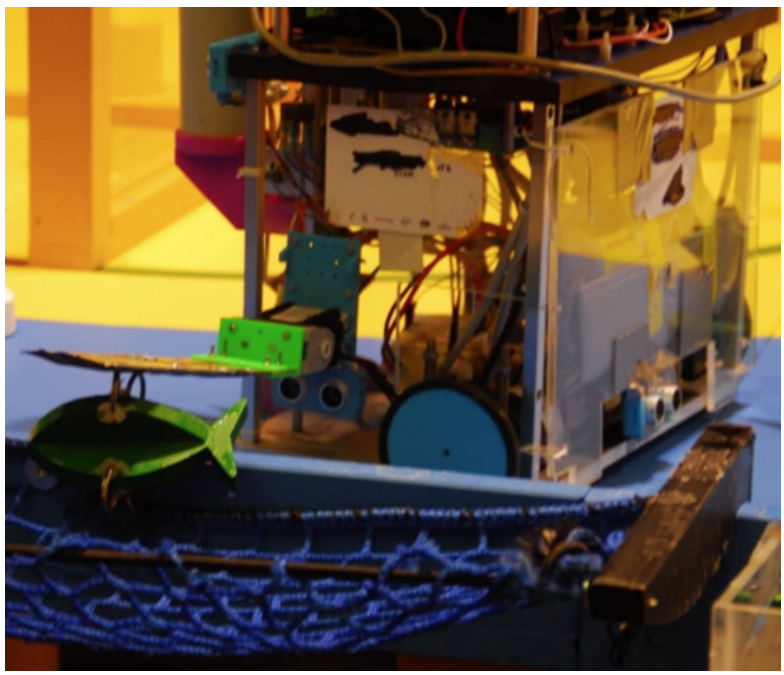
A partir de ce modèle théorique, nous avons commencé la réalisation des différentes pièces. En adaptant le système au fur et à mesure des séries de tests et des problèmes techniques

constatés, nous sommes parvenus à réaliser un bras capable de prendre jusqu'à 2 poissons en même temps.

Seulement, en arrivant au concours le jour J, nous nous sommes rendu compte de 2 problèmes :

- Selon le nombre de poissons attrapés par le dispositif, l'aimantation était plus ou moins forte et les 2 plaques avaient de temps en temps du mal à se détacher l'une de l'autre
- Le périmètre du robot déployé était 3 centimètres plus grands que celui autorisé et la validation statique n'était dès lors pas possible

Pour remédier à ce problème, nous avons décidé de retirer la plaque présente entre les aimants et les poissons et de diminuer la longueur de la plaque aimantée. Dès lors, le décrochage des poissons de la plaque aimanté a été réalisé en plaçant les poissons au-dessus du filet et en venant les cogner sur le bord du plateau à côté duquel se trouve le filet.



7.4. Résultat

Au final, le peu temps disponible entre les matchs de la compétition ainsi que les différents problèmes qu'il a fallu résoudre tout au long de celle-ci ont fait que nous n'avons pu envoyer notre robot vers le bac à poissons que lors de 2 matchs. Lors d'un de ceux-ci, un premier poisson a été saisi et relâché dans le filet alors que lors de l'autre, le robot a réussi à en mettre un dans le filet avant de faire un aller-retour pour aller en chercher un autre et le déposer dans le filet à son tour.

8. Le parasol

8.1. But

A la fin des 90 secondes du match, le robot dispose de 5 secondes pour ouvrir un parasol afin d'obtenir 20 points supplémentaires. Pendant une réunion avec le groupe, nous avons décidé de réaliser cette action étant donné qu'elle est "simple" et qu'elle rapporte beaucoup de points. Pour ce voir attribuer ces 20 points, il est nécessaire que le parasol sorte et qu'il s'ouvre. Le parasol doit être caché pendant toute la durée du match.

8.2. Mécanisme

Pour exécuter cette tâche, nous devions disposer d'un mécanisme capable d'effectuer une action de levage ainsi qu'une action d'ouverture assez rapidement afin de respecter les consignes données dans le règlement du concours.

Nous avons imaginé deux systèmes utilisant plus ou moins de ressources. Le premier utilisant un moteur pour faire l'action de levage et un petit ressort pour l'ouverture du parasol. Le second utilisant un ressort avec un système déclencheur pour permettre le levage et un ressort pour l'ouverture du parasol. Nous avons décidé de poursuivre avec le second mécanisme.
Pour réaliser ce mécanisme, voici le matériel utilisé:

- un tube en pvc d'un diamètre intérieur de 35mm
- un dynamixel AX12A
- un ressort de compression de bic
- un ressort de compression d'un diamètre inférieur à 35mm
- un parasol à cocktail
- une bague de fixation
- des fixations pour installer le mécanisme sur le robot

Malheureusement, après les tests nous nous sommes rendu compte que le mécanisme était trop puissant pour le parasol en papier. En effet, le ressort se décompressait beaucoup trop rapidement cassant le parasol.

Nous nous sommes donc repliés sur la première réflexion utilisant un "ascenseur". Ce mécanisme est composé de :

- un tube en PVC de diamètre intérieur 45mm
- un ressort afin d'ouvrir le parasol
- un parasol à cocktail
- une bague de fixation
- une barre métallique composée d'une courroie (nous l'appellerons le "rail")
- un dynamixel faisant office de moteur
- deux interrupteurs de fin de course (un en haut du tube et l'autre en bas du tube)

8.3. Fonctionnement

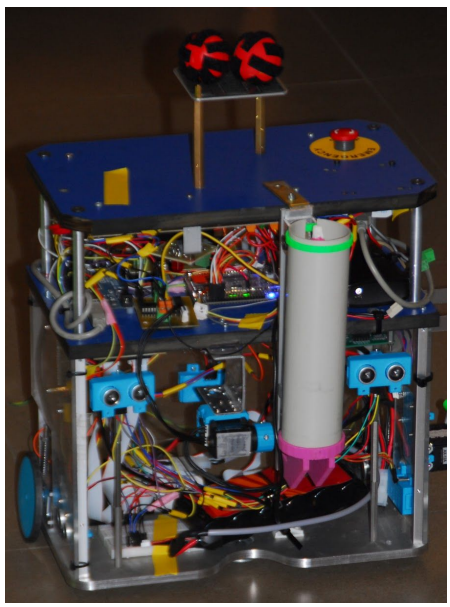
Sur la partie haute du robot, nous allons placer un tube qui contiendra la quasi-totalité du mécanisme. Le parasol, fixé sur le rail grâce au support du tube, contient une fente qui est maintenue ouverte par la cale. Avant de fermer la fente, nous avons placé le support du parasol dans le tube. Ce support du parasol est alors attaché à la courroie en pinçant cette courroie entre les deux plaquettes, permettant alors au support du parasol d'être entraîné par le dynamixel et de parcourir tout le tube. Nous avons alors placé le parasol dans sa bague en n'oubliant pas de placer le petit ressort qui permet l'ouverture du parasol. Pour finir l'assemblage, nous insérons la bague du parasol dans le tube pour qu'il s'imbrique avec le support du parasol et nous plaçons tous le mécanisme à son état de repos (c'est à dire contre l'interrupteur de fin de course du bas).

Après les 90 secondes du match, le dynamixel commence à tourner, ce qui fait monter le support, la bague et le parasol au moyen de la courroie. Une fois hors du tube, le parasol va alors s'ouvrir au moyen du petit ressort et le dynamixel est arrêté par l'interrupteur de fin de course du haut.

Pour remettre le mécanisme dans son état de repos, nous déconnectons le dynamixel. Ensuite nous faisons manuellement redescendre le support, la bague et la parasol, en faisant bien attention à bien replier le parasol pour qu'il rentre dans le tube.

8.4. Intégration sur le robot

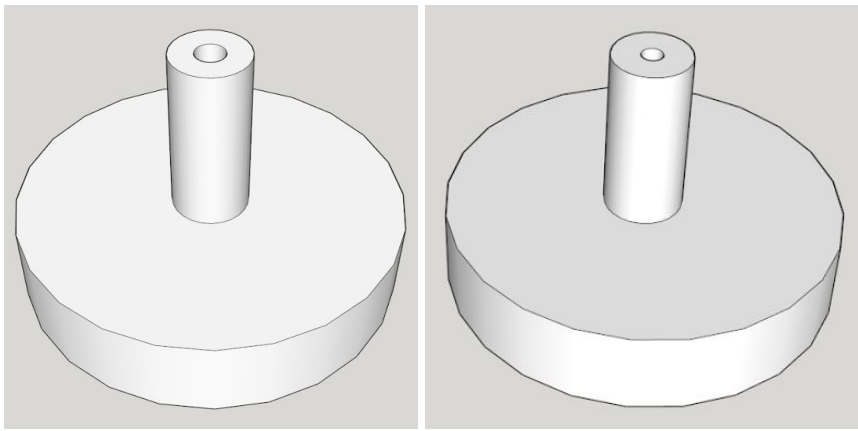
L'ensemble du mécanisme a été intégré sur la face avant du robot comme montré sur la figure suivante :



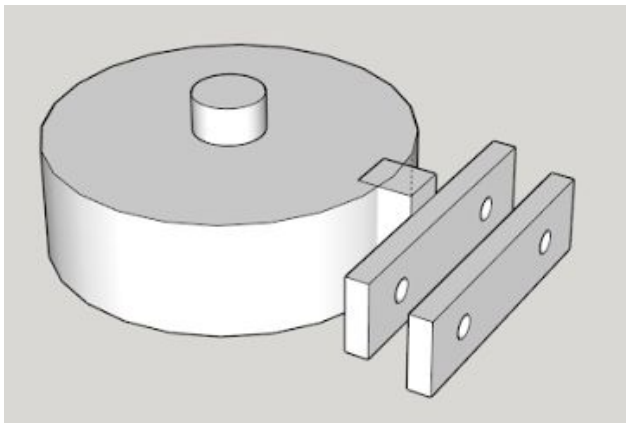
8.5. Pièces réalisées

Pour réaliser ce mécanisme, nous avons conçu plusieurs pièces que nous avons soit imprimé en 3D soit récupérées du robot de l'année dernière :

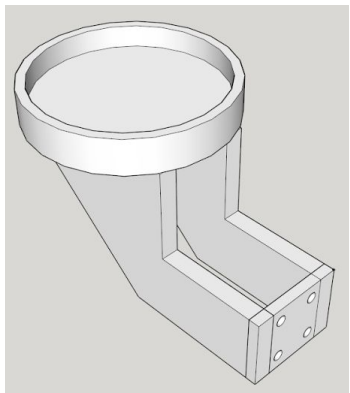
- la bague du parasol : le parasol vient se fixer dessus ainsi que le ressort qui permet l'ouverture du parasol. Ce support est quand à lui fixer sur la courroie au moyen d'un système de serrage. Il s'agit de la pièce mobile de notre mécanisme, c'est à dire celle qui va se déplacer à l'intérieur du tube



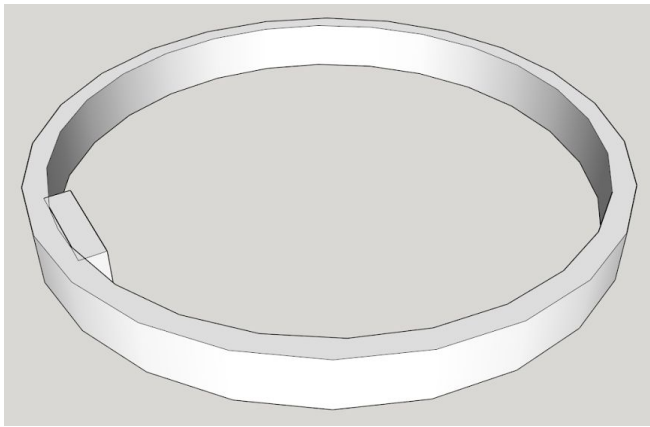
- le support du parasol : ce support permet de déposer la pièce décrite précédemment et d'attacher la courroie. Ainsi, lorsque la courroie monte/descend, la pièce monte/descend également



- Le support du tube : cette pièce a pour unique but de maintenir le tube à la bonne distance par rapport au rail



- la cale : permet au tube de garder le bon diamètre. En effet, comme nous avons dû réaliser une tranche sur le tube afin de pouvoir insérer le support du parasol, le tube avait tendance à perdre sa forme



9. Points positifs et problèmes rencontrés

9.1. Points positifs

Voici les quelques points positifs intéressants qui ont fait l'objet de notre réussite lors du concours Eurobot :

- Bon fonctionnement de la régulation. Le robot réalisait les bonnes distances et les bonnes rotations. De plus, il roulait bien droit !
- Aucun problème à déplorer depuis que nous avons repenser la structure du câblage mais ce n'est pas encore l'idéal (voir pistes d'améliorations)
- Bonne gestion du timing de la Raspberry et de l'Arduino qui nous a permi de rapporter beaucoup de point grâce à la funny action (déploiement du parasol) !
- Bonne autonomie des batteries au Lithium-Fer qui nous a permis de concourir toute une journée.
- Bon fonctionnement des différents capteurs à ultrason qui nous a permis de nous faire homologuer avant les premiers match.
- Codes commentés pour ceux qui prendront le relais.
- Résolution des pannes rapides grâce à nos différents tests allant du plus simple au plus complet avec à chaque test un éléments rajouté en plus.

9.2. Problèmes rencontrés

Comme chaque année, nous avons rencontré bon nombre de problèmes. Ceux qui sont cités ici ont été les plus gênants :

- Câblage initialement bordélique

- Certains fils étaient défectueux. Des journées entières ont été perdues pour trouver lesquels posaient problème
- Beaucoup de temps perdu pour la réinitialisation de la Raspberry et la mise en place du Wifi (Communication SSH)
- Mauvais placement des cartes au sein du robot qui nous empêchait de forer dans le châssis dans le but de pouvoir placer des accessoires.
- Bonne régulation pour tout type d'avance mais l'ensemble roue/arbre/moteur/roue codeuses n'étaient pas assez bien agencés pour avoir quelque chose de stable et fiable. De ce fait, le robot va parfois patiner et donc empêcher la régulation.
- Nous avions opté au départ pour un bras monté en 2 pièces métalliques liées par une ficelle. Ces deux pièces devaient se séparer par la force du servomoteur qui contrôlait le bras afin de lâcher le poisson attrapé au préalable. Nous avons finalement opté pour un raclement via le bord de la table car le souci rencontré étaient que les aimants du bras étaient trop puissant pour le servomoteur qui n'arrivait pas à forcer le poisson à se détacher.

10. Conseils et pistes d'améliorations

- Repenser le câblage et le positionnement des cartes pour plus de clarté. Avec un peu plus de temps, nous pensions organiser les différents blocs de la structure hardware dans différents étages du robot. De cette manière, le câble ne s'entrelacerait pas et il y aurait beaucoup moins de risque de court-circuits.
- La régulation n'est pas top. La régulation est bonne mais souvent, le robot met beaucoup de temps pour atteindre sa consigne. En effet, pour les rotations, le robot dépasse souvent sa consigne et met souvent du temps à revenir au bon angle. De ce fait, à lui tout seul le nombre d'action à effectuer en 1min30 était assez limiter. Le mieux que nous avons pu faire était de pousser un château et ramasser 2 poissons, ce qui est déjà pas mal. Pour améliorer cela, il serait intéressant de tenter la méthode Ziegler Nichols pour déterminer les meilleurs coefficients de régulation possible, et pourquoi pas intégrer une action Dérivée pour former un PID et ainsi anticiper l'approche d'une consigne.
- Changer les roues. Il faut des roues qui adhèrent le sol au maximum pour éviter le tout patinage.
- Pour plus de précision, il serait intéressant d'installer un petit système mécanique qui permettrait de raccorder directement l'arbre des codeurs avec ceux des moteurs. De cette manière, il n'y aurait plus besoin d'utiliser les roues codeuses qui posaient parfois problème.

11. Conclusion

Cette expérience collective nous a permis de travailler sur un projet de groupe concret et ce, sur une période conséquente de 8 mois. Il n'a pas été toujours facile de pouvoir gérer la collaboration et la coordination des tâches de chacun, mais nous avons tout au moins appris à y faire face tout en réalisant des projets de groupes en parallèle. Ce fût également l'occasion de réaliser un projet d'envergure internationale et de pouvoir représenter la

section électronique/informatique de l'ECAM ; une lourde épreuve que nous avons finalement relevé avec succès avec l'obtention de 224 points lors de la coupe de Belgique de robotique, nous classant 6ème sur 9 belges.

La première étape a tout d'abord été de s'approprier les règles du jeu et de s'immerger en analysant scrupuleusement le travail réalisé par les années antérieures. Sur ce dernier point, une attention particulière a été consacrée tout au long de l'année. Ce travail a été d'améliorer la visibilité et la compréhension des codes Arduino et Python et de leur enchaînement afin de permettre aux prochaines années de pouvoir directement s'engager dans l'amélioration de l'intelligence des codages au lieu de prendre du temps à la compréhension de celui-ci.

Ensuite il nous a fallu réaliser une panoplie de tests indépendants permettant d'avancer en parallèle sur les différents aspects relatant du projet à savoir la communication I2C, la manipulation des dynamixels, le fonctionnement des capteurs ultra son, la régulation, etc. pour finalement intégrer l'ensemble des tests dans un fichier complet.

12. Bonus

Nous avons réalisé une vidéo récapitulative du déroulement du projet pendant toute cette année jusqu'au match du concours Eurobot. Vous pouvons aller la visionner la vidéo sur YouTube en cliquant sur le lien ci-joint: https://www.youtube.com/watch?v=u6--y2PZbos