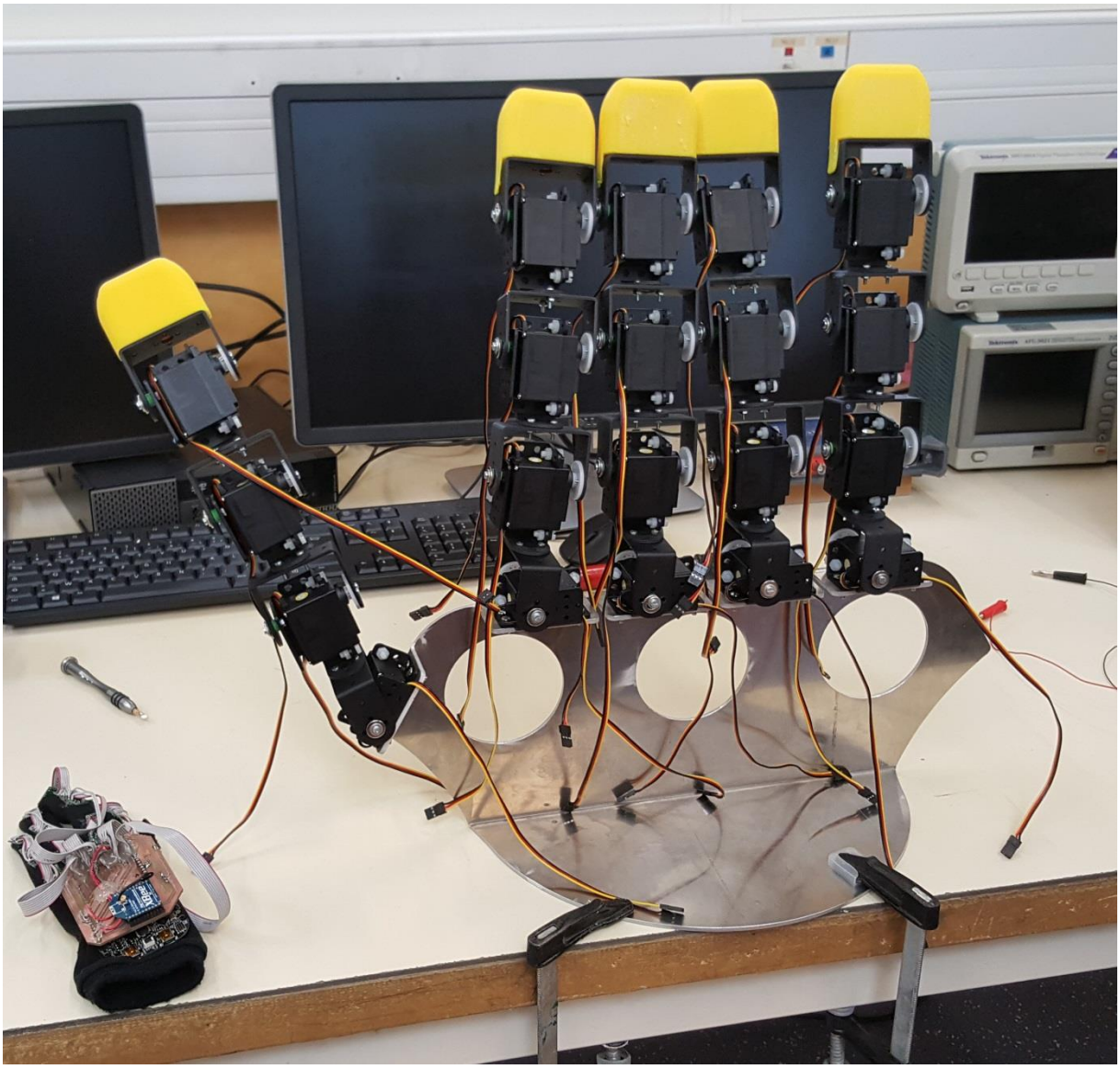
22/06/2016



IUT
CACHAN

RAPPORT DE PROJET

LICENCE MECSE option SESAM 2015/2016 | JOUAN Kevin

Dans ce rapport nous trouverons une présentation du projet d'étudiant de la promotion 2015/2016 de la formation Licence Métiers de l'Electronique : Communication et Systèmes Embarqués (MECSE) option Systèmes Embarqués et Supervision par Application Mobil (SESAM).

Dans un premier temps, une présentation du projet sera faite avec explication au mieux des attentes du produit finale. Suivis d'une partie expliquant les différents points réalisés dans le temps impartie. Nous verrons plusieurs points :

- Mécanique : ici nous verrons la conception mécanique de la main
- Communication : Ici nous verrons quelle façon nous parlons entre les systèmes
- Capteurs : Ici nous verrons tout le travail autour des capteurs
- Servomoteur : Ici nous verrons tout le travail autour des servomoteurs
- Android : Ici nous verrons la conception de l'application et son rôle dans la gestion de la main

En fin de rapport nous ferons un point sur l'avancement du projet et les objectifs de poursuite pour répondre à la problématique de départ.

Contact :

JOUAN Kevin
jouan.kevin@yahoo.fr
0604023524

Table des matières

1. Présentation du projet LAMR

Ce projet de Licence a pour objectif de concevoir une machine d'aide à la personne dans des tâches spécifiques. Cette machine sera, dans ce projet, une main robotique. A terme, il en aboutira un robot humanoïde qui pourra venir en aide aux pays qui ont dû faire face à des catastrophes naturelles ou anthropiques. Par exemple au Japon avec l'explosion de la centrale nucléaire de Fukushima, au sud-est du Brésil avec la rupture du barrage dans l'état du Minas Gerais, ou encore en Afrique avec les champs de mine.

Le développement de cette main permettra dans un premier temps de faire des manipulations à hauts risques, par exemple la transformation de produits chimiques dans un laboratoire ou l'utilisation de poudre explosive, sans mettre en danger l'Etre Humain.
Nous avons décidé d'appeler cette main : LAMR pour tout simplement : LA Main Robotique.

1.1.Le Principe

De nombreux prototypes de mains robotiques existent déjà. Leur principe est de reproduire une main humaine. Pour cela, des servomoteurs sont mis dans une partie de l'avant-bras et actionnent des tiges ou des fils permettant de plier un ou plusieurs doigts.

Ce système permet d'avoir un mouvement de doigts proche de celui d'un être humain au niveau musculaire. Cependant, contrairement à un doigt humain, celui du robot se pliera entièrement, et donc ne pourra faire l'ensemble des mouvements de doigts humain.

Le prototype que nous souhaitons concevoir a pour objectif de permettre de mettre en œuvre tous les mouvements de doigts possibles. Parmi ces mouvements, on citera : pouvoir tenir un objet, pointer du doigt ainsi que toutes les autres actions qu'une main humaine puisse faire.

La main robotique sera contrôlée par un Etre Humain munis d'un gant sur lequel sont fixés des capteurs. La liaison se fera sans fils. Une interface Android permettra d'avoir un visuel approfondit de l'état de la main et du gant.

2. Conception

2.1.Les composants

En annexe, des liens sont à disposition pour les datasheet et les sites d'achat.

2.1.1. Les Servomoteurs

Figure 1: Servomoteurs HITEC HS422

Les servomoteurs utilisés pour ce premier prototype sont des HITEC HS422. Nous les avons choisis pour leur poids (47g) ainsi que leur couple (4.7 kg/cm).

2.1.2. L'armature

Figure 4: Armature U3848

Figure 3: armature U5543

Figure 2: Armature en L

Pour la création de l'armature nous utilisons des pièces préfabriquées compatibles avec les servomoteurs. Voici le mode de fixation que nous utiliserons :

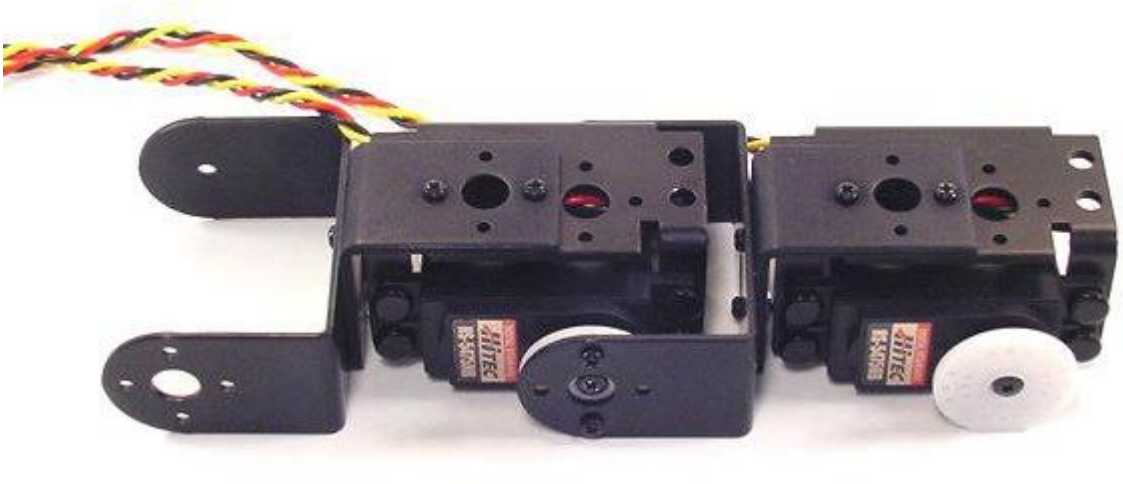


Figure 5: Fixation armature

Ce système de fixation est la plus repondu dans le monde de la robotique.

2.1.3. La carte de développement

Figure 6: KL25Z

La carte de développement utilisé est une KL25Z de chez FREESCALE (maintenant NXP).Nous l'avions choisi pour ses nombreuses broches. Deux cartes sont utilisées, une pour les capteurs, une pour les servomoteurs. Mais suite à une mauvaise lecture, nous ne pouvions contrôler que 10 servomoteur en même temps. Suite à cela, nous nous sommes tournés vers une carte d'interface qui nous permet de piloter seize servomoteurs.

2.1.4. La carte d'interfaçage

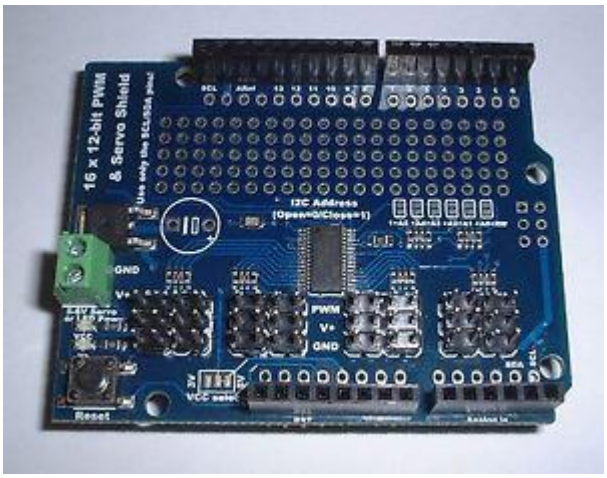


Figure 7:Carte interfaçage

Cette carte est munie du composant PCA9685 qui permet de générer un signal PWM sur 16 pins distincts. Il communique en I2C avec la carte de développement. Dans le code il suffira d'indiquer la pin puis le pourcentage de la PWM.

2.1.5. Communication

2.1.5.1. Xbee

Figure 8: Xbee

Pour la communication entre le gant et la main robotique nous utilisons des xbee. Simple de mise en œuvre, il permette une communication rapide et sur des grandes distances.

2.1.5.2. Bluetooth

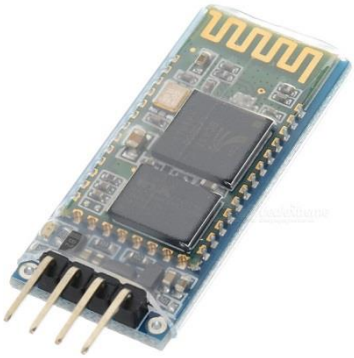


Figure 9: Module Bluetooth HC06

Un module Bluetooth est soudé sur la carte du gant pour la réception et l'émission de donnes vers un téléphone ou une tablette. Le téléphone servira de supervision sur le système.

2.1.6. Capteurs

Figure 10: capteur LSM303D

Pour la partie gant nous utilisons des capteurs de chez POLOLU. Ce sont des Accéléromètre/compas, nous utiliserons que l'accéléromètre. Ils communiquent en bus SPI.

2.1.7. Base/Paume

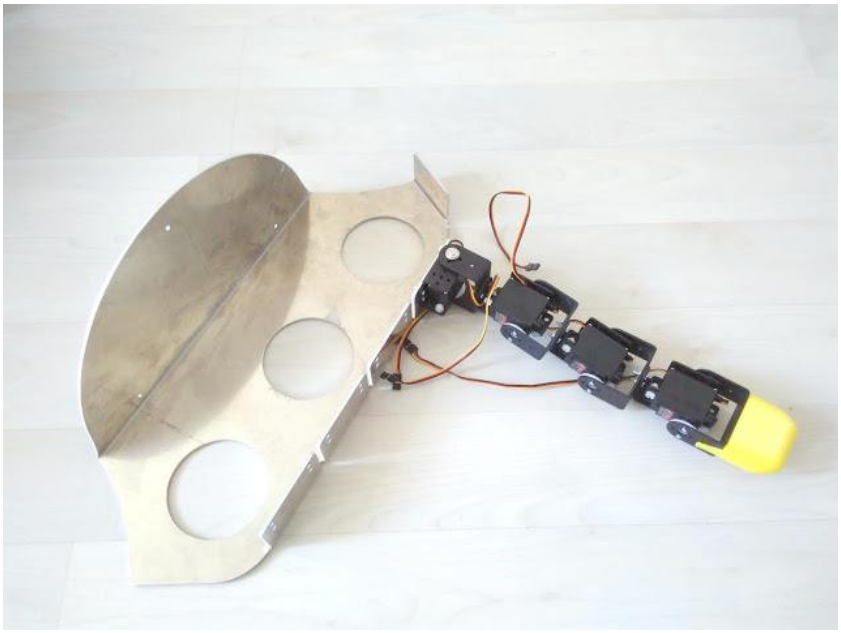


Figure 11: Base réalisé par Innovlab

Nous avons confié la conception du support des doigts à Innovlab, présent dans l'IUT de Cachan. Ce premier prototype nous à servis lors de notre présentation pour notre formation en Licence. Par la suite, un design et une forme sera définie pour ressembler le plus à une paume de main.

2.2.Impression 3D

2.2.1. Bout de doigt

Figure 12: Impression 3D bout doigt

Pour une ressembler le mieux à une main, nous avons conçu sur le logiciel SolidWorks une pièce en 3D. Cette pièce est mise au bout du doigt robotique.

2.2.2. Axe de Rotation

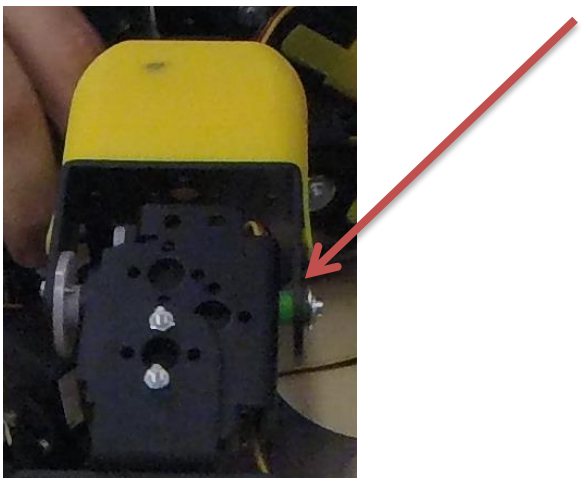


Figure 13: Impression 3D cylindre

Nous avons également conçu un petit cylindre vert permettant la rotation de l'armature en U.

2.3.Conception de la main robotique

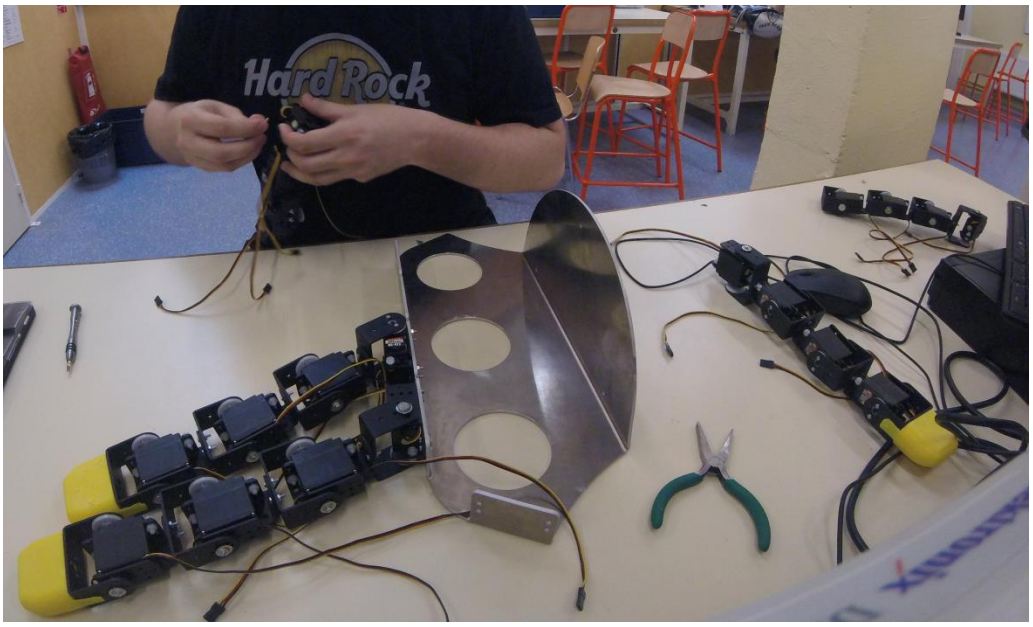


Figure 14: Fixation des doigts ensemble

Après avoir fixé les doigts ensemble suivant le modèle de la Figure 5: Fixation armature, nous assemblons les servomoteurs de tilt gauche droite sur le châssis. Tous les servomoteurs des phalanges sont pilotés par le composant de la carte shield. N'ayant pas assez de sortie, nous décidons de pilotés les servomoteurs de tilt gauche droite directement avec les PWM de la KL25Z.

2.4.Conception de la main capteur

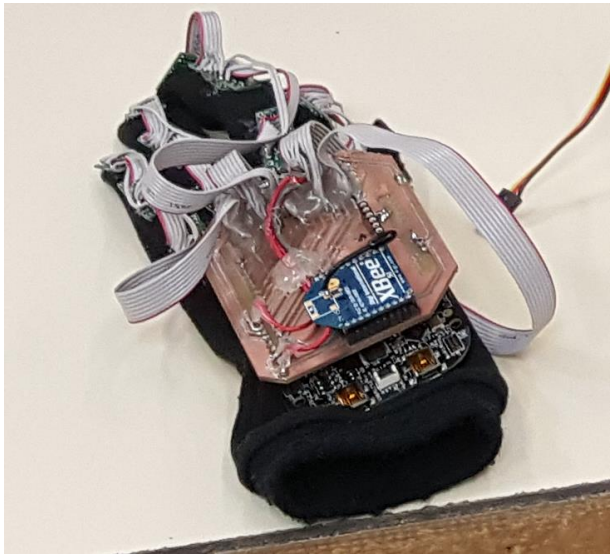


Figure 15: Assemblage de la main capteur

Les capteurs sont collés sur le gant au niveau de chaque phalange. Des nappes permettent la transmission des signaux et des alimentations.

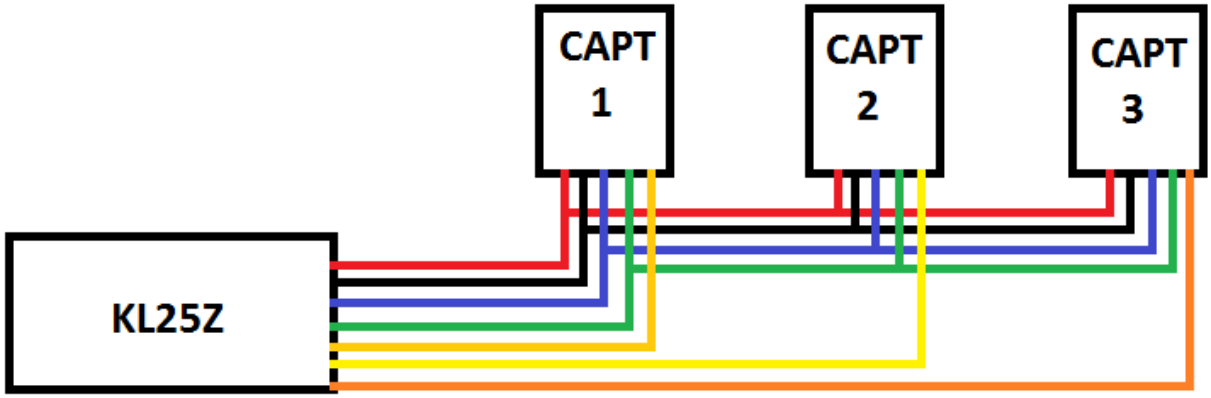


Figure 16: Câblage des Capteurs

Voici le schéma de câblage des capteurs avec pour légendes :

- En Rouge : l'alimentation
- **En Noir** : La masse
- En Bleu : SDA
- En Vert : SDO
- En Jaune, Orange et Ocre : les Chip Select du bus spi

Ceci est un exemple pour 3 capteurs, Il est de même avec le reste des capteurs. Cependant, Suite à des tests, mesure et dépannage, nous avons découvert que lorsqu'on a beaucoup de capteur, une grande impédance ce fait.

Lors d'un test sur les 5 doigts, les capteurs les plus éloignés, donc ceux ce trouvant à l'extrémité des doigts, ne répondais pas. Si on retire un doigt, tous les capteurs répondent présent. Nous avons donc été obligé de configurer un 2éme bus SPI pour pouvoir tout faire fonctionné. Pour éviter tout autre problème nous avons 3 doigts sur un bus et les 2 autres sur le 2éme bus. Cela est dû à une trop grande impédance sur le bus.

Le Bus appelé capteur_2 est celui du pouce et index, le bus capteur est celui des autres doigts.

3. Programmation

Ici nous allons détailler chaque code que nous avons développé pour ce projet. Cela permettra de bien comprendre la méthode de travail et le fonctionnement du programme. Nous avons programmé sous KEIL avec du code mbed

3.1.Main Robotique

3.1.1. Programme Fonctionnel

3.1.1.1. Define

```
//Phalanges doigt 0
#define D0M1 0x00
#define D0M2 0x01
#define D0M3 0x02
#define D0M0 0x03

//Phalanges doigt 1
#define D1M1 0x04
#define D1M2 0x05
#define D1M3 0x06
#define D1M0 0x07

//Phalanges doigt 2
#define D2M1 0x06
#define D2M2 0x07
#define D2M3 0x08

//Phalanges doigt 3
#define D3M1 0x09
#define D3M2 0x0A
#define D3M3 0x0B

//Phalanges doigt 4
#define D4M1 0x0C
#define D4M2 0x0D
#define D4M3 0x0E

//Tilt doigts
#define D0M0
#define D1M0
#define D2M0
#define D3M0
#define D4M0
```

Figure 17: Programmation Moteur Define

Ce code permet de définir l'adresse des servomoteurs. RAPPEL : Les phalanges sont contrôler par le PCA9685, l'adresse donné est donc le numéro de sortie du composant. Le nom donné comme par exemple D2M3 signifie : Doigt 2 Moteur 3. Le programme coté main capteur défini l'ordre des doigts et des servomoteurs. Ici nous avons donc le doigt : Majeur, Moteur : base. Donc le capteur D2C3 contrôle le moteur D2M3.

L'adressage des moteurs de tilt gauche droit n'a pas été fait. Il faudra indiquer la patte que le signale PWM utilisera sur la carte KL25Z.

3.1.1.2. Création des objets

```
DigitalOut myled(LED1);
I2C i2c_PCA(PTC2, PTC1);
PCA9685 PCA_SERVO(ADDR, i2c_PCA, 50);

Serial xbee(PTE0,PTE1);
```

Figure 18: Programmation Moteur Création des objets

Nous avons ici la création des entrées sorties pour la carte :

- Une LED pour vérifier que la carte rentre bien dans la boucle while(1) que l'on verra plus tard.
- Le Bus I2C qui est nécessaire au fonctionnement de la bibliothèque du composant PCA9685.
- Le composant PCA9685 qui va nous permettre d'envoyer des instructions au servomoteur.
- Une liaison Série pour la communication Xbee.

3.1.1.3. Interruption

```
void irq_xbee(void){
    text_xbee[idx_carac_xbee_in] = xbee.getc();
    send_measure = 1;
    idx_carac_xbee_in =  (idx_carac_xbee_in+1) % 5;

}
```

Figure 19: Programmation Moteur Interruption Xbee

Voici le code que vas exécuter le programme lorsqu'une trame envoyé par le Xbee Gant sera reçu par le Xbee Robot. Il va tout simplement mettre la trame dans le tableau text_xbee.

3.1.1.4. Angle des Servomoteurs

```
void angle(int moteur, int valeur){

    valeur = valeur * 10;// Mise en forme correspondant
    valeur = valeur + 1100;//  Ajout de l'offset

    if(valeur >= 2400)
    {
        valeur = 2400;
    }
    if(valeur <= 1100)
    {
        valeur = 1100;
    }

    PCA_SERVO.set_pwm_pw(moteur, valeur);
}
```

Figure 20: Programmation Moteur Angle du Servomoteur

Cette fonction sert à donne l'ordre au servomoteur d'aller à un angle précis. Elle demande deux paramètres, le premier est le nom du servomoteur (donc celui indiquer dans le define), et l'angle qu'on veut lui indiquer (donc la case du tableau reçu du xbee correspondant). Voici un tableau expliquant le principe, on n'inclus pas les servomoteurs de tilt gauche et droit.

Nous choisissons d'assigner ici la case 0 du tableau de réception au capteur du bout du pouce (réf define programme carte capteur).

Numéro de la case du tableau	Valeur Tableau Réception	Fonction angle
0	87	angle (D0M1, text_xbee [0])
1	28	angle (D0M2, text_xbee [1])
2	88	angle (D0M3, text_xbee [2])
3	104	angle (D1M1, text_xbee [3])
4	82	angle (D1M2, text_xbee [4])
5	45	angle (D1M3, text_xbee [5])
6	72	angle (D2M1, text_xbee [6])
7	8	angle (D2M2, text_xbee [7])
8	101	angle (D2M3, text_xbee [8])
9	4	angle (D3M1, text_xbee [9])
10	88	angle (D3M2, text_xbee [10])
11	78	angle (D3M3, text_xbee [11])
12	47	angle (D4M1, text_xbee [12])
13	107	angle (D4M2, text_xbee [13])
14	65	angle (D4M3, text_xbee [14])

Les valeurs sont comprises entre 0 et 120. Nous multiplions par 10 puis ajoutons un offset de 1100 pour rentrer dans la plage du servomoteur, soit entre 1100 et 2400. Nous les avons mesurées en laissant un peu de course avant les buter en cas de forçage. Pour finir nous exécutons la fonction PCA_SERVO.set_pwm.pw(moteur, valeur) ; pour donner l'instruction au servomoteur désirer.

3.1.1.5. Fonction Main

```
//  Configuration xbee
xbee.baud(115200);// Vitesse du xbee
xbee.attach(&irq_xbee);// Assignation des interruptions

PCA_SERVO.init();
for(i=0;i<=15;i++){
    PCA_SERVO.set_pwm_output_on_0(i, 0);
}
```

Figure 21: Programmation Moteur Main

Au début du main, nous initions la vitesse du xbee et l'attachement de l'interruption avec la fonction irq_xbee que nous avons expliqué précédemment. Nous faisons un init du composant PCA9685 et une mise à 0 de toutes les pwm.

```
while(1) {
        //angle(D0M1, text_xbee[1]);
        //angle(D1M1, text_xbee[2]);
        if(send_measure == 1)
        {
            angle(D0M1, text_xbee[1]);
            angle(D0M2, text_xbee[1]);
            angle(D0M3, text_xbee[1]);

            angle(D1M1, text_xbee[2]);
            angle(D1M2, text_xbee[2]);
            angle(D1M3, text_xbee[2]);

            send_measure = 0;
        }
}
```

Figure 22: Programmation Moteur Main suite

Cette boucle infinie va servir à envoyer en continu les valeurs reçues. Pour notre présentation, nous avons récupéré les valeurs de l'axe X du capteur du bout de l'index et du majeur pour les transmettre respectueusement sur les trois servomoteurs correspondant au doigt. Par manque de temps nous n'avons pas pu récupérer les six capteurs à envoyer aux servomoteurs. Sur le programme ci-dessus, nous avons câblé l'index à la place du pouce et le majeur à la place de l'index.

3.1.2. Amélioration du programme

Voici quelque piste pour poursuivre le codage de la main robotique.

Tous les calculs sont faits sur la carte capteurs. Les valeurs reçues peuvent être directement envoyé aux servomoteurs après mise en forme si nécessaire.

En premier lieu, il faut créer le programme permettant de contrôler les cinq servomoteurs à la base des doigts permettant de tilt gauche droite. Il faut également définir des limites pour ne pas bloquer les autres doigts.

Dans la boucle while(1) et dans la boucle if, suffit de mettre les fonctions angle comme définie dans le tableau.

Les cinq dernières cases du tableau serviront de valeur pour les servomoteurs du tilt. Cela sera calculer avant l'envoyé en fonction des données des capteurs. De même pour la comparaison du coefficient entre les valeurs pour déduire si toutes les phalanges ce plie ou non.

3.2.Main Capteurs

3.2.1. Programme Fonctionnel

3.2.1.1. Define

```
//Phalanges doigt 0
DigitalOut D0C1(PTA16);// Pouce bout
DigitalOut D0C2(PTA17);// Pouce millieu
DigitalOut D0C3(PTE31);// Pouce base

//Phalanges doigt 1
DigitalOut D1C1(PTC13);// Index bout
DigitalOut D1C2(PTC16);// Index millieu
DigitalOut D1C3(PTC17);// Index base

//Phalanges doigt 2
DigitalOut D2C1(PTC10);// Majeur bout
DigitalOut D2C2(PTC11);// Majeur millieu
DigitalOut D2C3(PTC12);// Majeur base

//Phalanges doigt 3
DigitalOut D3C1(PTC4);// Annulaire bout
DigitalOut D3C2(PTC5);// Annulaire millieu
DigitalOut D3C3(PTC6);// Annulaire base

//Phalanges doigt 4
DigitalOut D4C1(PTC7);// Orriculaire bout
DigitalOut D4C2(PTC0);// Orriculaire millieu
DigitalOut D4C3(PTC3);// Orriculaire base
```

Figure 23: Programmation Capteur Define

Ce code permet de définir l'adresse des capteurs. Le nom donné comme par exemple D2C3 signifie : Doigt 2 Capteur3. C'est ici que l'on définit l'ordre des doigts et des servomoteurs. Ici nous avons donc le doigt : Majeur, capteur : base. Donc le capteur D2C3 contrôle le moteur D2M3.

3.2.1.2. Création des objets

```
DigitalOut myled(LED1);//  creqtion d'une led
SPI capteur(PTD2, PTD3, PTD1); // mosi, miso, sclk
SPI capteur_2(PTD6, PTD7, PTD5); // mosi, miso, sclk
Ticker action; //  creation d'un timer

Serial xbee(PTE0,PTE1);
Serial hc06(PTE0,PTE1);
```

Figure 24: Programmation Capteur Création des Objets

Nous avons ici la création des entrées sorties pour la carte :

- Une LED pour vérifier que la carte rentre bien dans la boucle while(1) que l'on verra plus tard.

- Deux Bus SPI qui serviront à communiquer avec les capteurs. (voir 2.4 Conception de la main capteur).
- Un timer pour contrôler la période d'activation de l'interruption.
- Une liaison Série pour la communication Xbee.

3.2.1.3. *Déclaration des structures pour les capteurs*

```
struct S_accel{
    int x;
    int y;
    int z;
    DigitalOut *CS;
}a[15], b[15];
```

Figure 25: Programmation Capteur Structure des capteurs

Cette structure va permettre de rassembler toute les valeurs des capteurs. En créant le tableau "a [15]" chaque case correspond à un capteur. Donc chaque cas contient la valeur x y et z ainsi que le numéro de la pin de son CS.

```
void irq_xbee(void){
    text_xbee[idx_carac_xbee_in] = xbee.getc();
    idx_carac_xbee_in = (idx_carac_xbee_in+1) % 20;
}
//
void irq_hc06(void){
    text[idx_carac_hc06_in] = hc06.getc();
    idx_carac_hc06_in = (idx_carac_hc06_in+1) % 20;
}
//
```

Figure 26: Programmation Capteur Interruption

Comme pour la carte des moteurs, nous créons des fonctions pour les interruptions. Ici nous avons en plus l'interruption pour le Bluetooth.

3.2.1.4. Assignation et mise à 1 des CS

```
void Init_Cs(void){
    //Phalanges doigt 0
    a[0].CS = &D0C1;
    a[1].CS = &D0C2;
    a[2].CS = &D0C3;

    //Phalanges doigt 1
    a[3].CS = &D1C1;
    a[4].CS = &D1C2;
    a[5].CS = &D1C3;

    //Phalanges doigt 2
    a[6].CS = &D2C1;
    a[7].CS = &D2C2;
    a[8].CS = &D2C3;

    //Phalanges doigt 3
    a[9].CS = &D3C1;
    a[10].CS = &D3C2;
    a[11].CS = &D3C3;

    //Phalanges doigt 4
    a[12].CS = &D4C1;
    a[13].CS = &D4C2;
    a[14].CS = &D4C3;

    for (i=0;i <= 14;i++){// Tous les CS a 1
        *(a[i].CS) = 1;
    }
 }
//
```

Figure 27: Programmation Capteur Initialisation CS

Cette fonction permet d'assigner les doigts à une case du tableau puis de mettre tous les CS à 1.

3.2.1.5. Ecriture dans un registre

```
void Write_Register(int cap, int addr, int valeur){
    //Ecriture dans un registre
    if(cap <=5){
        *(a[cap].CS) = 0;
        valeur = (1<<8)+ valeur;
        capteur_2.write(addr);
        capteur_2.write(valeur);
        *(a[cap].CS) = 1;
    }else{
        //Ecriture dans un registre
        *(a[cap].CS) = 0;
        valeur = (1<<8)+ valeur;
        capteur.write(addr);
        capteur.write(valeur);
        *(a[cap].CS) = 1;
    }
}
//
```

Figure 28: Programmation Capteur Ecriture dans un registre

Grâce à cette fonction, nous pouvons écrire dans un registre d'un des capteurs. Le code a été codé de façon à ce que le choix du bus spi soit transparent à l'utilisateur. Pour information, la première condition du if commande le bus 2 (pouce et index) et le else le reste des doigts. Il sera de même pour les autres fonctions qui doivent utiliser les capteurs. Rien de spécial à expliquer sur le code, ne serait-ce que nous suivons le protocole de n'importe quel bus SPI.

3.2.1.6. Lecture dans un registre

```
int Read_Register(int cap, int addr){
    int addr_2;
    // Lecture d'un registre et affichage sur port comm
    if(cap <=5){
        addr_2 = (8<<4) + addr;
        *(a[cap].CS) = 0;
        capteur_2.write(addr_2);
        term = capteur_2.write(0x00);
        printf("Registre = 0x%X\r\n", term);
        *(a[cap].CS) = 1;
        return term;
    }else{
        addr_2 = (8<<4) + addr;
        *(a[cap].CS) = 0;
        capteur.write(addr_2);
        term = capteur.write(0x00);
        printf("Registre = 0x%X\r\n", term);
        *(a[cap].CS) = 1;
        return term;
    }
}
//
```

Figure 29: Programmation Capteur Lecture dans un registre

Comme pour l'écriture, cette fonction suit le protocole du bus SPI pour la lecture d'un registre. Elle gère automatiquement quel bus SPI choisir.

3.2.1.7. Lecture des Axes

```
void Read_x(int cap){
    if(cap <=5){
        *(a[cap].CS) = 0;
        capteur_2.write(0xA8);
        acc_x[0]= capteur_2.write(0x00);
        //printf("X_L = 0x%X\r\n", acc_x[0]);  //Affichacge LSB axe X
        *(a[cap].CS) = 1;

        *(a[cap].CS) = 0;
        capteur_2.write(0xA9);
        acc_x[1] = capteur_2.write(0x00);
        //printf("X_H = 0x%X\r\n", acc_x[1]);  // Affichage MSB axe X
        *(a[cap].CS) = 1;

        a[cap].x = ((acc_x[1] <<8) | acc_x[0]);  //  Mise en forme valeur
        //printf("X = 0x%d\r\n", a[cap].x);  //  Affichage valeur axe x
    }else{
        *(a[cap].CS) = 0;
        capteur.write(0xA8);
        acc_x[0]= capteur.write(0x00);
        //printf("X_L = 0x%X\r\n", acc_x[0]);  //Affichacge LSB axe X
        *(a[cap].CS) = 1;

        *(a[cap].CS) = 0;
        capteur.write(0xA9);
        acc_x[1] = capteur.write(0x00);
        //printf("X_H = 0x%X\r\n", acc_x[1]);  // Affichage MSB axe X
        *(a[cap].CS) = 1;

        a[cap].x = (acc_x[1] <<8)+ acc_x[0];  //  Mise en forme valeur
        //printf("X = %d\r\n", a[cap].x);  //  Affichage valeur axe x
    }
}
//
```

Figure 30: Programmation Capteur Axe Z

```
void Read_y(int cap){
    if(cap <=5){
        *(a[cap].CS) = 0;
        capteur_2.write(0xAA);
        acc_y[0]= capteur_2.write(0x00);
        //printf("Y_L = 0x%X\r\n", acc_y[0]);  //Affichacge LSB axe Y
        *(a[cap].CS) = 1;

        *(a[cap].CS) = 0;
        capteur_2.write(0xAB);
        acc_y[1] = capteur_2.write(0x00);
        //printf("Y_H = 0x%X\r\n", acc_y[1]);  // Affichage MSB axe Y
        *(a[cap].CS) = 1;

        a[cap].y = (acc_y[1] <<8)+ acc_y[cap];  //  Mise en forme valeur
        //printf("Y = 0x%X\r\n", a[0].y);  //  Affichage valeur axe Y
    }else{
        *(a[cap].CS) = 0;
        capteur.write(0xAA);
        acc_y[0]= capteur.write(0x00);
        //printf("Y_L = 0x%X\r\n", acc_y[0]);  //Affichacge LSB axe Y
        *(a[cap].CS) = 1;

        *(a[cap].CS) = 0;
        capteur.write(0xAB);
        acc_y[1] = capteur.write(0x00);
        //printf("Y_H = 0x%X\r\n", acc_y[1]);  // Affichage MSB axe Y
        *(a[cap].CS) = 1;

        a[cap].y = (acc_y[1] <<8)+ acc_y[cap];  //  Mise en forme valeur
        //printf("Y = 0x%X\r\n", a[0].y);  //  Affichage valeur axe Y
    }
}
//
```

Figure 31: Programmation Capteur Lecture Y

```
void Read_z(int cap){
    if(cap <=5){
        *(a[cap].CS) = 0;
        capteur_2.write(0xAC);
        acc_z[0]= capteur_2.write(0x00);
        //printf("Z_L = 0x%X\r\n", acc_z[0]);  //Affichacge LSB axe Z
        *(a[cap].CS) = 1;

        *(a[cap].CS) = 0;
        capteur_2.write(0xAD);
        acc_z[1] = capteur_2.write(0x00);
        //printf("Z_H = 0x%X\r\n", acc_z[1]);  // Affichage MSB axe Z
        *(a[cap].CS) = 1;

        a[cap].z = (acc_z[1] <<8)+ acc_z[0];  //  Mise en forme valeur
        //printf("Z = 0x%X\r\n", a[cap].z);  //  Affichage valeur axe Z
    }else{
        *(a[cap].CS) = 0;
        capteur.write(0xAC);
        acc_z[0]= capteur.write(0x00);
        //printf("Z_L = 0x%X\r\n", acc_z[0]);  //Affichacge LSB axe Z
        *(a[cap].CS) = 1;

        *(a[cap].CS) = 0;
        capteur.write(0xAD);
        acc_z[1] = capteur.write(0x00);
        //printf("Z_H = 0x%X\r\n", acc_z[1]);  // Affichage MSB axe Z
        *(a[cap].CS) = 1;

        a[cap].z = (acc_z[1] <<8)+ acc_z[0];  //  Mise en forme valeur
        //printf("Z = 0x%X\r\n", a[cap].z);  //  Affichage valeur axe Z
    }
}
//
```

Figure 32: Programmation Capteur Lecture Axe Z

C'est trois fonctions permettent de récupérer les valeurs des trois axes. Tout en sélectionnant le bon bus SPI, elles enregistrent le MSB et LSB de l'axe qui lui concerne et la mets en forme (décalage du msb).

3.2.1.8. Vérification de la Présence des Capteurs

```
void Capteur_present(void){// verification presence capteurs
    i = 0;
    for (i=0;i <= 5;i++){
        present = Read_Register(i, 0x0F);
        if( present == 0b01001001){
            printf("capteur_1 %X = OK\r\n", i);
        } else{
            printf("capteur_1 %X = Error\r\n", i);
        }
    }
    for (i=6;i <= 14;i++){
        present = Read_Register(i, 0x0F);
        if( present == 0b01001001){
            printf("capteur_2 %X = OK\r\n", i);
        } else{
            printf("capteur_2 %X = Error\r\n", i);
        }
    }
}
//
```

Figure 33: Programmation Capteur Vérification des capteurs

Avec cette fonction on va pouvoir faire une vérification au démarrage que tous les capteurs nous répondent.

3.2.1.9. Action de l'Interruption

```
void lecture(void){
    for(i=0;i <= 14;i++){
        //  Lecture des axes
        Read_x(i);
        Read_y(i);
        Read_z(i);

        a[i].x = (a[i].x + 32768)/2;
        a[i].y = (a[i].y + 32768)/2;
        a[i].z = (a[i].z + 32768)/2;

        a[i].x = a[i].x /100;

        if(a[i].x >= 240)
        {
            a[i].x = 240;
        }
        if(a[i].x <= 180)
        {
            a[i].x = 180;
        }

        a[i].x = 180 - (a[i].x);
        a[i].x = -(a[i].x * 2);
        printf("%d\r\n", a[3].x);


/*          angleX = (0,0167 * a[i].x) - 300;

            angleX = (angleX_old + angleX) /2;
            angleX_old = angleX;
*/


            // Ajout dans le tableau
            data[i] = a[i].x;
            //printf("%d\r\n", data[3]);
        }
        xbee.printf("%c", data[0]);
        xbee.printf("%c", data[3]);
        xbee.printf("%c", data[6]);
        xbee.printf("%c", data[9]);
        xbee.printf("%c", data[12]);
}
//
```

Figure 34: Programmation Capteurs Lecture

Voilà la partie la plus complexe du projet. Tous les calculs ce font ici, on récupère les valeurs puis on les mets en forme. C'est ici que l'on testera le mouvement des capteurs pour déterminer comment les capteurs bougent. Ici nous récupérons simplement l'axe X et la mettons en forme pour être en accord avec les servomoteurs. Puis nous envoyons le tableau par xbee.

3.2.1.10. Fonction Main

```
//  Configuration xbee
xbee.baud(115200);// Vitesse du xbee
xbee.attach(&irq_xbee);// Assignation des interruptions

hc06.baud(115200);// Vitesse du xbee
hc06.attach(&irq_hc06);// Assignation des interruptions

//  Configuration bus spi
capteur.format(8,3);// Selection parametre du bus SPI
capteur.frequency(5000000);// Vitesse du bus SPI

capteur_2.format(8,3);// Selection parametre du bus SPI
capteur_2.frequency(5000000);// Vitesse du bus SPI

Init_Cs();//  Assignation des chip select

Capteur_present();//  Verification presence capteurs

for (i=0;i <= 5;i++){//  Mise en marche capteurs
    Write_Register(i, 0x20, 0x67);// registre CTRL_REG1_A AODR = 10Hz
}
//
for (i=6;i <= 14;i++){//  Mise en marche capteurs
    Write_Register(i, 0x20, 0x67);// registre CTRL_REG1_A AODR = 10Hz
}
//
action.attach(&lecture, 0.1);

while(1) {

}
```

Figure 35: Programmation Capteur Fonction Main

On initialise le xbee, le Bluetooth, les bus SPI. On utilise la fonction Init_Cs pour configurer et mettre à 1 les CS. Pour finir, nous vérifions la présence de tous les capteurs, on active les capteurs et on assigne l'interruption pour avoir une lecture régulière des valeurs. On finit par une boucle while(1) infinie pour entrer en interruption quand cela est demandé.

3.2.2. Amélioration du programme

Voici encore quelque piste pour améliorer le programme capteur.

Pour commencer il faut récupérer toutes les valeurs des capteurs et envoyer la valeur de X d'un capteur vers son binôme coter servomoteur. Une fois cela fait, faire un test pour voir si tous les doigts bougent. Forcer les moteurs de tilt à 50% pour maintenir les doigts droit.

Ensuite il faut ensuite étudier les mouvements possibles de la main et regarder les valeurs que les capteurs envoient pour programmer les coefficients et ainsi envoyer les bonnes valeurs aux servomoteurs.

Il reste à faire le code pour le Bluetooth, récupérer et envoyer des valeurs. Et pour finir, un code pour des capteurs de force mis au bout des doigts faisant coefficients multiplicateur pour donner de la force ou du couple a la main robotique. Par exemple cela augmente l'angle demandé (comme il n'y a pas d'asservissement sur les moteurs sélectionnés.

4. Conclusion

Toute la partie hardware est finie, il ne restera plus que fixer les capteurs de force, câbler les alimentations. Pour la partie software, les pistes indiquées dans les parties sont de bons points de départ pour poursuivre le projet.