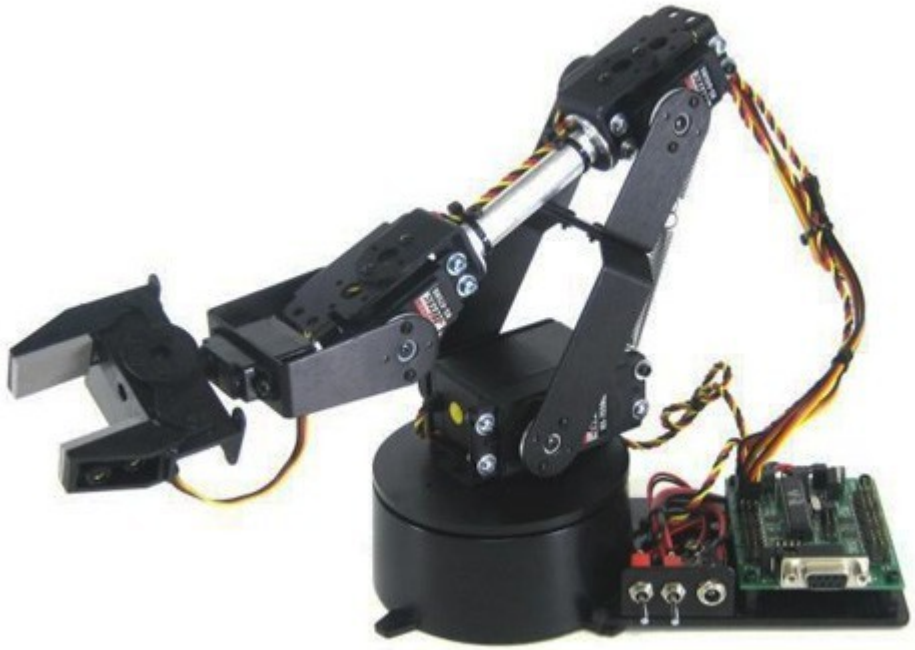
Projet : Bras Robotique
Contrôle du mouvement

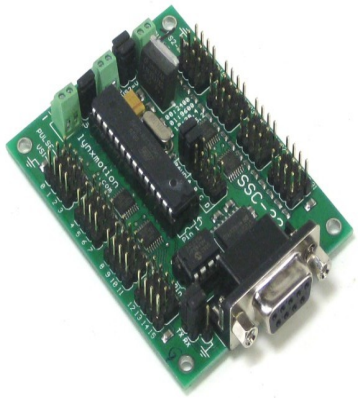


Ben younes & Tiamiou

POLYTECH GRENOBLE

I- Présentation du contexte.

Ce projet à pour but de développer un logiciel de contrôle de bras manipulateur pour l'assistance de personnes handicapés.



Notre bras Robotique : le AL5B est équipé d'une carte SSC32. Cette carte permet une communication bidirectionnel et dispose de 32 canaux pouvant contrôler des servomoteurs.
Celui-ci utilise une connexion série de 8 bits.

Le robot est aussi équipée d'une webcam lui permettant de détecter des objets grâce à des capteurs.

Ce projet est divisé en deux partie : La première consiste à s'occuper de la détection du capteur.
La deuxième partie,qui est la notre, est de s'occuper de toute la partie communication avec le robot ainsi que

II Cahier des charges :

Comme dit précédemment, nous allons nous concentrer uniquement sur la connexion avec le robot et de sa trajectoire.

Nous avons fixé les objectifs suivant :

1.Le logiciel doit être capable de communiquer avec tous robots équipé d'une carte SSC-32

2. Notre logiciel doit être capable de calculer une trajectoire correcte afin d'attraper l'objet .

3.-Pour récupérer les coordonnées de la pince. Un simulateur doit être implémenté représentant le bras robotique virtuellement.

4.On doit être capable avec le deuxième groupe de communiquer ensemble afin de détecter l'objet.

III – Travail effectué

Etape 1 :Établir et configurer la connexion avec la carte SSC-32

La première partie de notre projet fut d'établir la communication avec le robot au travers de la carte SSC-32. Pour cela, nous avons à notre disposition le manuel SSC-32 qui regroupe toutes les instructions nécessaires pour contrôler les différents servomoteurs, voici un rapide résumer des instructions disponibles :

Command Types and Groups			
1	Servo Movement	7	Read Analog Inputs
2	Discrete Output	8	12 Servo Hexapod Gait Sequencer
3	Byte Output	9	Query Hex Sequencer
4	Query Movement Status	10	Get Version
5	Query Pulse Width	11	Go to Boot
6	Read Digital Inputs	12	MiniSSC-II Compatibility

Servo Move or Group Move.

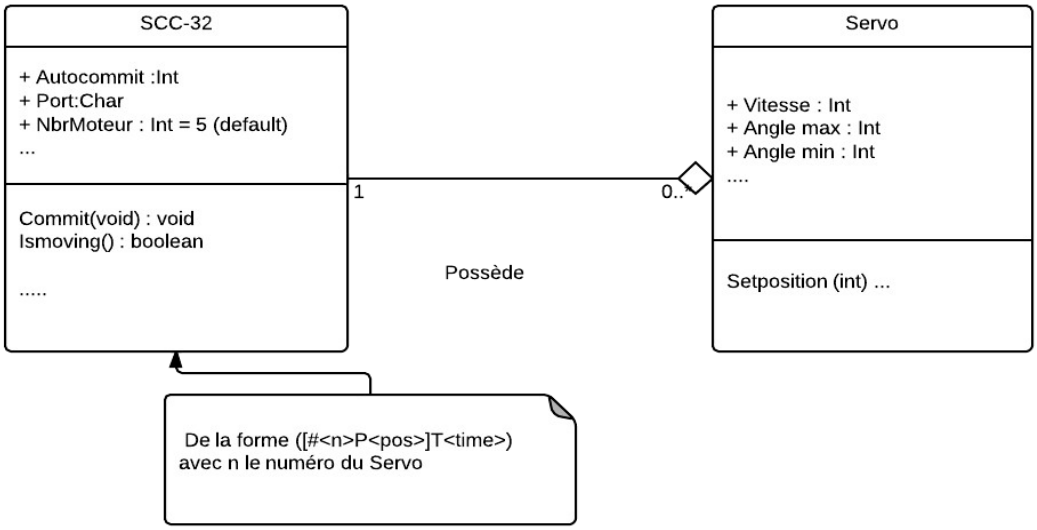
# <ch> P <pw> S <spd> ... # <ch> P <pw> S <spd> T <time> <cr>	
<ch>	Channel number in decimal, 0-31
<pw>	Pulse width in microseconds, 500-2500
<spd>	Movement speed in uS per second for one channel (Optional)
<time>	Time in mS for the entire move, affects all channels, 65535 max (Optional)
<cr>	Carriage return character, ASCII 13 (Required to initiate action)
<esc>	Cancel the current action, ASCII 27

Par exemple : Si on envoie en ASCII la commande suivante: "#5 P1600 S100 <cr>" Cette commande va faire bouger le servomoteur connecté au Chanel n°5 à la position « 1600 ». La position est définit comme la durée des impulsions qui détermine l'angle absolu de l'axe de sortie et donc la position du bras de

commande du servomoteur. Le tout exécuté à une vitesse définit à 752 us. Pour mieux comprendre comment la vitesse est définie, on considère qu'a 1000uS il est effectué en 1 seconde une rotation de 90°. Donc pour une valeur de la vitesse à 100 uS signifie que le servomoteur va prendre 10 seconde pour faire une rotation à 90°.

De plus, il existe des commandes qui permettent de recevoir des informations provenant de la carte. Par exemple si on l'envoie la commande : "Q <cr>" renvoie "+" si le robot est en mouvement.

Donc à partir des ces informations et des poste dans le forum du créateur de la carte, nous avons développé en python deux classes :



La première classe SSC-32 regroupe des méthode permettant d'établir la connexion et d'envoyer les différentes commandes à la carte. (Socket, port,). Celle-ci possède plusieurs objet Servo. Un servomoteur est définie par plusieurs attribut (Position ,vitesse,angle min, angle max...).

Pour bien comprendre, nous allons prendre un exemple :

Tout d'abord on initialise la connexion USB, généralement le port est « /dev/ttyUSB1 ». Le 115200 représente le braudbit(la vitesse de connexion).

ssc = ssc32.SSC32('/dev/ttyUSB1', 115200 ,vitesse=1000)

On initialise nos objets servomoteurs grâce à la fonction __getitem__ :

```
Servo1 = ssc['Servo1']
Servo2 = ssc['Servo2']
...
```

Pour cette exemple, les valeurs d'angle min,max... sont par défaut celle écrit dans le code.

On définit une liste chaque servomoteur la position en degrés voulus :

```
Position =      [(Servo1, 50),
                 (Servo2, -130 ), ...]

def ChargerPosition(listePosition):

    for move in listePosition:

        move[0].degree(move[1]) . #   degree
        ssc.commit(600) # 600 est le temps désiré.
        While (ssc.IsMoving){ # tant que le robot est en mouvement
        time.sleep(0.3) } #On temporise pour ne pas exécuter toutes les
                              commandes en même temps.
```

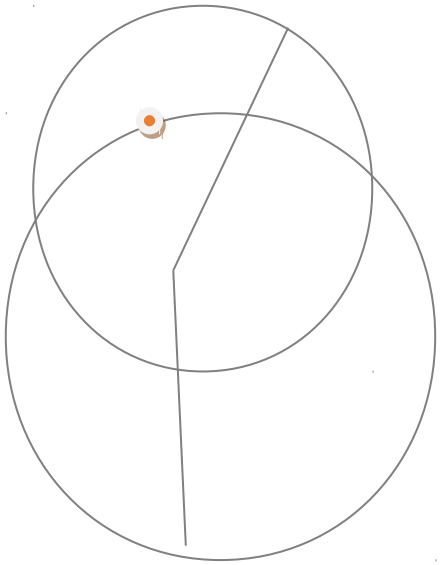
Une fois les valeurs chargées, il suffit d'utiliser la fonction commit() qui va envoyé pour chaque servomoteur la commande suivante :*[#<n>P<pos>]T<time>.*

Malheureusement, nous n'avons pas pu tester ces fonctions sur le bras robotique Le robot ce déplace mais pas du tout au bonne valeurs . Cependant même en utilisant le logiciel officiel fournit avec le robot, le résultat est le même. De plus il y a eu des soucis au niveau d'un servomoteur qui surchauffe.

Partie n°2 :Le simulateur

Le simulateur à plusieurs utilités. Mais la principale est de pouvoir avoir les coordonnées de la pince vis-à-vis du socle du robot en temps réel afin de pouvoir se positionner par rapport à l'objet.
Dans cette partie nous avons utilisé un module graphique 3D appelé "Visual" pour le langage de programmation python.
Celle-ci permet de simplifier la création d'objet 3D et leurs interactions.
Première étape création des différents axe :
On a fait en sorte que l'on puisse construire le robot en donnant uniquement la longueur des axes. Ainsi, en positionnant le point de départ à une coordonnée quelconque, on peut construire les axes en fonctions des angles de départ (on force le robot à s'allonger et à devenir le plus droit possible pour que les angles de départ correspondent). On en déduit ainsi les points d'intersection des axes. Pour les déplacements des bras, si je bouge le bras 1, le bras 2 bouge aussi. Les points suivent donc la même rotation. Mais si je bouge le bras 2, le bras 1 ne doit pas bouger. Ceci a été pris en compte. Pour garder la pince parallèle (qui va être utile pour le mode recherche), on l'abstient de rotation lorsque le bras 1 ou 2 bougent,

Par la suite, le but était de faire atteindre le point d'intersection entre le bras 2 et 3 (que l'on appelera axe2) sur une destination choisie.
Pour cela, on cherche le point d'intersection entre le cercle de centre axe2 et de rayon bras3, et le cercle de centre axe0 et de rayon ||Axe0 vers Destination||. Il y en aura deux. Pour chacun, on calcule l'angle entre l'intersection et la destination suivant axe0, on cherche la coordonnée du point axe2 suivant la rotation bras 1 et 2 par cet angle, puis pour chacune des intersections on garde celle qui détermine l'axe2 le plus proche de la destination.



Il s'en suit de nombreux calculs pour prendre en compte la rotation sur le plan xOz, pour déterminer les coordonnées des positions des objets sur les bras, pour déterminer la façon dont le robot doit récupérer un objet (destination décalée puis la pince avance vers l'objet).
De même, nous avons permis au robot de suivre toutes les trajectoires de points en 3D, afin de mettre en place le mode recherche du marqueur. Les calculs ne sont pas détaillés car il s'agit essentiellement de calcul d'intersections, de rotations simples et compliquées ainsi que de calculs de position parfois très complexes. (Quand on passe d'un plan XY à un plan XYZ suivant une rotation xOz, une partie des x se transforment z).

IV Amélioration possible

Nous avons pu établir la connexion avec le groupe travaillant sur la détection d'image. Grâce à un client/serveur agent , on peut échanger des informations grâce à un fichier xml que l'on reçoit.
Nous avons cependant nous n'avons pas eu le temps de l'implémenter dans le simulateur. Car on ne reçois que les coordonnées par rapport l'image et non de l'objet.
La solution que nous avons envisagé est que la camera doit guider le robot, on lui indiquant une direction jusqu'à que l'objet soit en face de la pince et ainsi attraper l'objet. Il nous aurais fallut quelques jours de plus pour pouvoir l'implémenter.

Ben younes & Tiamiou