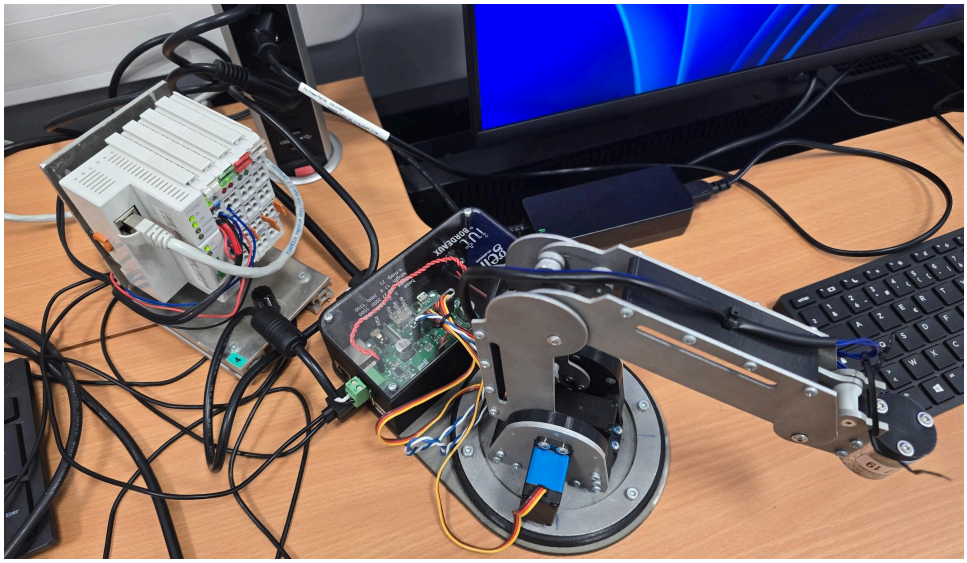
Compte Rendu de TP Informatique Industrielle



Travaux Pratiques Informatique Industrielle

Nom : HEBO Dionisio
Groupe : B1b

Sommaire

Introduction

Ce rapport présente les travaux pratiques réalisés dans le cadre du module d'Informatique Industrielle. L'objectif principal était de commander un bras robotique en programmant en C d'une carte Raspberry Pi connectée au bras.

Pour réaliser cela, nous avons suivi plusieurs étapes :

- **Gestion des Entrées/Sorties :** nous avons appris à accéder directement aux ports de la carte pour activer les composants.
- **Communication :** nous avons mis en place une communication série pour échanger des données entre l'ordinateur et la carte, d'abord avec un terminal puis avec une application en C.
- **Contrôle du robot :** nous avons programmé la carte pour recevoir des ordres et piloter les servomoteurs de la base, de l'avant-bras et de la pince.
- **Transmission de commandes :** enfin, nous avons développé une application pour envoyer des commandes précises avec des paramètres au robot.

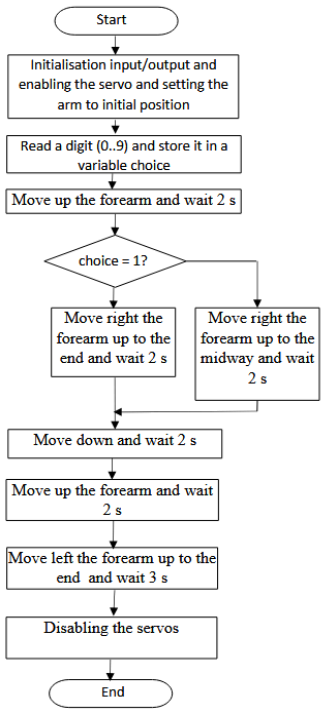
TP1 : Accessing the input/output of the Raspberry board

Pour le premier TP, j'ai commencé par importr les bibliothèques qui me seraient utiles pour tout ce projet et par faire la définition des toutes les variables plus importantes, les connexions des servomoteurs :

```
#include <stdio.h>
#include <stdlib.h>
#include <pigpio.h>

#define S1_EN       3
#define S2EN        17
#define S3_EN       22
#define EA_EN       23
#define S1_SIGNAL   2
#define S2_SIGNAL   4
#define S3_SIGNAL   27
```

Il nous a été proposé de développer un programme qui respecte cet algorithme ci dessous :



Cet algorithme dit que lors de l'initialisation du code, il faut initialiser les entrées et sorties et activer les servomoteurs. Après l'initialisation, nous voulons que le bras du robot prenne sa position initiale et puis, il faut lire un chiffre entre 0 et 9 et le stocker dans une variable qu'on va appeler "choix":

```
int choix;

    if (gpioInitialise() < 0){
        return -1;
    }
    gpioSetMode(S1_EN, PI_OUTPUT);
    gpioSetMode(S3_EN, PI_OUTPUT);

    gpioWrite(S3_EN, 1);
    gpioWrite(S1_EN, 1);
    //gpioWrite(S3_EN, 1);

    gpioServo(S3_SIGNAL, 2000);//faire descendre le bras
    gpioWrite(S3_EN, 0);
    //time_sleep(2);
    gpioServo(S1_SIGNAL, 1000); //rotation
    gpioServo(S2_SIGNAL, 2000); //arm
    time_sleep(0.5);
    gpioWrite(S3_EN, 0);
    gpioWrite(S1_EN, 0);

    gpioWrite(S3_EN, 1);//faire monter le bras
    gpioServo(S3_SIGNAL, 2000);
    gpioWrite(S3_EN, 0);
    time_sleep(2);
    printf("\n Votre choix =");
    scanf("%d", &choix);
```

Après cette variable "choix", il faut prendre une décision : si le contenu stocké dans la variable est égale à 1, le bras doit se mouvoir vers le haut au maximum et attendre deux secondes et sinon, le bras doit se mouvoir vers le milieu de son amplitude et attendre deux secondes :

```
if (choix ==1){
        gpioWrite(S1_EN, 1);
        gpioWrite(S3_EN, 1);
        gpioServo(S1_SIGNAL, 2000);
        gpioServo(S3_SIGNAL,1500);
        time_sleep(2);

    }
    else{
        gpioWrite(S3_EN, 1);
        gpioServo(S3_SIGNAL, 1500);
        time_sleep(1);
        gpioServo(S3_SIGNAL, 2000);
        time_sleep(1);
        gpioWrite(S3_EN, 0);

    }
```

Après l'une de ces deux actions, il faut que le bras descend, puis remonte et pour finir, il faut qu'il aille à gauche et après on arrête les servomoteurs, avec un intervalle de deux secondes entre chacune de ces actions :

```
    gpioWrite(S3_EN, 1);
    gpioServo(S3_SIGNAL, 2000);
    time_sleep(2);
    gpioServo(S3_SIGNAL, 1500);
    time_sleep(2);
    gpioWrite(S1_EN, 1);
    gpioServo(S1_SIGNAL, 1500);
    time_sleep(3);
    gpioServo(S3_SIGNAL, 1500);
    gpioWrite(S3_EN, 0);
    gpioWrite(S1_EN, 0);
    gpioTerminate();
```

Tout le code présenté jusqu'ici a été écrit à l'intérieur du fichier main, puis exécuté pour vérifier son fonctionnement.

TP2 : Receiving and transmitting data between two terminal applications

Pour ce TP, il nous a été demandé d'établir la connexion entre la carte Raspberry et le pc Windows en utilisant un **adaptateur USB/RS232** pour pouvoir transmettre l'information entre les deux.

Pour répondre à ce besoin, nous avons utilisé le software Hterm sur Windows que nous avons configuré avec un baud rate de 19200 bit/s, 8 bit d'information, 1 bit de stop et sans bit de parité.

De la partie Linux (Raspberry), il a fallu chercher le nom qu'il a donné à la liaison série à l'aide la commande **ls/dev/tty*** sur le terminal Linux. Pour faire de même sur Windows, nous avons utilisé le gestionnaire de périphériques pour trouver la port COM qui a été utilisée.
Sur Linux nous avons trouvé que le nom de la port série c'était **dev/ttyUSB0** et sur Windows c'était la port **COM4.**

Après avoir configuré chacun des éléments comme il nous a été demandé de faire, nous avons fait la transmission entre les équipements et voilà ce que nous avons obtenu :

1. Le code ASCII pour "P", "A" et "R" :
   P : 50
   A : 41
   R : 52
2. Les bytes pour les entiers non signés de 16 bits "24568" et "45678" :
   24568 : 00110101 01100110 01100110 00111000
   45678 : 01100010 00110010 00110110 01100101

TP3 : Receiving or transmitting data with an application developed in C language

Pour ce TP, il nous a été demandé de télécharger sur moodle une application qui sera utilisée pour transmettre des données entre la carte Raspberry et le système Windows.

Le nom de la port série utilisée dans le TP précédent est toujours utile pour ce TP. Il faut lire cette port et si la valeur contenue est inférieure à zéro, il faut arrêter le programme.

```
int fd;
    Input_Output_Initialization();
    signal(SIGINT, stop);
    /* Ouverture de la liaison serie */
    fd  = configserie((char*)"/dev/ttyUSB0");
    if (fd < 0)
    {
        getchar();
        return (EXIT_FAILURE);
    }
```

Quand on lit le code du fichier main, nous pouvons voir que le code a été conçu pour que l'application puisse recevoir 4 bytes et envoyer 3 bytes. De ces 4 bytes qu'il peut recevoir, il y en a un qui indique le nom du servomoteur à être commandé, 2 bytes qui correspondent à la valeur qui sera adressée au servomoteur et 1 byte null, et ces 4 bytes sont stockés dans une variable nommée "message".

```
printf("\nport  ouvert");
    do{
        int i;
        lireChaineCaracteres(fd, message);
        //lireChaineNbCaracteres(fd, message, 4);
        printf("\nmessage1 = ");

        for(i=0;i<4;i++)
        {
            printf("%02X ",message[i]);
        }
```

```
        printf("\n");

        fflush(stdout);
        if (RobotArmControl(message)){
            ack(fd);
        }
        else{
            nak(fd);
        }
    }
```

Les fonctions “ack (acknowledgement)” et “nak (non acknowledgement)” sont déclarées dans le fichier commserie.c , et non acknowledgement respectivement, sont utilisées pour dire si la commande exécutée est valide (ack) ou non (nak).

```
void ack(int fd){
    char ackMesage[] = {'A', 'C', 'K'};
    ecrire(fd, ackMesage, 3);
}
void nak(int fd){
    char nakMesage[] = {'N', 'A', 'K'};
    ecrire(fd, nakMesage, 3);
}
```

Après avoir vérifié les fichiers (main.c, frame_servo.c et commserie.c), nous avons construit l’application à l’aide de la commande “make” et lancé l’application et nous avons pu vérifier le bon fonctionnement de notre application.

TP4 : Controlling the servos

Dans ce TP nous devons faire une structure de décision pour savoir quel servomoteur sera manipulé et un mappage pour savoir quelles valeurs on va l'envoyer.

Pour répondre à ce besoin , il a fallu manipuler le fichier "frame_servo.c" et à l'intérieur de la fonction "RobotArmControl" créer la structure de décision, ce que j'ai choisi de faire avec un "switch-case" :

```
int RobotArmControl(unsigned char *requete, int fd){

/************************************           command           servo
*********************************/
    int result = 0;
    uint16_t valeur;
    uint16_t valeur_gpio;
```

Jusqu'ici, ce qu'on fait ce déclarer la fonction comme une fonction qui doit renvoyer un entier en prenant comme paramètres une chaîne de caractère et un entier. On définit aussi les variables "result (celle qui sera renvoyée à la fin de la fonction)", "valeur (celle qui va stocker la valeur contenue dans le paramètre requete)" et "valeur_gpio (celle qui va stocker le résultat du mappage).

Dans le code ci-dessous, je fais un switch avec l'information stockée dans la première case du tableau "requete" (information correspondante au nom du servomoteur à être utilisé) :

```
switch(requete[0])
{
    case 'R':
        //On concatène les indices 1 et 2 dans valeur
        valeur = (requete[1]<<8) | requete[2];
        //On mape de [0x0000, 0xFFFF)
        valeur_gpio = 1000 + (valeur*1000)/0xFFFF;
        printf("Rotation base : %08X mappé en %d\n", valeur, valeur_gpio);
        result = 1;
        gpioServo(S1_SIGNAL, valeur_gpio);
        break;
    case 'F':
        //On concatène les indices 1 et 2 dans valeur
```

```
        valeur = (requete[1]<<8) | requete[2];
        //On mape de [0x0000, 0xFFFF)
        valeur_gpio = 1500 + (valeur*500)/0xFFFF;
        printf("Forearm : %08X mappé en %d\n", valeur, valeur_gpio);
        result = 1;
        gpioServo(S3_SIGNAL, valeur_gpio);
        break;
    case 'A':
        //On concatène les indices 1 et 2 dans valeur
        valeur = (requete[1]<<8) | requete[2];
        //On mape de [0x0000, 0xFFFF)
        valeur_gpio = 1100 + (valeur*900)/0xFFFF;
        printf("A : %08X mappé en %d\n", valeur, valeur_gpio);
        result = 1;
        gpioServo(S2_SIGNAL, valeur_gpio);
        break;
    case 'E':
        //Pour l'électroaimant, nous pouvons avoir que deux informations
        //Soit il est actif (1), soit il n'est pas actif (0).
        valeur = requete[2];
        printf("electroiman valeur en %d\n", valeur);
        result = 1;
        gpioWrite(EA_EN, requete[2];);
        break;
    default:
        result = 0;
        gpioTerminate();
        break;
}
```

Dans le cas où l'information contenue dans la première position de “requete” ne correspondent à aucun des servomoteurs, on stocke zéro dans la variable “result” et on éteint les servomoteurs. Cette fonction est appelée dans le fichier “main.c” et si elle renvoie “1” on aura un message affiché disant que tout s’est bien passé et dans l’autre cas, on aura un message disant qu’il y a eu une erreur :

```
if (RobotArmControl(message)){
            ack(fd);
        }
        else{
            nak(fd);
        }
```

Pour vérifier le bon fonctionnement de notre code, j'ai construit l'application à l'aide de la commande "make" et j'ai vérifié en utilisant le HTerm travers une liaison série avec Wndows.

TP5 : Developing an application intended to transmit the commands with parameters

Pour ce TP il fallait faire un code Python sur Windows qui allait envoyer une trame et recevoir une autre comme réponse.
On a commencé par importr la librairie “serial” (il faut installer si elle ne l’est pas déjà) et par la déclaration des variables :

```
#the needed library
import serial
sendCommand = bytearray(4)
receivedData = bytearray(3)

sendCommand[0] = ord('...')
sendCommand[1] = 0xFF
sendCommand[2] = 0xFF
sendCommand[3] = 0x00
```

1. “sendCommand” stocke combien d'octets seront envoyés par notre code, création d’un tableur qui a 4 octets libres.
2. “sendCommand[0]” stocke le nom du servomoteur à être utilisé dans la première position du tableau, à l’intérieur de ord(‘...’) nous pouvons mettre une de ces trois options : A, F ou R.
3. “sendCommand[1]” et “sendCommand[2]” stockent les valeurs qui seront envoyées au servomoteur à être commandé dans la deuxième et troisième position du tableau , elles peuvent stocker de “0x00” à “0xFF” chacune. Une fois concaténées, ces valeurs peuvent donner une valeur entre 0 et 65535.
4. “sendCommand[3]” c’est un octet null, dans la quatrième position du tableau.
5. “receivedData” stocke combien d’octets le code attend dans la réponse, création d’un tableur qui a 3 octets libre.

Dans cette partie, on définit la variable “h1” associée à la port “COM4” qui travaille à une vitesse de 19200 bit/s et sans temps d’attente défini et après on envoie les 4 octets stockés dans la variable “sendCommand” :

```
# the programme is opening a virtual port defined over USB
h1 = serial.Serial("COM4",baudrate = 19200,timeout = None) # no timeout
print("transmitting for data")
h1.write(sendCommand) #each character is stored into a byte
```

Dans le code ci-dessous on reçoit les 3 octets attendues dans la variable “receivedData”, on les affiche et après le programme s’arrêt :

```
receivedData = h1.read(3)
print("Je reçois la réponse", receivedData)
h1.close()
```

TP6 et TP7 : Controlling the position of the prehensor

Pour ce TP, il fallait exploiter un fichier .csv pour créer une structure capable de stocker les données de ce tableau. Ce tableau sera utilisé dans une application à construire qui devra demander à l'utilisateur quelle position de ce tableau il veut utiliser et envoyer les données correspondantes à cette ligne au prehensor.
Pour répondre à ce cahier de charges, il a fallu commencer par créer un fichier "ik.c" où la structure serait créée. La structure est conçue de trois parties :

1. Une variable du type "int" qui s'appelle "pos" : variable qui a va indiquer la positions des données.
2. Une variable du type "float" qui s'appelle "forearm" : variable à envoyer à un des servomoteurs pour attendre la position souhaitée.
3. Une variable du type "float" qui s'appelle "arm" : variable à envoyer à un des servomoteurs pour attendre la position souhaitée.

Dans le code ci-dessous, on crée la structure et une variable du type de la structure que l'on vient de créer, on crée une fonction qui va lire le fichier ".csv" et attribuer des valeurs aux variables qui constituent la structure :

```
#include <stdio.h>
typedef struct{
    int pos;
    float forearm;
    float arm;
}controlAngles;

controlAngles target[110];

void assimile(void){
    FILE *pi = fopen("outAnglesFileV2.csv", "r");
    for (int i = 0; i < 100; i++){
        fscanf(pi, "%d, %f, %f", &target[i].pos, &target[i].forearm,
&target[i].arm);
    }
    fclose(pi);
}
```

Sur le fichier "main.c", on vient ajouter une partie qui demande à l'utilisateur de choisir un nombre et ce nombre est utilisé pour dire sur quelle position du tableau on va prendre les valeurs

qui seront envoyées aux servomoteurs mais ces valeurs ne seront pas envoyées telles qu'elles sont, elles vont passer par un mappage :

```
do{
    printf("Choisissez un nombre entre 100 et 199 : ");
    int pos;
    scanf("%d", &pos);
    int indice = recherche(pos);
          printf("Valeurs trouvées forearm = %f arm = %f\n",
target[indice].forearm, target[indice].arm);

    float angle_forearm = target[indice].forearm;
    float angle_arm =  target[indice].arm;

    float control_arm = -10.5*angle_arm + 2015;
    float control_forearm = -10.45*angle_forearm + 2005;

    ServoControl('F', control_forearm);
    ServoControl('A', control_arm);
    }
```

Il est possible de voir qu'une fonction dont on n'a pas encore parlé a été utilisée, la fonction recherche. Cette fonction est déclarée dans le fichier "ik.c" et elle cherche à voir si la position choisie par l'utilisateur existe vraiment ou pas. Dans le cas où elle existe, elle va renvoyer la position choisie par l'utilisateur et sinon, elle va renvoyer "-1", qui correspond à un problème trouvé :

```
int recherche(int pos){
    for (int i = 0; i < 100; i++){
        if (target[i].pos == pos){
            return i;
        }
    }
    return -1;
}
```

Après tout ça, j'ai créé l'application pour vérifier le fonctionnement du code.

Conclusion

Au fil de ces travaux pratiques, j'ai progressivement appris à commander un bras robotique à l'aide d'une Raspberry Pi et d'applications développées en C. Les premiers TPs m'ont permis de comprendre le fonctionnement des entrées/sorties, des servomoteurs et les bases de la communication série. J'ai ensuite mis en place un échange fiable de données entre la carte et un PC, puis développé une application capable de décoder des trames, mapper des valeurs et piloter chaque articulation du robot.

Les derniers TPs ont introduit la gestion de données externes via un fichier CSV et la création d'une structure permettant de contrôler précisément les positions du préhenseur. L'ensemble du projet m'a ainsi offert une vision complète du lien entre programmation bas niveau, communication série et contrôle matériel, tout en aboutissant à une application fonctionnelle et robuste que j'ai pu développer et tester moi-même du début à la fin.