File "/controlemoteur_bis/main.cpp" printed from os.mbed.com on 4/6/2022

```
/*
 * PROJET PROTIS 2021-2022
 * LEnsE / IOGS
 *
 * Flora Silberzan, Samuel Gerente, Etienne Loiselet, Florian Motyl,
 * Gregoire Guyon
 */

#include "mbed.h"


Serial RF(PC_10, PC_11); // Création de la liaison avec le module RF


/* DÉFINITION DES BROCHES DE COMMANDES DES MOTEURS EN TANT QUE SORTIES PWM */
PwmOut MOT1_D1(PA_10);
PwmOut MOT1_D2(PA_15); // Moteur 1

PwmOut MOT2_D1(PB_14);
PwmOut MOT2_D2(PB_13); // Moteur 2

PwmOut MOT3_D1(PB_8);
PwmOut MOT3_D2(PC_9); // Moteur 3



/* DÉFINITION DES BROCHES CONNECTÉES AUX ENCODEURS EN TANT QU'ENTRÉE NUMÉRIQUE.
 *
 * La sortie A de chaque encodeur sur un interruption et la B sur une entrée
 * classique
 */
InterruptIn M1_A(PB_0);
DigitalIn M1_B(PC_1); // Encodeur 1

InterruptIn M2_A(PB_2);
DigitalIn M2_B(PB_1); // Encodeur 2

InterruptIn M3_A(PB_4);
DigitalIn M3_B(PB_10); // Encodeur 3


/* DÉFINITION DES TICKER POUR L'ÉCHANTILLONNAGE DE L'ASSERVISEMENT DES MOTEURS */
Ticker ticker_M1;
Ticker ticker_M2;
Ticker ticker_M3;


/* PROTOTYPES DES FONCTIONS RELIÉES AUX INTERRUPTIONS SUR LES SIGNAUX DES
 * ENCODEURS ET PERMETTANT DE COMPTER LES IMPULSIONS
 *
 * On compte les fronts montants et descendants sur le signal A de chaque
 * encodeur. Le sens de rotation est obtenu en regardant l'état du signal B.
 */
void interrupt_M1_A_rise(void);
void interrupt_M1_A_fall(void);

void interrupt_M2_A_rise(void);
void interrupt_M2_A_fall(void);

void interrupt_M3_A_rise(void);
void interrupt_M3_A_fall(void);


/* PROTOTYPES DES FONCTIONS PERMETTANT DE FAIRE TOURNER LES MOTEURS D'UN NOMBRE
 * D'IMPULSION DONNÉE
 */
void move_M1(int consigne_N);
void move_M2(int consigne_N);
void move_M3(int consigne_N);


/* PROTOTYPES DES FONCTIONS CALCULANT L'ASSERVISSEMENT DES MOTEURS */
void asservissement_vitesse_M1();
void asservissement_vitesse_M2();
void asservissement_vitesse_M3();


/* PROTOTYPES DES FONCTIONS PERMETTANT DE FAIRE AVANCER ET TOURNER LE ROBOT */
void avancer(int distance);
void tourner_droite(int distance);
void tourner_gauche(int distance);


/* DÉFINITION DES VARIABLES UTILISÉES LORS DE L'ASSERVISEMENT DES MOTEURS */

int consigne_position_M1;          // Consigne de position du moteur 1
float consigne_vitesse_M1 = 0;     // Consigne de vitesse du moteur 1
int impulsion_M1 = 0;              // Nombre d'impulsion actuel du moteur 1
float erreur_vitesse_M1;           // Erreur de vitesse du moteur 1
int prev_impulsion_M1 = 0;         // Nombre d'impulsion à l'étape
                                   // d'asservissement précédente du moteur 1
float prev_erreur_vitesse_M1 = 0;  // Erreur de vitesse à l'étape
                                   // d'asservissement précédente du moteur 1
float commande_vitesse_M1;         // Commande de vitesse du moteur 1
float vitesse_M1;                  // Vitesse du moteur 1
float somme_erreur_vitesse_M1 = 0; // Somme des erreurs de vitesse du moteur 1
float percent_M1 = 0;              // Pourcentage d'avancement du moteur 1

```

```
 99
100 int consigne_position_M2;          // Consigne de position du moteur 2
101 float consigne_vitesse_M2 = 0;     // Consigne de vitesse du moteur 2
102 int impulsion_M2 = 0;              // Nombre d'impulsion actuel du moteur 2
103 float erreur_vitesse_M2;           // Erreur de vitesse du moteur 2
104 int prev_impulsion_M2 = 0;         // Nombre d'impulsion à l'étape
105                                    // d'asservissement précédente du moteur 2
106 float prev_erreur_vitesse_M2 = 0;  // Erreur de vitesse à l'étape
107                                    // d'asservissement précédente du moteur 2
108 float commande_vitesse_M2;         // Commande de vitesse du moteur 2
109 float vitesse_M2;                  // Vitesse du moteur 2
110 float somme_erreur_vitesse_M2=0;   // Somme des erreurs de vitesse du moteur 2
111 float percent_M2 = 0;              // Pourcentage d'avancement du moteur 2
112
113
114 int consigne_position_M3;          // Consigne de position du moteur 3
115 float consigne_vitesse_M3 = 0;     // Consigne de vitesse du moteur 3
116 int impulsion_M3 = 0;              // Nombre d'impulsion actuel du moteur 3
117 float erreur_vitesse_M3;           // Erreur de vitesse du moteur 3
118 int prev_impulsion_M3 = 0;         // Nombre d'impulsion à l'étape
119                                    // d'asservissement précédente du moteur 3
120 float prev_erreur_vitesse_M3 = 0;  // Erreur de vitesse à l'étape
121                                    // d'asservissement précédente du moteur 3
122 float commande_vitesse_M3;         // Commande de vitesse du moteur 3
123 float vitesse_M3;                  // Vitesse du moteur 3
124 float somme_erreur_vitesse_M3 = 0; // Somme des erreurs de vitesse du moteur 3
125 float percent_M3 = 0;              // Pourcentage d'avancement du moteur 3
126
127
128
129 int distance_case = 77600;       // Nombre d'impulsion correspondant à une case
130 int distance_quart_tour = 46194; // Nombre d'impulsion correspondant à quart
131                                  // de tour
132
133 float Te = 0.05; // Période d'enchantillonage de l'asservissement (en seconde)
134
135 char c; // Charactère pour la lecture sur liaison série
136
137
138 /*****
139  * PROGRAMME PINCIPAL
140  */
141 int main() {
142
143     /* LES FONCTIONS DE COMPTAGE D'IMPULSION SONT ATTACHÉES AU INTÉRRUPTIONS */
144     M1_A.rise(&interrupt_M1_A_rise);
145     M1_A.fall(&interrupt_M1_A_fall);
146
147     M2_A.rise(&interrupt_M2_A_rise);
148     M2_A.fall(&interrupt_M2_A_fall);
149
150     M3_A.rise(&interrupt_M3_A_rise);
151     M3_A.fall(&interrupt_M3_A_fall);
152
153
154     RF.baud(19200); // Régléage de la vitesse de la liaison série avec le module RF
155
156     /* Régléage de la période des signaux PWM commandant les moteurs (10ms)*/
157     MOT1_D1.period_ms(10);
158     MOT1_D2.period_ms(10);
159
160     MOT2_D1.period_ms(10);
161     MOT2_D2.period_ms(10);
162
163     MOT3_D1.period_ms(10);
164     MOT3_D2.period_ms(10);
165
166     /* BOUCLE INFINI */
167     while(1)
168     {
169         // Si un caractère est disponible à la lecture dans le buffer de la
170         // liaison série avec le module RF, on rentre dans cette boucle
171         if(RF.readable()) {
172
173             c = RF.getc(); // Le caractère est lu et stocké dans la variable "c"
174
175             // En fonctin du caratère lu, on fait différente action
176             switch(c) {
177                 case 'a': // Avancer 1 case
178                     avancer(distance_case);
179                     break;
180                 case 'b': // Avancer 2 case
181                     avancer(2*distance_case);
182                     break;
183                 case 'c': // Avancer 3 case
184                     avancer(3*distance_case);
185                     break;
186                 case 'e': // Avancer 4 case
187                     avancer(4*distance_case);
188                     break;
189                 case 'h': // Avancer 5 case
190                     avancer(5*distance_case);
191                     break;
192                 case 'i': // Avancer 6 case
193                     avancer(6*distance_case);
194                     break;
195                 case 'j': // Avancer 7 case
196                     avancer(7*distance_case);
197                     break;
198                 case 'k': // Avancer 8 case
199
```

```
                    avancer(8*distance_case);
                    break;
                case '1': // Avancer 9 case
                    avancer(9*distance_case);
                    break;
                case 'm': // Avancer 10 case
                    avancer(10*distance_case);
                    break;
                case 'n': // Avancer 11 case
                    avancer(11*distance_case);
                    break;
                case 'o': // Avancer 12 case
                    avancer(12*distance_case);
                    break;
                case 'p': // Avancer 13 case
                    avancer(13*distance_case);
                    break;
                case 'q': // Avancer 14 case
                    avancer(14*distance_case);
                    break;
                case 'r': // Avancer 15 case
                    avancer(15*distance_case);
                    break;

                case 'd': // Quart de tour à droite
                    tourner_droite(distance_quart_tour);
                    break;
                case 'g': // Quart de tour à gauche
                    tourner_gauche(distance_quart_tour);
                    break;
                case '&': // Demi tour
                    tourner_droite(2*distance_quart_tour);
                    break;

                case '?': // Demande pour renvoyer l'avancement
                    // Calcul de l'avancement (en pourcentage) de l'action en cours
                    int avancement = int(percent_M2*100);

                    // On renvoie sur la liaison série un caractère codant
                    // l'avancement
                    if(avancement <= 10) {RF.putc('0');}
                    else if (avancement > 10 && avancement <= 20) {RF.putc('1');}
                    else if (avancement > 20 && avancement <= 30) {RF.putc('2');}
                    else if (avancement > 30 && avancement <= 40) {RF.putc('3');}
                    else if (avancement > 40 && avancement <= 50) {RF.putc('4');}
                    else if (avancement > 50 && avancement <= 60) {RF.putc('5');}
                    else if (avancement > 60 && avancement <= 70) {RF.putc('6');}
                    else if (avancement > 70 && avancement <= 80) {RF.putc('7');}
                    else if (avancement > 80 && avancement <= 90) {RF.putc('8');}
                    else if (avancement > 90 && avancement < 100) {RF.putc('9');}
                    else if(avancement >= 100) {RF.putc('f');}
                }
        }
    }
}

/*****
 * Les fonctions suivante sont appelées lors des fronts montant et descendant
 * sur le signal A des encodeurs et permettent de calculer le nombre d'impulsion
 * parcouru par chaque moteur
 *
 * Le sens est determiner par la l'état du signal B (cf doc technique).
 */
void interrupt_M1_A_rise()
    {
       if(M1_B == 1) { impulsion_M1++; }
       else { impulsion_M1--; }
    }

void interrupt_M1_A_fall()
    {
       if(M1_B == 0) { impulsion_M1++; }
       else { impulsion_M1--; }
    }

void interrupt_M2_A_rise()
    {
       if(M2_B == 1) { impulsion_M2++; }
       else { impulsion_M2--; }
    }

void interrupt_M2_A_fall()
    {
       if(M2_B == 0) { impulsion_M2++; }
       else { impulsion_M2--; }
    }

void interrupt_M3_A_rise()
    {
       if(M3_B == 1) { impulsion_M3++; }
       else { impulsion_M3--; }
    }

void interrupt_M3_A_fall()
    {
       if(M3_B == 0) { impulsion_M3++; }
       else { impulsion_M3--; }

    }
```

```
/*****
 * Fonctions permettant de faire avancer chaque moteur d'un nombre d'impulsion
 * donné. Ce nombre peu etre positif on négatif en fonction du sens de rotation.
 */
void move_M1(int consigne_impulsion)
    {
    // Chargement de la consigne d'impulsion
    consigne_position_M1 = consigne_impulsion;

    // Réinitialisation des varaibles nécessaire à l'asservissement
    impulsion_M1 = 0;
    prev_impulsion_M1 = 0;
    somme_erreur_vitesse_M1 = 0;
    prev_erreur_vitesse_M1 = 0;

    // La fonction d'asservissement est attachée au Ticker pour être appelée à
    // intervalle régulier
    ticker_M1.attach(&asservissement_vitesse_M1, Te);
    }

void move_M2(int consigne_impulsion)
    {
    // Chargement de la consigne d'impulsion
    consigne_position_M2 = consigne_impulsion;

    // Réinitialisation des varaibles nécessaire à l'asservissement
    impulsion_M2 = 0;
    prev_impulsion_M2 = 0;
    somme_erreur_vitesse_M2 = 0;
    prev_erreur_vitesse_M2 = 0;

    // La fonction d'asservissement est attachée au Ticker pour être appelée à
    // intervalle régulier
    ticker_M2.attach(&asservissement_vitesse_M2, Te);
    }

void move_M3(int consigne_impulsion)
    {
    // Chargement de la consigne d'impulsion
    consigne_position_M3 = consigne_impulsion;

    // Réinitialisation des varaibles nécessaire à l'asservissement
    impulsion_M3 = 0;
    prev_impulsion_M3 = 0;
    somme_erreur_vitesse_M3 = 0;
    prev_erreur_vitesse_M3 = 0;

    // La fonction d'asservissement est attachée au Ticker pour être appelée à
    // intervalle régulier
    ticker_M3.attach(&asservissement_vitesse_M3, Te);
    }


/*****
 * avancer : Fonction permettant de faire avancer le robot en ligne droite sur
 * un nombre d'impulsion donné
 */
void avancer(int impulsion)  {
    move_M1(impulsion);
    move_M2(impulsion);
    }

/*****
 * tourner_droite : Fonction permettant de faire tourner vers la gauche le robot
 * sur un nombre d'impulsion donné
 */
void tourner_droite(int impulsion) {
    move_M1(impulsion);
    move_M2(-impulsion);
    move_M3(impulsion);

    }

/*****
 * tourner_droite : Fonction permettant de faire tourner vers la gauche le robot
 * sur un nombre d'impulsion donné
 */
void tourner_gauche(int impulsion)   {
    move_M1(-impulsion);
    move_M2(impulsion);
    move_M3(-impulsion);
    }


/* PARAMÈTRE DE L'ASSERVISSEMENT PID EN VITESSE */
float Kp = 0.00001;  // Proportionnel
float Ki = 0.000005; // Intégral
float Kd = 0.001;    //Dériation


float p = 0.3; // Variable permettant de définir la consigne de vitesse en
               // trapèze (non fonctionnel)

/*****
 * Fonctions permettant l'asservissement des moteurs
 * Les trois moteurs ont une fonction similaire, seul celle du moteur 1 est
 * commentée
 */
void asservissement_vitesse_M1()  {
```

```
    // Calcul du pourcentage d'avancement dans sa consigne du moteur
    percent_M1 = 1.0*impulsion_M1/consigne_position_M1;

    // Calcul de la consigne de vitesse suivant une loi sinusoidale.
    // Il y a un offset (10000) car le moteur commence à tourner pour une
    // consigne de l'ordre de 10000.
    // Le signe de la commande est convervé.
    consigne_vitesse_M1 = - consigne_position_M1/abs(consigne_position_M1) * (10000+sin(3.14159265*abs(percent_M1))*50000);

    //Calcul de la consigne de vitesse suivant une loi trapézoïdal
    //(non fonctionnel)

    //if(abs(percent_M1) <= p) { consigne_vitesse_M1 = -consigne_position_M1/abs(consigne_position_M1) * (10000 + percent_M1*50000/p); }
    //if(abs(percent_M1) > p && abs(percent_M1) <= 1-p) { consigne_vitesse_M1 = -consigne_position_M1/abs(consigne_position_M1) * (10000 + 50000*1); }
    //if(abs(percent_M1) > p) { consigne_vitesse_M1 = -consigne_position_M1/abs(consigne_position_M1) * (10000 + 50000*(1-percent_M1)/p); }

    // Calcul de la vitesse du moteur (en impulsion/seconde)
    vitesse_M1 = (impulsion_M1 - prev_impulsion_M1)/Te;
    prev_impulsion_M1 = impulsion_M1; //Stockage du nombre d'impulsion

    // Calcul de l'erreur de vitesse
    erreur_vitesse_M1 = -vitesse_M1 + consigne_vitesse_M1;

    // Stockage de l'erreur de vitesse pour la partie dériavation du PID
    prev_erreur_vitesse_M1 = erreur_vitesse_M1;

    // Calcul de la somme des erreurs de vitesse (pour la partie Intégration du PID)
    somme_erreur_vitesse_M1 += erreur_vitesse_M1;

    // Calcul de la commande de vitesse, asservissement PID
    commande_vitesse_M1 = Kp*erreur_vitesse_M1 + Ki*somme_erreur_vitesse_M1 + Kp*(erreur_vitesse_M1-prev_erreur_vitesse_M1);

    // Quand le pourcentage d'avancement arrive à 1, le moteur est arrêté et
    // l'asservisement arrêté
    if(abs(percent_M1) >= 1) {
        commande_vitesse_M1 = 0;
        ticker_M1.detach();
        }

    // En fonction du signe de la commande, le moteur est mis en rotation dans
    // un sens ou l'autre
    if(commande_vitesse_M1 >= 0) {
        // La commande maximale est 1
        if(commande_vitesse_M1 > 1) {   commande_vitesse_M1 = 1;    }

        // Commande du moteur
        MOT1_D1.write(commande_vitesse_M1);
        MOT1_D2.write(0);
        }

    if(commande_vitesse_M1 < 0) {
        // Inversion du signe de la commande car la commande .write prend
        //un argument positif
        commande_vitesse_M1 = -commande_vitesse_M1;

        // La commande maximale est 1
        if(commande_vitesse_M1> 1 ) {   commande_vitesse_M1 = 1;    }

        // Commande du moteur
        MOT1_D1.write(0);
        MOT1_D2.write(commande_vitesse_M1);
        }
}


void asservissement_vitesse_M2()  {

    percent_M2 = 1.0*impulsion_M2/consigne_position_M2;

    consigne_vitesse_M2 = consigne_position_M2/abs(consigne_position_M2) * (10000+sin(3.14159265*abs(percent_M2))*50000);

    //if(abs(percent_M2) <= p) { consigne_vitesse_M2 = consigne_position_M2/abs(consigne_position_M2) * (10000 + percent_M2*50000/p); }
    //if(abs(percent_M2) > p && abs(percent_M2) <= 1-p) { consigne_vitesse_M2 = consigne_position_M2/abs(consigne_position_M2) * (10000 + 50000*1); }
    //if(abs(percent_M2) > p) { consigne_vitesse_M2 = consigne_position_M2/abs(consigne_position_M2) * (10000 + 50000*(1-percent_M2)/p); }

    vitesse_M2 = (impulsion_M2 - prev_impulsion_M2)/Te;
    prev_impulsion_M2 = impulsion_M2;

    erreur_vitesse_M2 = -vitesse_M2 + consigne_vitesse_M2;
    somme_erreur_vitesse_M2 += erreur_vitesse_M2;

    commande_vitesse_M2 = Kp*erreur_vitesse_M2 + Ki*somme_erreur_vitesse_M2 + Kp*(erreur_vitesse_M2-prev_erreur_vitesse_M2);

    if(abs(percent_M2) >= 1) {
        commande_vitesse_M2 = 0;
        ticker_M2.detach();
        }

    prev_erreur_vitesse_M2 = erreur_vitesse_M2;

    if(commande_vitesse_M2 >= 0) {
        if(commande_vitesse_M2 > 1) {   commande_vitesse_M2 = 1;    }

        MOT2_D1.write(commande_vitesse_M2);
        MOT2_D2.write(0);
        }

    if(commande_vitesse_M2 < 0) {
        commande_vitesse_M2 = -commande_vitesse_M2;
```

```
        if(commande_vitesse_M2 > 1 ) {    commande_vitesse_M2 = 1;     }

        MOT2_D1.write(0);
        MOT2_D2.write(commande_vitesse_M2);
        }
}


void asservissement_vitesse_M3()  {

    percent_M3 = 1.0*impulsion_M3/consigne_position_M3;

    consigne_vitesse_M3 = - consigne_position_M3/abs(consigne_position_M3) * (10000+sin(3.14159265*abs(percent_M3))*50000);

    //if(abs(percent_M3) <= p) { consigne_vitesse_M3 = -consigne_position_M3/abs(consigne_position_M3) * (10000 + percent_M3*50000/p); }
    //if(abs(percent_M3) > p && abs(percent_M3) <= 1-p) { consigne_vitesse_M3 = -consigne_position_M3/abs(consigne_position_M3) * (10000 + 50000*1); }
    //if(abs(percent_M3) > p) { consigne_vitesse_M3 = -consigne_position_M3/abs(consigne_position_M3) * (10000 + 50000*(1-percent_M3)/p); }

    vitesse_M3 = (impulsion_M3 - prev_impulsion_M3)/Te;

    prev_impulsion_M3 = impulsion_M3;

    erreur_vitesse_M3 = -vitesse_M3 + consigne_vitesse_M3;
    somme_erreur_vitesse_M3 += erreur_vitesse_M3;

    commande_vitesse_M3 = Kp*erreur_vitesse_M3 + Ki*somme_erreur_vitesse_M3 + Kp*(erreur_vitesse_M3-prev_erreur_vitesse_M3);

    if(abs(percent_M3) >= 1) {
        commande_vitesse_M3 = 0;
        ticker_M3.detach();
        }

    prev_erreur_vitesse_M3 = erreur_vitesse_M3;


    if(commande_vitesse_M3 >= 0) {
        if(commande_vitesse_M3 > 1) {    commande_vitesse_M3 = 1;     }

        MOT3_D1.write(commande_vitesse_M3);
        MOT3_D2.write(0);
        }

    if(commande_vitesse_M3 < 0) {
        commande_vitesse_M3 = -commande_vitesse_M3;
        if(commande_vitesse_M3> 1 ) {    commande_vitesse_M3 = 1;     }

        MOT3_D1.write(0);
        MOT3_D2.write(commande_vitesse_M3);
        }
}
```

File "/controlemoteur_bis/main.cpp" printed from os.mbed.com on 4/6/2022