Lab. #5 : Régulateur de processus à microcontrôleur

II. But du laboratoire et matériel requis

Ce laboratoire vise à montrer comment un microcontrôleur peut servir à réguler un processus industriel. Deux types de commandes seront mises en œuvre : binaire (dite aussi oui-non) et PID (proportionnel-dérivatif-intégral). Un objectif secondaire est de montrer comment réaliser une interface de puissance avec isolation de masse.

Matériel requis:

- Carte d'extension MSP-EXP430FR6989 sans le microcontrôleur **ou** kit de composants équivalents, plaquette de montage, et fils de raccordement.
- Moteur électrique à courant continu avec interface optoélectronique
- Kit de développement CY8CKIT-059 PSoC 5LP
- Oscilloscope

I. Introduction

Les microcontrôleurs sont souvent utilisés comme régulateurs de processus. Les variables pouvant être régulées incluent la température, l'intensité lumineuse, la vitesse de rotation, ou tout autre paramètre physique d'intérêt. Il existe deux types fondamentaux de commande pour atteindre le but : la commande en boucle ouverte et la commande en boucle fermée. La première génère un signal de commande sans vérifier si le résultat désiré sera atteint; la commande en boucle fermée utilise un capteur et une boucle de contre-réaction pour mesurer le résultat et adapter la commande en fonction de l'erreur entre l'effet désiré et celui obtenu.

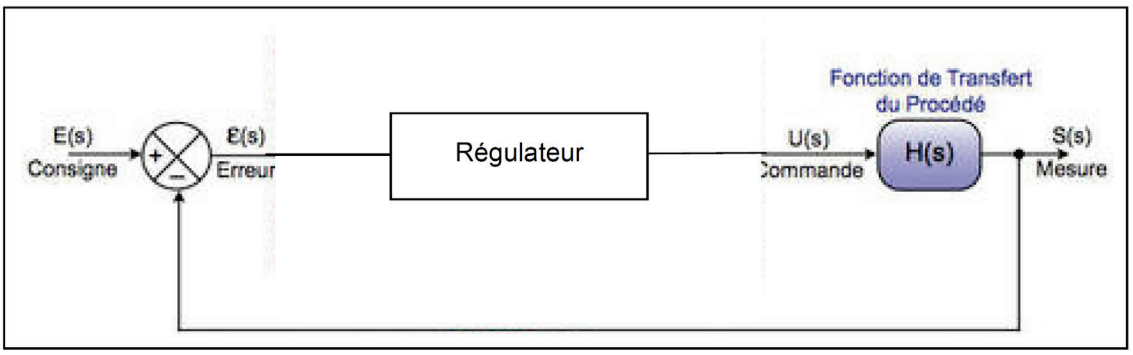

Figure 1. Système de commande en boucle fermée

L'algorithme de régulation en boucle fermée le plus populaire est l'algorithme PID (proportionnel-intégral-dérivatif). Il est applicable à tout processus pouvant être modelé par un système linéaire de premier ou second ordre. Le correcteur PID fait intervenir trois termes de correction, un qui réagit à l'erreur instantanée (terme proportionnel), un qui tient compte de l'historique de l'erreur (terme intégral) et un qui réagit aux fluctuations récentes de l'erreur (terme dérivatif) ; son équation est donc :

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

où K_p, K_i et K_d sont les gains proportionnel, intégral et dérivatif, respectivement.

Si on discrétise le temps avec une fréquence d'échantillonnage f_e qui est très grande devant la fréquence de Nyquist des signaux traités, l'équation précédente peut être écrite :

$$u(k) = K_p e(t) + K_i \sum_{j=0}^{k} e(k) + K_d \frac{e(k) - e(k-1)}{T_e}$$

Où l'indice k représente le temps t=kT_e

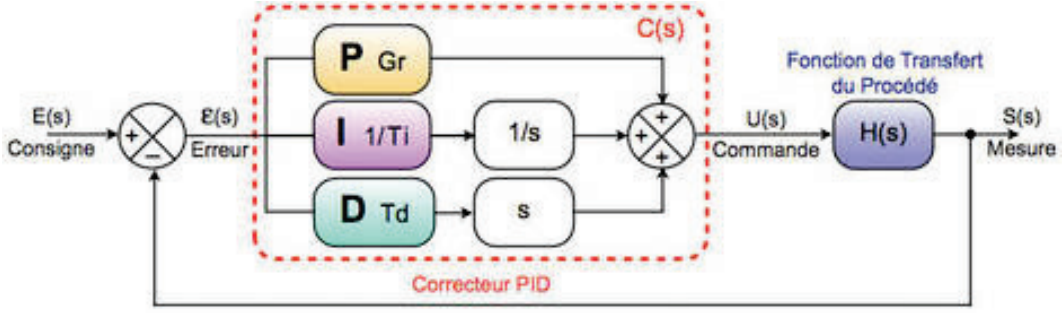

Figure 2. Système de commande PID

Le terme dérivatif est souvent la source de comportements oscillatoires lorsqu'il est mal ajusté et on l'omet souvent.

Le choix des gains K_p, K_i et K_d se fait de manière à optimiser la vitesse de réponse, le temps de réponse et l'amplitude des dépassements par rapport à la valeur désirée (point de consigne) du paramètre régulé (voir Figure 3). Il n'existe malheureusement pas de formule miracle pour y arriver, car les trois gains ne sont pas indépendants.

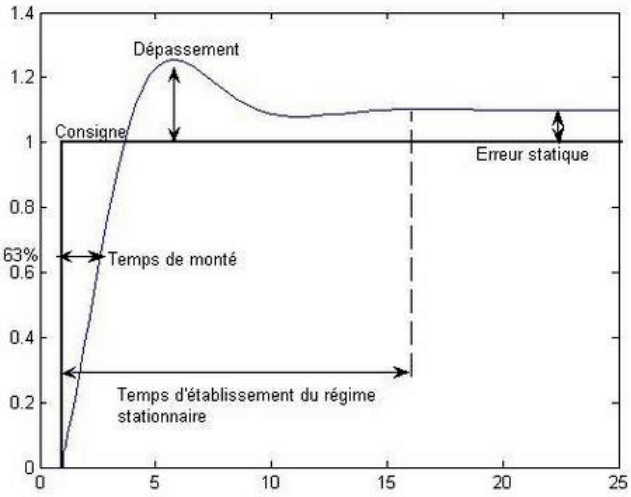

Figure 3: Réponse typique à un effort de correction

Le Tableau suivant résume l'impact de chacun des trois gains sur la réponse :

Effet d'augmenter un gain				
Gain	Temps de montée	Dépassement	Temps de réponse	Erreur due à une variation permanente de charge
K_p	baisse	augmente	Petit changement	diminue
K_i	baisse	augmente	augmente	éliminée
K_d	Indéfini	diminue	augmente	nil

Une méthode de réglage heuristique, due à Ziegler–Nichols, est comme suit : On initialise K_i and K_d à zéro, et on augmente K_p jusqu'à une valeur critique, K_c, à laquelle la réponse commence à osciller. K_c et la période d'oscillation P_c sont alors utilisés pour régler les gains comme suit :

Type de commande	K_p	K_i	K_d
P	$0.50K_c$	-	-
PI	$0.45K_c$	$1.2K_p / P_c$	-
PID	$0.60K_c$	$2K_p / P_c$	$K_pP_c / 8$

II. Manipulations

Le montage dont le schéma suit sert à stabiliser la vitesse de rotation d'un moteur. Cela est fait en mesurant la vitesse de rotation du moteur à l'un d'un coupleur optique dont le chemin lumineux est interrompu régulièrement par une un disque troué fixé à l'arbre du moteur. Une impulsion en forme de cloche est générée par le photodétecteur du coupleur optique à chaque révolution du moteur. Elle est passée à travers un inverseur à seuil qui la transforme en impulsion carrée et alimente une ligne de capture du MSP430. Le temps de capture de deux impulsions successives permet de déterminer la vitesse de rotation instantanée et l'information sert à mesurer l'erreur par rapport à la vitesse de rotation désirée. L'erreur est alors utilisée pour calculer la correction du PID qui servira à moduler une ligne de sortie PWM du microcontrôleur. L'information est transmise à un coupleur optique pour l'isolation de masse et la sortie de ce dernier alimente la base d'un transistor qui sert d'interface de puissance au moteur. À l'origine, le cycle de travail de la sortie PWM est fixé à 50%.

L'exemple de code joint réalise la fonction PID décrite précédemment pour un microcontrôleur MSP430F6989 de Texas Instrument. Il opère avec un point de consigne correspondant à une vitesse de rotation de 1400 tours/minute.

1) Étudier le code et en déduire le mode de correction mis en œuvre; donner le/les gains correspondants.
2) Modifier le code pour le PSOC 5LP. Il faudra créer un projet comme suit :

a. Ajouter dans le canevas un composant "Counter" et une broche d'entrée numérique pour relier l'horloge du compteur à la sortie OUT de l'interface moteur (sortie de l'un des inverseurs schmidt sur la carte de l'interface).
b. Ajoutez un composant "PWM" dans le canevas et une broche de sortie numérique pour relier le PWM à l'entrée IN de l'interface moteur (entrée de l'un des inverseurs schmidt sur la carte de l'interface).
c. Configurer le compteur en mode capture et interruptions, et régler la période des interruptions pour déclencher la logique de commande du PWM déjà présente dans le code pour MSP430.

3) Construire et exécuter le code en modifiant la tension d'alimentation ou la charge du moteur afin de changer sa vitesse de rotation instantanée, et vérifier avec un oscilloscope que le PID ramène la vitesse à sa valeur de consigne à chaque fois.
4) Modifier le code pour mettre en œuvre une commande oui-non et commenter son fonctionnement en comparaison au mode précédent

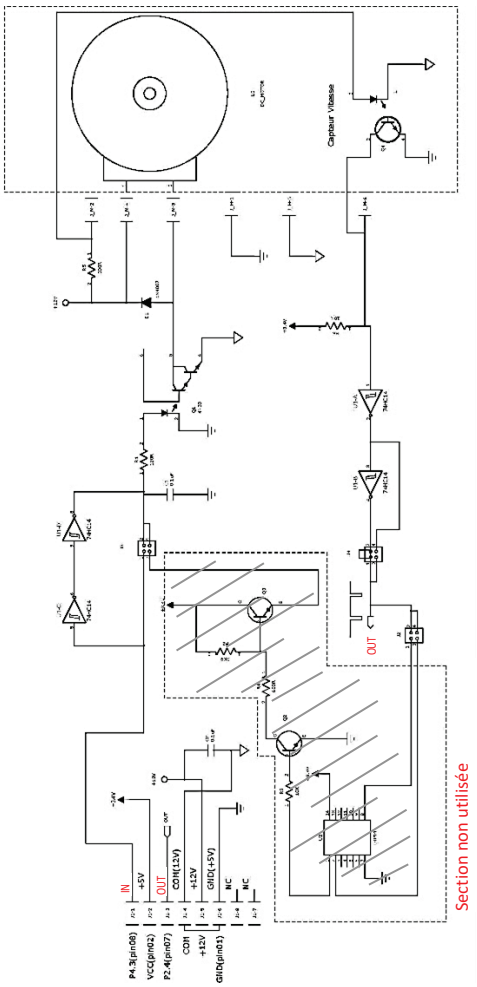

Schéma de l'interface moteur

Exemple de code pour un microcontrôleur MSP430

```
//*******************************************************************************
// MSP430FR6989 - Contrôle PDI
//
// Description: Programme de contrôle de la vitesse de rotation d'un moteur CC
//
// le moteur C.C. est contrôlé par PWM (Timer_A0 en mode TA0.1, broche P1.6)
// La vitesse du moteur en RPM (révolutions/minute) est mesurée grâce aux
// impulsions d'un capteur optoélectronique qui génère une impulsion par révolution
// à un compteur (Timer_B0 en mode capture et interruptions, broche P2.0;.
// Les interruptions sont aux 36 ms et L'horloge est SMCLK par le DCO par défaut de
// ~1.12MHz
//                MSP430FR6989
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//            |         P2.0/TB2|<-- CCI0B ROTATION PULSE INPUT
//            |         P1.6/TA2|--> TACCR2 - PWM OUTPUT TO MOTOR
//
//*******************************************************************************
```

```c
#include "msp430x22x4.h"

long err=0, temp0, errprev=0, outcont=0, pfactor=0, ifactor=0;
long dfactor=0, accerr=2400, setpoint=5000, rpmspeed=0xFFFF;
unsigned int temp=0, readRPM=0, tbase, pwmout ;

// Comptes d'horloge à 140 kHz (1.12 Mhz avec pré-diviseur  div8)
#define setpoint    (6000) // 140000/6000=23.3 tour/s = point de consigne(x60=1400 RPM)
#define setpointmax (4200) // 33.3 tour/sec (x60 = 1998 RPM)
#define setpointmin (10000)// 14 tour/sec (x60 = 840 RPM)
#define deadband    (100)  // Temps mort (deadband)
#define VitMax   (2096)    // 4000RPM @ 140Khz(timerB clk)
#define VitMin   (56000)   // 150 RPM @ 140Khz(timerB clk)
// taux de modulation PWM avec smclk = 1.12Mhz
#define pwmmax (512)       // 100% ratio PWM à ~2kHz (1.12M/512)
//Gains
#define pgain (10)         // Gain proportionnel
#define dgain (0)          // Gain dérivatif
#define igain (1)          // Gain intégral


void delay(unsigned int delai){
  while (--delai) ;
}

void main(void){

   PM5CTL0 &= ~LOCKLPM5;      // réactiver les lignes d'e/s au cas où
   WDTCTL = WDTPW | WDTHOLD;  // Stopper le chien de garde

   // configuration des horloges
   CSCTL0_H = CSKEY >> 8;          // Unlock CS registers
   CSCTL1 = DCOFSEL_0;             // set DCO to 1.12 MHz
   CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK; // set sources
   CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;                 // set dividers by 1
   CSCTL0_H = 0;

// LED témoin à P1.0
   P1DIR |= 0x01;

// Initialisation de Timer A en mode TA0.1 (PWM), et P1.6 en sortie
    P1DIR  |= BIT6;
    P1SEL0 |= BIT6;
    P1SEL1 |= BIT6;

    TA0CCR0 = pwmmax - 1;           // Période PWM = 2Khz => impulsion 500uSec
    TA0CCR1 = 256;                  // TA0CCR1=[0 512]=> PWM =[0 100]% ; 50% maintenant
    TA0CCTL1 = OUTMOD_7;            // Reset/set
    TA0CTL = TASSEL__SMCLK | MC__UP;// SMCLK used, UP_mode
    delay(65000);                   // Délai de démarrage

// Initialisation de Timer_B0 en mode capture avec interruptions (mesure du délai entre
// impulsions de rotation -mesure RPM-, P2.0 en entrée
    P2DIR &= ~BIT0;         // P2.0 en entrée pour les impulsions de rotation
    P2SEL0 &= ~BIT0;        // P2.0 TB0.CCI6B options
    P2SEL1 |= BIT0;
```

```
    // Compteur à zéro, SMCLK, comptage en continu, div/8 (horloge à 140kHz)
    TB0CTL = TBCLR | TBSSEL__SMCLK | MC__CONTINUOUS | ID__8 ;
    // Rising edge, CCI6B, Capture, Synchronize, Interruptions
    TB0CCTL6 = CM_1 | CCIS_1 | CAP | SCS | CCIE;
    TB0CCTL6 &= ~CCIFG;     //Clear Interrupt flag

//Initialisation de la base de temps pour générer des interruptions avec T=36 ms
    TB0CCR4 = 5000;      // (SMCLK/8)/5000 => 140KHZ/5000 => 7.14uSec*5000 = 0.0357 sec
    TB0CCTL4 = CCIE;     // Interruption de Timer B Activée

// Interruptions autorisées. Mode low-power 0 en attendant
    __bis_SR_register(LPM0_bits | GIE);
}

// Vecteur d'interruption pour Timer_B
#pragma vector=TIMERB1_VECTOR
__interrupt void TBX_ISR(void)
{
  switch (TBIV)              // Traitementde la source
    {
     case  TB0IV_TB0CCR4:    // Base de temps pour les corrections aux 36 ms
          TBCCR1 += 5000;     // ajouter 36 ms au compteur pour la prochaine interruption
          P1OUT ^= 0x01;      // Inverser la LED témoin (P1.0)
          // Algorithme de correction
          if (rpmspeed < VitMax) rpmspeed = VitMax; //  S'assurer des limites de vitesse
          if (rpmspeed > VitMin) rpmspeed = VitMin;

          err = rpmspeed - setpoint;   // - erreur
          // temp0 = abs(err);          // Dead band
          // if (temp0 < deadband) err = 0;

          // Calcul de la correction p
          pfactor = pgain * err;
          if (pfactor > 32767)pfactor = 32767;    // s'assurer des limites
          if (pfactor < -32768)pfactor = -32768;

          // Calcul de la correction d
          dfactor = dgain * (err - errprev);
          errprev = err;

          // Calcul de la correction i
          accerr = accerr + err;               // Calcul de l'erreur accumulée
          if (accerr > 32767) accerr = 32767;  // Limitation
          if (accerr < -32768) accerr = -32768;
          ifactor = igain * accerr;            // correction i
          if (ifactor > 32767)ifactor = 32767; // Limitation
          if (ifactor < -32768)ifactor = -32768;

          // Correction totale
          outcont = ifactor;
          outcont += dfactor;
          outcont += pfactor;
          if (outcont > 32767) outcont = 32767; // Limitation
          if (outcont < 0) outcont = 0;
          outcont=outcont>>6;  // projection dans [0 512] => cycle de travail [0 100]%
          TACCR2 = outcont>>6; // mise à jour du cycle de travail du PWM
```

```
        break;

     case TB0IV_TB0CCR6: // Impulsion reçue du capteur sur P2.0
         rpmspeed = TB0CCR6 - temp; // Mesure de la vitesse RPM (délai entre impulsions)
         temp = TB0CCR6;
         break;

    default: // On ne devrait jamais se rendre ici
         break;

    }

}
```