Classe de problèmes MECATRO-3

Réguler en vitesse un moteur à courant continu. Implanter un correcteur dans la régulation.

Analyser la structure d'une chaîne fonctionnelle
Générer un programme et l'implanter dans le système cible
Modifier un programme pour faire évoluer le comportement du système
Choisir un type de correcteur adapté

Modéliser - Simuler - Expérimenter

L'objectif de ce Tp est d'obtenir une mesure de la vitesse de rotation du moteur pour construire un asservissement de vitesse.

1 Asservissement en vitesse du moteur

1.1 Objectif

Objectif : évaluer la vitesse de rotation du moteur à l'aide des signaux fournis par les codeurs.

1.2 Fonction d'interruption

Pour réaliser l'asservissement en vitesse du moteur, il faut connaître la vitesse de rotation du moteur. On compare alors la consigne à la mesure de la vitesse de rotation du moteur et on élabore un signal écart. Cet écart est parfois corrigé/amplifié et donne l'ordre au préactionneur de moduler l'énergie à destination de l'actionneur.

Nous avons donc besoin dans un premier temps de mesurer la vitesse de rotation du moteur. Or le moteur possède deux codeurs magnétiques en quadrature de phase. Compter les passages devant le codeur magnétique permet d'obtenir une information sur la vitesse de rotation du moteur.

Un programme Arduino se compose de deux fonctions essentielles : `setup` et `loop`. La fonction `setup` est appelée une seule fois lorsque le programme commence et la fonction `loop` est appelée en permanence.

Il est possible d'arrêter brièvement et temporairement la fonction `loop` à chaque fois qu'un événement (changement d'état d'une entrée TOR) se produit. Pour cela il faut utiliser les broches numériques d'interruption, les broches 2 et 3, numérotées 0 et 1.

On propose ici un programme simple pour prendre en main la fonction `attachInterrupt(num, fonction, quand)`. Avec un bouton poussoir, on souhaite allumer ou éteindre une LED beaucoup plus simplement qu'au premier TP, avec l'aide de ce tuto. Le code suivant est téléchargeable ici :

```
1 int Led = 13; // Pin de la LED
2 int etat = LOW; // Etat de la LED
3
4 void setup()
```

```
{
 pinMode(Led, OUTPUT);
 attachInterrupt(0, clignote, RISING); // Lors d'un changement de son etat, on
    attache a la broche 2 la fonction clignote
 Serial.begin(9600);
}

void loop() {
  digitalWrite(Led, etat);
  Serial.println(etat);
}

void clignote() // fonction liee a l'interruption externe 0
{
  etat = !etat; // prend le complementaire
}
```

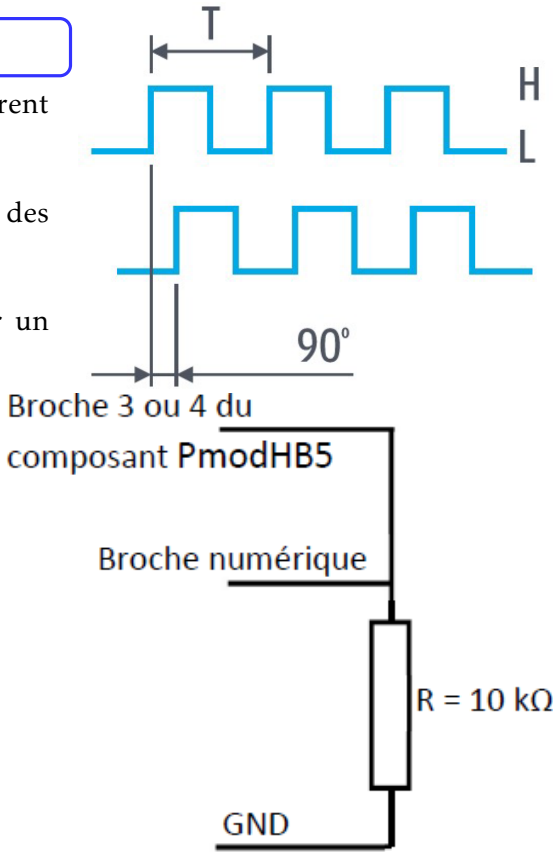
On note que la fonction `clignote` est définie en dehors du `setup` et du `loop`. Elle ne doit pas avoir d'argument et ne rien renvoyer.

1.3 Codeurs magnétiques et compteur

Le motoréducteur est équipé de deux codeurs magnétiques qui délivrent chacun un signal de 0 ou 5 V.

Ces codeurs sont décalés de 90°, le graphe ci-contre donne le profil des signaux délivrés lorsque le moteur tourne.

Les broches 3 et 4 délivrent ces signaux. On cherche ici à réaliser un compteur à partir des codeurs magnétiques sur l'arbre moteur.



Les broches 3 et 4 du composant PmodHB5 délivrent chacune un signal de 0 ou 5 V. Il faut donc relier chacune de ces broches à une broche entrée TOR de la carte Arduino de manière à pouvoir compter le nombre d'impulsions à des intervalles de temps réguliers.

De cette façon, on peut déterminer la vitesse de rotation du moteur. Pour un seul codeur, si on compte les fronts montants et descendants du signal, et que l'on a évalué 6 impulsions en 10 ms, cela veut dire que le moteur effectuerait 3 tours en 10 ms et donc qu'il tournerait à 300 tr/s soit 18 000 tr/min. Pour 2 codeurs, il y aura 4 impulsions par tour.

A chaque changement d'état des signaux des codeurs, on va incrémenter un compteur.

REMARQUE : Le fait d'avoir deux codeurs peut permettre de détecter le sens de rotation. On peut aussi décrémenter le compteur.

À chaque fois que la fonction `comptageA` est appelée, le niveau (haut ou bas) de la voie A sera comparé au niveau de la voie B. En fonction de cela, on détectera le sens de rotation et on incrémentera ou décrémentera le

compteur d'impulsion. On peut faire alors de même avec une fonction `comptageB` sur le niveau de la voie B. On gagne ainsi en résolution.

- Effectuer le câblage des broches 3 et 4 du composant PmodHB5 avec les broches d'interruption de la carte ;
- Définir les broches numériques 2 et 3 en tant que variables `signA` et `signB` ;
- Définir une variable entière `Nimpulsion` et l'initialiser à 0 ;
- Écrire deux fonctions `comptageA` et `comptageB` permettant d'incrémenter ou de décrémenter un compteur d'impulsion (ces fonctions seront placées avant la fonction `setup` et seront appelées à chaque interruption) ;

Le code est accessible par ce lien.

```
#include "digitalWriteFast.h"

// Affectation des broches
const int Pin_EN = 10;
const int Pin_DIR = 8 ;
const int Pin_Cap_A = 2;
const int Pin_Cap_B = 3;


// Variables
long vitesse = 255 ;
long Nimpulsion = 0;


void setup() {
  pinMode(Pin_EN, OUTPUT);
  pinMode(Pin_DIR, OUTPUT);
  pinMode(Pin_Cap_A, INPUT);
  pinMode(Pin_Cap_B, INPUT);
  Serial.begin(115200);
  attachInterrupt(0, comptageA, CHANGE);
  digitalWrite(Pin_EN, vitesse);
}

void loop(){
  Serial.println(Nimpulsion);
}


void comptageA(){
  if (boolean digitalReadFast(Pin_Cap_A) == boolean digitalReadFast(Pin_Cap_B)){
    Nimpulsion = Nimpulsion - 1;
  }
  else {
    Nimpulsion = Nimpulsion + 1;
  }
}
```

1.4 Mesure numérique de la vitesse

Il faut maintenant calculer la vitesse de rotation du moteur.

Pour cela nous avons besoin d'un `timer` qui déclenchera un calcul toutes les 10 millisecondes. En effet, puisqu'on sait incrémenter/décrémenter un compteur grâce aux capteurs magnétiques, il est possible de compter le nombre d'incréments reçus sur un temps donné.

Pour un seul codeur, si on compte les fronts montants et descendants du signal, et que l'on a évalué 6 impulsions en 10 ms, cela veut dire que le moteur effectuerait 3 tours en 10 ms et donc qu'il tournerait à 300 tr/s soit 18 000 tr/min. Pour 2 codeurs, il y aura 4 impulsions par tour.

Le fabricant annonce une vitesse en sortie de réducteur de 150 tr/min à vide sous une tension de 6 V avec une réduction de 52,734 ce qui correspond à une vitesse moteur de 7910 tr/min soit 131,8 tr/s. Pour un ordre de commande de hacheur de 255 et une alimentation de 6 V le moteur tournera donc à 7910 tr/min.

Comme nous allons alimenter le hacheur avec 9 V, le moteur devrait pouvoir tourner à presque 200 tr/s. Nous essaierons de réguler la vitesse à 100 tr/s.

Il convient donc maintenant d'obtenir une mesure numérique de la vitesse en tr/s.

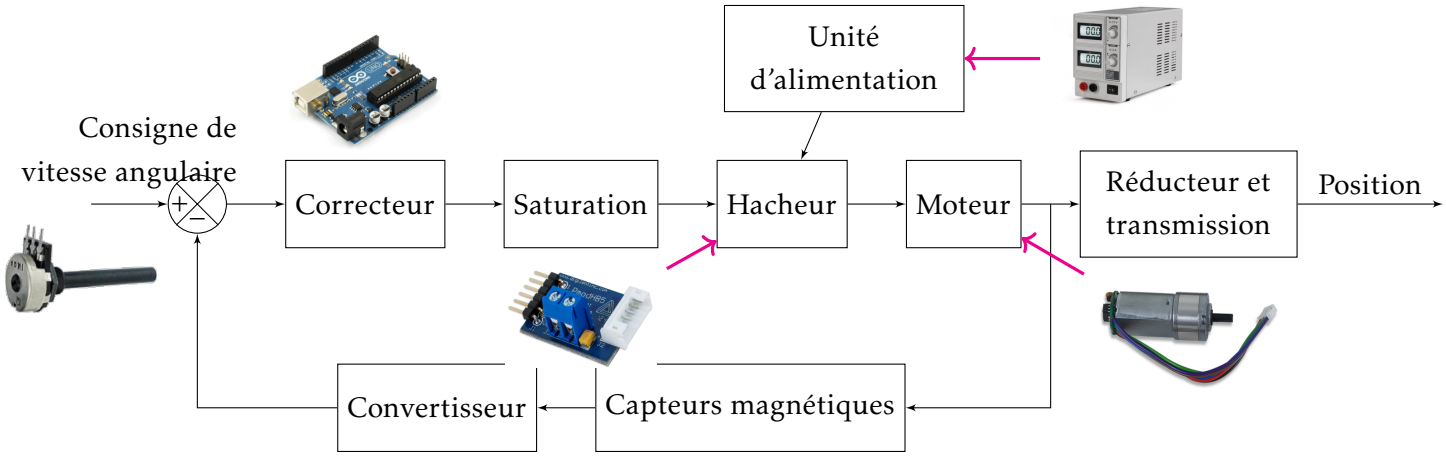
Chaque tour du moteur permet d'obtenir 4 incréments grâce aux fonctions d'interruption `ComptageA` et `ComptageB`. Ainsi `Nimpulsion/4` permet d'obtenir le nombre de tour de moteur sur la période de comptage.

Comme nous allons lancer une fonction `calcul_vitesse` toutes les 10 ms (tous les 100ièmes de seconde), il suffira de multiplier par `100` `Nimpulsion/4` pour connaître la vitesse en tr/s. Le code est ici.

```
#include <digitalWriteFast.h>
#include <MsTimer2.h>

// Affectation des broches
const int Pin_EN = 10;
const int Pin_DIR = 8 ;
const int Pin_Cap_A = 2;
const int Pin_Cap_B = 3;

// Variables
long mes_vit = 0 ;
long Nimpulsion = 0;


void setup() {
  pinMode(Pin_EN, OUTPUT);
  pinMode(Pin_DIR, OUTPUT);
  pinMode(Pin_Cap_A, INPUT);
  pinMode(Pin_Cap_B, INPUT);
  digitalWrite(Pin_EN, rap_cyc);
  Serial.begin(115200);
  attachInterrupt(0, comptageA, CHANGE);
```

```
23   attachInterrupt(1, comptageB, CHANGE);
24   MsTimer2::set(10, calcul_vitesse); // 500ms period
25   delay(10);
26   MsTimer2::start();
27 }
28
29 void loop(){
30   analogWrite(Pin_EN, 255);
31 }
32
33
34 void comptageA(){
35   if (boolean digitalReadFast(Pin_Cap_A) == boolean digitalReadFast(Pin_Cap_B)){
36     Nimpulsion = Nimpulsion - 1;
37   }
38   else {
39     Nimpulsion = Nimpulsion + 1;
40   }
41 }
42
43 void comptageB(){
44   if (boolean digitalReadFast(Pin_Cap_A) == boolean digitalReadFast(Pin_Cap_B)){
45     Nimpulsion = Nimpulsion + 1;
46   }
47   else {
48     Nimpulsion = Nimpulsion - 1;
49   }
50 }
51
52
53 void calcul_vitesse(){
54   mes_vit = 100*Nimpulsion/4;
55   Serial.println(mes_vit);
56   Serial.println("␣");
57   Nimpulsion = 0;
58 }
```

1.5 Ecart - correction



Grâce à la partie précédente, on connait vitesse de rotation du moteur en tr/s. Il est maintenant possible de piloter le hacheur (`Pin_EN`) en fonction de l'écart entre la consigne de vitesse et la mesure.

On peut dans un premier temps choisir un correcteur proportionnel K_p. Ainsi l'écart de vitesse, pourra être corrigé pour donner une valeur entre 0 et 255 au préactionneur. Cette valeur règle le rapport cyclique du hacheur et permet, en moyenne, de faire varier la tension d'alimentation du moteur.

```
void asservissement(){
  mes_vit = 100*Nimpulsion/4;
  Nimpulsion = 0;
  ecart = consigne_vit - mes_vit; // Calcul de l'ecart
  rap_cyc = int(kp*255/200*ecart); // Consigne pour hacheur
  if (rap_cyc > 255){
    rap_cyc = 255; // saturation
  }
  if (rap_cyc < 0) {
    rap_cyc = 0; // EN positif. pour aller en arriere il faut changer Dir
  }
  Serial.println("Rapport cyclique :");
  Serial.println(rap_cyc);
  analogWrite(Pin_EN, rap_cyc);
}
```

2 Visualisation sous Python

Si on souhaite tracer l'évolution de la vitesse en fonction du temps la démarche est plus compliquée. Il faut stocker les données envoyées vers le PC et les traiter. Pour cela il faut un outil logiciel capable de lire les données sur la liaison série. On peut écrire un programme Python qui peut récupérer les informations provenant de la liaison série, les stocker sous forme de liste et les exploiter.

La liaison série ne transmet que des octets. Il faut donc coder les infos au niveau de la carte Arduino avant de les envoyer et les décoder au niveau du programme Python. Mais attention, en fonction de son type, une variable peut être codées sur 1, 2 ou 4 octets

char	1 o
int	2 o
float	1 o

Ici il y a plusieurs solutions :

- soit on envoie le nombre d'impulsion correspondant à la variable `Nimpulsion` de type `int` donc 2 octets
- soit on envoie directement la vitesse évaluée correspondant à la variable `mes_vit` de type `float` donc 4 octets.

Dans le premier, cas il faudra refaire le calcul mais ce calcul peut être fait après saisie, ce qui permet de gagner un peu de temps. La courbe représentant l'évolution ne sera donc pas un tracé en temps réelle.

```
void envoi(){
  Nimpulsion0 = lowByte(Nimpulsion); // Premier octet de Nimpulsion
  Nimpulsion1 = highByte(Nimpulsion); // octet suivant de Nimpulsion
  Serial.write(Nimpulsion0);
  Serial.write(Nimpulsion1);
}
```

Il faut donc définir deux nouvelles variables de type `byte` (octet) et écrire une fonction `envoi` qui va écrire les deux octets sur la liaison série. Cette fonction envoi sera intégrée à la fonction asservissement.
Les deux octets seront lus et réassociés par le programme Python à l'aide du module « struct » (Voir programme python en ressource sur le drive et édité ci-dessous).

```
import serial
import struct
import time
import matplotlib.pyplot as plt

serie = serial.Serial("COM3", 115200)
time.sleep(2)

serie.flushInput()
duree = 3
serie.write(bytes("d", 'UTF-8'))
temps_init = time.time()
Te = 0.01
ltemps = []
lomega = []
inctemps = 0
print("Debut acquisition")

while (time.time() - temps_init < duree):
    motrecu = serie.read(2)
    motdecode = struct.unpack('h', motrecu)
    Nimpulson = motdecode[0]
    omega = Nimpulson * 100 / 4
    lomega.append(omega)
    ltemps.append(inctemps * Te)
    inctemps = inctemps + 1

print("Acquisition terminee")
serie.write(bytes("s", 'UTF-8'))
time.sleep(0.1)
serie.flushInput()
serie.close()

# Presentation des resultats
fig1=plt.figure()
plt.title("Evolution de la vitesse")
plt.xlabel('temps en s')
plt.ylabel('w(t) en trs/s')
plt.plot(ltemps, lomega, color='g')
plt.show(block = False)
```

Pour synchroniser le programme Python avec la carte Arduino, on va définir une procédure de début et de fin de prise d'informations. Pour cela on va écrire le code ci-dessous dans la fonction `loop` en définissant au préalable une nouvelle variable `c` de type `char`.

```
1  void loop(){
2    if (Serial.available() != 0){
3      c = (char)Serial.read();
4      if (c == 'd'){
5        MsTimer2::start();
6      }
7      else{
8        MsTimer2::stop();
9      }
10   }
11 }
```

Une fois le programme python édité et lancé, le caractère `d` est envoyé vers la carte via la liaison série. Après une période de 3 s, c'est le caractère `s` qui est envoyé.

Si le buffer de la liaison série au niveau de la carte arduino contient un octet, on lit cet octet et si cet octet correspond au caractère `d`, on démarre le *timer*. Dès qu'il y aura un octet correspondant à un autre caractère, le *timer* sera stoppé.

La courbe représentant l'évolution de la vitesse sera alors tracée.