Sommaire

Objectifs

- Savoir Implémenter un correcteur PID Numérique (Modèle du 2nd Ordre)
- Savoir régler les paramètres de son correcteur
- Savoir optimiser les performances de son correcteur
- Arduino & Problème de la Saturation (Divergence)
- Etc.

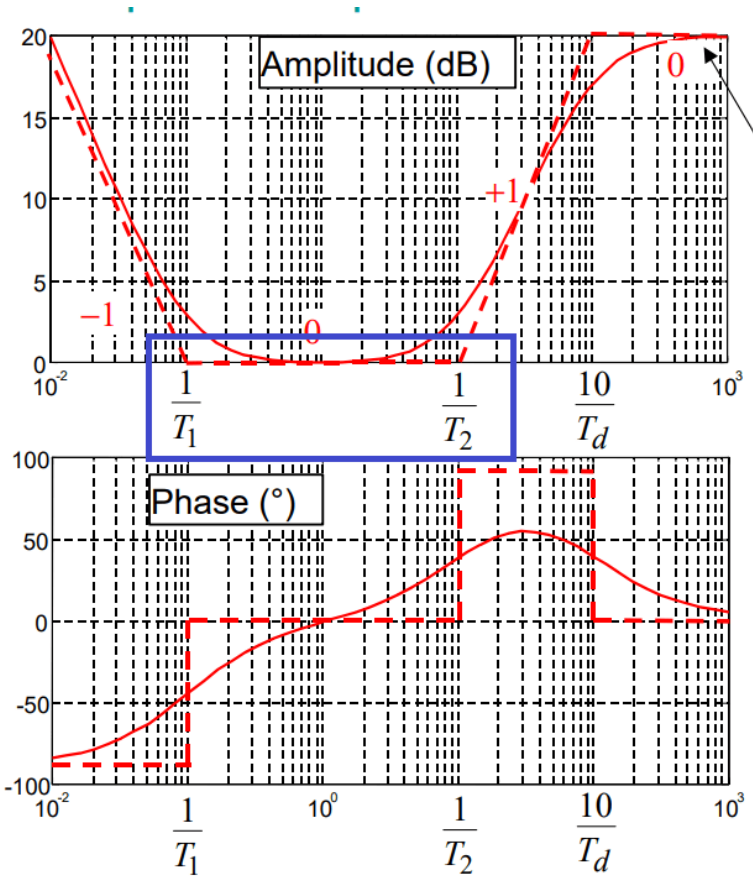
Fonctionnement

Le tuto a pour objectif d'implémenter et régler efficacement les paramètres d'un correcteur PID numérique réel. Contrairement à un PID théorique, le PID numérique dispose d'un filtre passe-bas en HF afin de limiter les perturbations. En revanche, la bande passante du filtre est directement liée à l'efficacité de l'effet dérivateur. Autrement dit, la stabilité du système (voir le tuto pour plus de détails).

Comment régler les paramètres du

correcteur PID ?

Il existe plusieurs techniques de réglage des paramètres d'un correcteur PID, en particulier la méthode de Ziegler-nichols. Dans ce tuto on verra une astuce qui va vous permettre de régler les paramètres du correcteur facilement en se basant sur un modèle du second ordre du correcteur. En effet, le correcteur sera décrit par une équation du 2nd ordre muni du filtre en HF. Le coefficient d'amortissement du PID doit être supérieur ou égal à 1. Il agit sur la largeur de bande de fréquence dédiée à l'intégrateur ainsi le dérivateur (voir le tuto pour des détails).



1

$$C(s) = K_c \frac{\left(1 + \frac{1}{T_i s} + T_d s\right)}{\left(1 + \frac{T_d}{N}\right)}$$

Filtrage des hautes fréquences

En pratique, la pulsation de coupure du système physique Wn (ou sa bande

> *passante) doit être connue d'avance afin d'en déduire les paramètres du réglage du correcteur PID. La réponse à un échelon en boucle ouverte peut être utilisée afin de mesurer Wn en se basant sur le temps de montée, le dépassement, la fréquence des oscillations, etc. Dans notre tuto la fréquence du système est fixée à 5Hz, zeta à 0.05.*

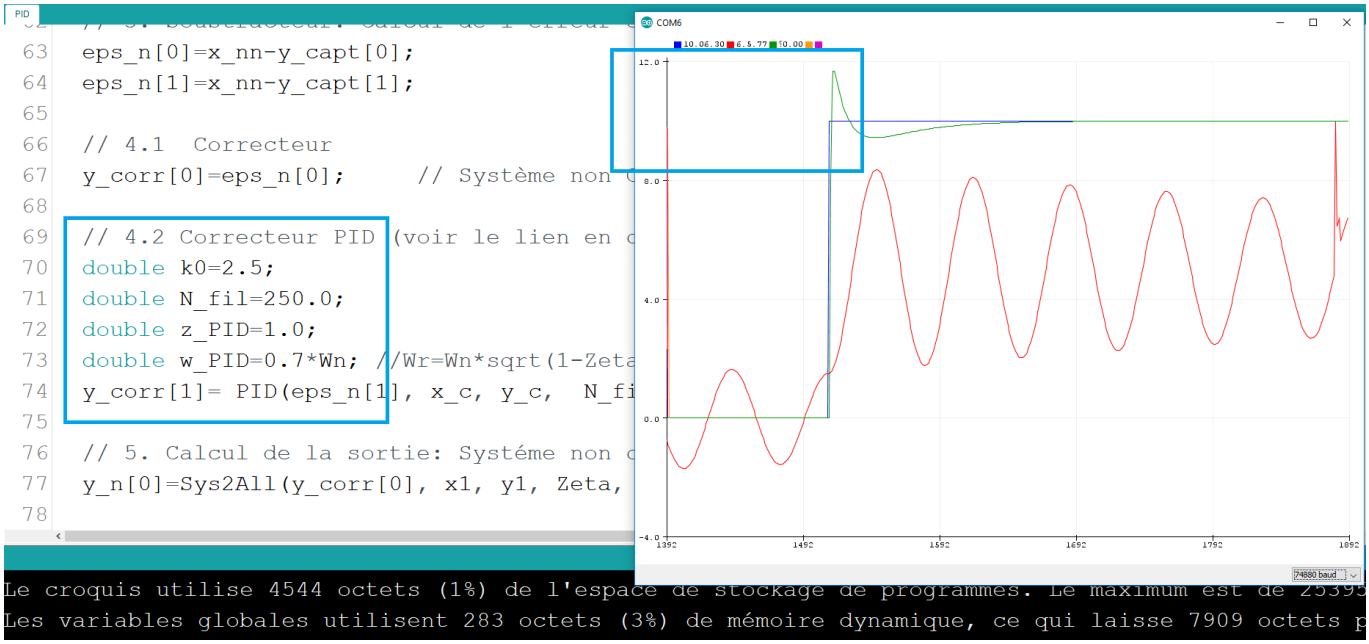
La fonction PID()

La fonction PID() sert à implémentée le modèle numérique du correcteur PID. Elle prend en entrée la sortie du soustracteur (ou l'erreur epsilon(n)), le paramètre N est lié à la bande passante du filtre HF, la période d'échantillonnage ainsi les paramètres du correcteur PID : 1. La coefficient d'amortissement zêta du correcteur au voisinage de 1 (>=1), 2. La pulsation de coupure du correcteur (au voisinage du celui du système physique Wn) de préférence <Wn et 3. La contrastante k0 ou l'action proportionnelle du correcteur PID. En suite la fonction retourne la sortie du correcteur. Ci-dessous la définition de la fonction PID().

```
double PID(double x_nn, double *x_cc, double *y_cc, double N_fil, double z, double
w, double k, double T)
{
  double t1=1.0/(w*(z+sqrt(z*z-1.0)));
  double t2=1.0/(w*(z-sqrt(z*z-1.0)));

  double td=t1+t2;
  double ti=t1*t2/td;

  /*
  Serial.print(t1);
  Serial.print("\t");
  Serial.print(t2);
  Serial.print("\t");
  Serial.print(td);
  Serial.print("\t");
  Serial.println(ti);
  */
  double a1=2.0*z/w;
```

```
  double a2=1.0/(w*w);
  double b1=ti;
  double b2=ti*t2/N_fil;

  double b[3];
  b[0]= a1/(2*T)+a2/(T*T);
  b[1]= 1.0-2.0*a2/(T*T);
  b[2]=-a1/(2*T)+a2/(T*T);

  double c[3];
  c[0]= b1/(2*T)+b2/(T*T);
  c[1]= -2.0*b2/(T*T);
  c[2]= -b1/(2*T)+b2/(T*T);
  // Variables de l'entrée et la sortie
  double y_nn=0.0;

  // Calcul de la nouvelle sortie
  y_nn=k*(b[0]*x_nn + b[1]*x_cc[0] + b[2]*x_cc[1]);
  y_nn=y_nn-(c[1]*y_cc[0] + c[2]*y_cc[1]);
  y_nn=y_nn/c[0];
  // Mise à jour de la sortie
  y_cc[1]=y_cc[0];
  y_cc[0]=y_nn;

  // Mise à jour de la sortie
  x_cc[1]=x_cc[0];
  x_cc[0]=x_nn;
  // Renvoie du résultat
  return y_nn;
}
```

> *Ci-dessous un exemple du test du correcteur PID d'un système oscillant.*



Le programme Arduino

```
/*
 * Objectifs:
 * 1. Savoir Implémenter un correcteur PID Numérique (Modèle du 2nd Ordre)
 * 2. Savoir régler les paramètres de son correcteur
 * 3. Savoir optimiser les performances de son correcteur
 * 4. Arduino & Problème de la Saturation (Divergence)
 */

#define   Fn      5.00
#define   Zeta    0.05 //0.70710678118
#define   K       1.0
#define   T_ms    2

#define   A_step  10.0    // Amplitude
#define   c_step  500     // Période = 2*c_step*T_ms

double Wn=2.0*PI*Fn;
double T_s=(double)T_ms/1000.0;
```

```
double x_nn=0.0;      // Consigne (entrée)
double y_n[2];        // "0" Non corrigé, "1": Corrigé
double eps_n[2];      // Erreur
double y_capt[2];    // Sortie du capteur
double y_corr[2];    // Sortie du correcteur

// Variables internes des systèmes
double x1[2], y1[3]; // Système Non Corrigé
double x2[2], y2[3]; // Système Corrigé

// Variables internes du correcteur
double x_c[2], y_c[2];

// Paramètres de l'échelon
unsigned long c=0; // Compteur (période)
bool Step=false;


void setup()
{
  // Port série de la réponse du système
  Serial.begin(19200);
}

void loop()
{
  // 1. La consigne (l'entrée) x(n) pour les deux systèmes
  c++; c=c%c_step;
  if(!c)
  {
    Step=!Step;
    c=0;
  }
  x_nn=A_step*(double)Step; // Réponse à un échelon x(n)=cte
  //x_nn=(double)c;           // Réponse à une rampe x(n)=n
  // 2. Sortie du capteur: Retour unitaire
  y_capt[0]=y_n[0];
  y_capt[1]=y_n[1];
  // 3. Soustracteur: Calcul de l'erreur eps(n)
  eps_n[0]=x_nn-y_capt[0];
  eps_n[1]=x_nn-y_capt[1];
  // 4.1  Correcteur
  y_corr[0]=eps_n[0];     // Système non Corrigé
```

```
  // 4.2 Correcteur PID (voir le lien en commentaire)
  double k0=2.5;
  double N_fil=250.0;
  double z_PID=1.0;
  double w_PID=0.7*Wn; //Wr=Wn*sqrt(1-Zeta*Zeta);
  y_corr[1]= PID(eps_n[1], x_c, y_c,  N_fil,  z_PID,  w_PID,  k0, T_s);
  // 5. Calcul de la sortie: Systéme non corrigé
  y_n[0]=Sys2All(y_corr[0], x1, y1, Zeta, Wn, K, T_s);

  // 5. Calcul de la sortie: Systéme corrigé
  y_n[1]=Sys2All(y_corr[1], x2, y2, Zeta, Wn, K, T_s);

  // Affichage des signaux
  Serial.print(x_nn); Serial.print(",");
  Serial.print(y_n[0]); Serial.print(",");
  Serial.println(y_n[1]);

  // Période d'échantillonnage
  delay(T_ms);
}



double Sys2All(double x_nn, double *x, double *y, double zeta, double wn, double k,
double T)
{
  // Paramètre du système
  double a1=2.0*zeta/wn;
  double a2=1.0/(wn*wn);

  const double b0=(a1/(2.0*T))+(a2/(T*T));
  const double b1=-2.0*a2/(T*T);
  const double b2=(-1.0*a1/(2.0*T))+(a2/(T*T));
  const double b[3]={b0,b1,b2};

  // Variables de l'entrée et la sortie
  double y_nn=0.0;
  // Calcul de la nouvelle sortie
  y_nn= -(y[0]*(1.0+b[1]))-(y[1]*b[2])+(k*x[0]); // y[1]: y(n-2), y[0]: y(n-1)
  y_nn/=b[0];
  // Mise à jour de la sortie
  y[1]=y[0];
  y[0]=y_nn;
```

```
  // Mise à jour de la sortie
  x[0]=x_nn;
  // Renvoie du résultat
  return y_nn;
}

double CorrPI(double x_nn, double *xpi, double *ypi, double kp, double ki, double T)
{
  // Variables de l'entrée et la sortie
  double y_nn=0.0;
  // Calcul de la nouvelle sortie
  y_nn=ypi[1] + kp*x_nn + (2.0*T*ki)*xpi[0] -kp*xpi[1];
  // y(n)=y(n-2)+ k1*x(n) + 2*T*k2*x(n-1) - k1*x(n-2)
  // Mise à jour de la sortie
  ypi[1]=ypi[0];
  ypi[0]=y_nn;

  // Mise à jour de la sortie
  xpi[1]=xpi[0];
  xpi[0]=x_nn;
  // Renvoie du résultat
  return y_nn;
}

double CorrPD_AP(double x_nn, double *x_cc, double *y_cc, double a, double t0,
double k0, double T)
{
  // Variables de l'entrée et la sortie
  double y_nn=0.0;

  // Paramètres du correcteur
  double alfa=a*t0;
  double beta=t0;
  // Calcul de la nouvelle sortie
  // Modèle Analogique: C(p)= k*(1+aTp)/(1+Tp)
  // Modèle numérique:  y(n)=[alfa*k 2kT -alfa*k]*[x(n) x(n-1) x(n-2)]'
  //                        -[2T -beta]*[y(n-1) y(n-2)]'
  //                    y(n)=y(n)/beta
  y_nn=(alfa*k0*x_nn)+(2*k0*T*x_cc[0]) - (alfa*k0*x_cc[1]);
  y_nn=y_nn- (2*T*y_cc[0]) - (beta*y_cc[1]);
  y_nn/=beta;
  // Mise à jour de la sortie
  y_cc[1]=y_cc[0];
  y_cc[0]=y_nn;
```

```
  // Mise à jour de la sortie
  x_cc[1]=x_cc[0];
  x_cc[0]=x_nn;
  // Renvoie du résultat
  return y_nn;
}

double PID(double x_nn, double *x_cc, double *y_cc, double N_fil, double z, double
w, double k, double T)
{
  double t1=1.0/(w*(z+sqrt(z*z-1.0)));
  double t2=1.0/(w*(z-sqrt(z*z-1.0)));

  double td=t1+t2;
  double ti=t1*t2/td;

  /*
  Serial.print(t1);
  Serial.print("\t");
  Serial.print(t2);
  Serial.print("\t");
  Serial.print(td);
  Serial.print("\t");
  Serial.println(ti);
  */
  double a1=2.0*z/w;
  double a2=1.0/(w*w);
  double b1=ti;
  double b2=ti*t2/N_fil;

  double b[3];
  b[0]= a1/(2*T)+a2/(T*T);
  b[1]= 1.0-2.0*a2/(T*T);
  b[2]=-a1/(2*T)+a2/(T*T);

  double c[3];
  c[0]= b1/(2*T)+b2/(T*T);
  c[1]= -2.0*b2/(T*T);
  c[2]= -b1/(2*T)+b2/(T*T);
  // Variables de l'entrée et la sortie
  double y_nn=0.0;

  // Calcul de la nouvelle sortie
  y_nn=k*(b[0]*x_nn + b[1]*x_cc[0] + b[2]*x_cc[1]);
```

```
  y_nn=y_nn-(c[1]*y_cc[0] + c[2]*y_cc[1]);
  y_nn=y_nn/c[0];
  // Mise à jour de la sortie
  y_cc[1]=y_cc[0];
  y_cc[0]=y_nn;

  // Mise à jour de la sortie
  x_cc[1]=x_cc[0];
  x_cc[0]=x_nn;
  // Renvoie du résultat
  return y_nn;
}
```

Accueil Asservissement avec Arduino