Sommaire

Objectifs

- Correction d'un système du 2nd ordre en BF
- Implémentation du correcteur Avance de Phase (AP):
- 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
- Analyse du correcteur Avance de Phase
- Précision/ Stabilité/ Rapidité du correcteur AP
- La réponse à un échelon d'un système en 2nd ordre

- Etc.

La fonction CorrPD_AP()

```
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;
}
```

Le programme Arduino complet

```
/*
 * 1. Correction d'un système du 2nd ordre en BF
 * 2. Implémentation du correcteur Avance de Phase (AP):
 *    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
 * 3. Analyse du correcteur Avance de Phase
 * 4. Précision/ Stabilité/ Rapidité du correcteur AP
 * 5. La réponse à un échelon d'un système en 2nd ordre
 * 6. Etc.
 *

                 -------------    -------------
 x(n) --[-]------     C(p)    ----     SYS2    ---------- y(n): Sortie Corrigée
         -       -------------    -------------   -
         -                                        -
         ---------------------<--------------------
         Correcteur PI: C(p)= k*(1+aTp)/(1+Tp)
                         -------------
 x(n) --[-]---------------    SYS2     ---------- y(n): Non Corrigée
         -                -------------           -
         -                                        -
         ---------------------<--------------------

*/


#define   Fn      10.00
#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 PI: C(p)= k(1+aTp)/(1+Tp)
  double k0=5.0;
  double phi_m=34.0*PI/180; // 34°
  double a=(1.0+sin(phi_m))/(1.0-sin(phi_m)); // a=(1-sin(phi))/(1+sin(phi))
  double t0=1.0/(sqrt(a)*(3.0*Wn));
  y_corr[1]= CorrPD_AP(eps_n[1], x_c, y_c,  a,  t0,  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;
}
```

Références des cours

- Correction des systèmes linéaires continus asservis
- Correction des systèmes Asservis
- Cours d’Automatique : les asservissements continus
- Méthodes de réglages des paramètres du régulateur PID

- Automatique Linéaire 1
- ~~AUTOMATIQUE 4 Correction des systèmes asservis~~
- Notion de correction des systèmes asservis
- Echantillonnage et quantification
- ~~Automatique régulation~~
- ~~AUTOMATIQUE Support de cours~~
- AUTOMATIQUE SYSTEMES ASSERVIS LINEAIRES CONTINUS
- AUTOMATIQUE SYSTEMES ASSERVIS

LINEAIRES ECHANTILLONNES

- Automatique: Commande des Systèmes Linéaires
- ~~Automatique Systèmes linéaires et asservissement~~

Accueil Asservissement avec Arduino