REGULATION ET ASSERVISSEMENT

COURS

TSSI cours asservissement.pdf

MODELES MATLAB A SAUVEGARDER

regulationPID_2022a.slx

filtreNumerique_2022a.slx

TP – Régulation de vitesse d'un moteur à courant continu avec PID

Objectifs du TP

- Mesurer la vitesse d'un moteur CC avec un codeur incrémental
- Comprendre la différence entre boucle ouverte et boucle fermée
- Mettre en œuvre progressivement un correcteur P, puis PI, puis PID
- Observer l'erreur statique, le dépassement et la sensibilité au bruit

---

Travail demandé

- Décrire la réponse du moteur en boucle ouverte
- Tracer ou décrire la courbe vitesse / consigne pour P
- Expliquer pourquoi un écart statique persiste
- Montrer comment I supprime cet écart
- Comparer les dépassements pour P, PI et PID
- Conclure sur l'intérêt des trois termes du PID

---

1) Commande du moteur en boucle ouverte

Manipulations :

- Choisir le mode boucle ouverte
- Choisir une consigne en tr/min
- Observer la vitesse indiquée par le programme
- Bloquer légèrement l'axe du moteur avec le doigt

Observations attendues :

- La vitesse chute immédiatement lorsque l'axe est freiné
- Le moteur ne corrige pas cette chute : c'est normal en boucle ouverte
- La vitesse dépend de la charge, des frottements et de la tension

Conclusion : La boucle ouverte ne permet pas de maintenir une vitesse constante

---

2) Mise en place d'un correcteur P (Proportionnel)

Manipulations :

- Activer le correcteur proportionnel : $u = K_p \cdot e$
- Fixer une consigne (ex : 1500 tr/min)
- Le programme convertit automatiquement en ticks/s
- Freiner légèrement le moteur avec le doigt
- Augmenter progressivement K_p : 0.2 → 0.5 → 1.0 → 2.0

Observations attendues :

- Le moteur augmente la PWM pour compenser la perturbation
- La vitesse remonte partiellement
- Il reste un écart statique : $\text{vitesse réelle} < \text{consigne}$
- Si K_p devient trop grand : oscillations, vibrations, instabilité

Conclusion : Le correcteur P réduit l'erreur, mais ne la supprime pas

---

3) Mise en place du correcteur I (Intégral)

Manipulations :

- Ajouter le terme intégral : $u = K_p e + K_I \int e(t)\,dt$
- Débuter avec K_I = 0.05, puis 0.1 max
- Freiner l'axe puis relâcher

Observations attendues :

- L'erreur statique disparaît
- La vitesse atteint précisément la consigne
- Si K_I trop fort : dépassement, oscillations lentes, instabilité

Conclusion : Le correcteur I supprime l'erreur statique, mais ne doit jamais être trop fort

---

4) Mise en place du correcteur D (Dérivé)

Manipulations :

- Ajouter le terme dérivé : $u = K_p e + K_I \int e\,dt + K_D \frac{de}{dt}$
- Tester avec K_D = 0.01, puis 0.05
- Freiner l'axe pour observer la réaction

Observations attendues :

- Le système est mieux amorti
- Le dépassement diminue
- La stabilité augmente

Attention : Si K_D trop élevé → bruit, vibrations, instabilité

Conclusion : Le terme D stabilise le système, mais n'améliore pas la précision

---

Synthèse des rôles P / I / D

Correcteur	Rôle principal	Risques si trop fort
P	réduit l'erreur	oscillations
I	supprime l'erreur statique	dépassement, instabilité
D	amortit, stabilise	amplification du bruit

Code Arduino du TP (consigne en tr/min, PID en ticks/s)

```
/*
   ======================================
      TP REGULATION : MODE BO / MODE PID
   ======================================

   MODE BO :
     - PWM = conversion(consigne_tr/min)
     - codeur mesure mais NE corrige PAS
     - aucune régulation

   MODE PID :
     - consigne tr/min convertie en ticks/s
     - codeur mesure vitesse
     - PID corrige PWM
*/

const int TICKS_PAR_TOUR = 20;  // à ajuster selon le codeur

// --- Pont en H ---
```

```
const int M_AV = 3;   // PWM forward
const int M_AR = 6;   // PWM reverse

// --- Codeur ---
const int canalA = 2;
const int canalB = 11;

volatile long ticks = 0;

// --- PID ---
float kp = 0.8;
float ki = 0.1;
float kd = 0.05;

float erreur, erreurPrec = 0;
float integral = 0;

float consigne_rpm = 0;      // consigne entrée par Serial (tr/min)
float consigne_ticks_s = 0;  // consigne convertie pour le PID

// --- Modes ---
enum Mode { BO, PID_MODE };
Mode mode = BO;

// --- Mesure ---
unsigned long lastMeasure = 0;
const unsigned long period = 100; // 100 ms


// ============ INTERRUPTIONS CODEUR ============
void ISR_codeur() {
  if (digitalRead(canalB))
    ticks++;
  else
    ticks--;
}


// ============ CONVERSION tr/min -> PWM (BO) ============
int rpmToPWM(float rpm) {
  // Ajuste selon ton moteur
  // Ex : 0–3000 tr/min -> 0–255 PWM
  if (rpm < 0) rpm = 0;
  if (rpm > 3000) rpm = 3000;

  return map(rpm, 0, 3000, 0, 255);
}
```

```
// ============ CONVERSION tr/min -> ticks/s (PID) ============
float rpmToTicksSec(float rpm) {
  return (rpm * TICKS_PAR_TOUR) / 60.0;
}


// ============ COMMANDE MOTEUR ============
void setPWM(int pwm) {
  if (pwm >= 0) {
    digitalWrite(M_AR, LOW);
    analogWrite(M_AV, pwm);
  } else {
    digitalWrite(M_AV, LOW);
    analogWrite(M_AR, -pwm);
  }
}


// ================================================
//                     SETUP
// ================================================
void setup() {
  Serial.begin(9600);

  pinMode(M_AV, OUTPUT);
  pinMode(M_AR, OUTPUT);
  pinMode(canalB, INPUT);

  attachInterrupt(digitalPinToInterrupt(canalA), ISR_codeur, RISING);

  Serial.println("=== TP REGULATION : BO / PID ===");
  Serial.println("Tapez BO ou PID pour changer de mode.");
  Serial.println("Tapez une consigne en tr/min (ex : 1500)");
}


// ================================================
//                     LOOP
// ================================================
void loop() {

  // ----------- Lecture mode / consigne ------------
  if (Serial.available() > 0) {
    String txt = Serial.readStringUntil('\n');
    txt.trim();

    if (txt.equalsIgnoreCase("BO")) {
      mode = BO;
      Serial.println("Mode = BOUCLE OUVERTE");
      return;
    }
```

```
  if (txt.equalsIgnoreCase("PID")) {
    mode = PID_MODE;
    Serial.println("Mode = PID");
    return;
  }

  // Sinon c'est une consigne tr/min
  consigne_rpm = txt.toFloat();
  consigne_ticks_s = rpmToTicksSec(consigne_rpm);

  Serial.print("Consigne = ");
  Serial.print(consigne_rpm);
  Serial.print(" tr/min  -> ");
  Serial.print(consigne_ticks_s);
  Serial.println(" ticks/s");
}


// ----------- Mesure toutes les 100 ms -----------
unsigned long now = millis();
if (now - lastMeasure < period) return;
lastMeasure = now;

long ticks_mes = ticks;
ticks = 0;

float vitesse_ticks_s = ticks_mes * (1000.0 / period);
float vitesse_rpm = (vitesse_ticks_s * 60.0) / TICKS_PAR_TOUR;


// ================================================
//                  MODE BO
// ================================================
if (mode == BO) {

  int pwm = rpmToPWM(consigne_rpm);  // conversion directe consigne → PWM
  setPWM(pwm);

  Serial.print("[BO] consigne = ");
  Serial.print(consigne_rpm);
  Serial.print(" tr/min | vitesse = ");
  Serial.print(vitesse_ticks_s);
  Serial.print(" ticks/s | ");
  Serial.print(vitesse_rpm);
  Serial.print(" tr/min | PWM = ");
  Serial.println(pwm);

  return;
```

```
  }


  // ================================================
  //                  MODE PID
  // ================================================
  if (mode == PID_MODE) {

    erreur = consigne_ticks_s - vitesse_ticks_s;
    integral += erreur * (period / 1000.0);
    float deriv = (erreur - erreurPrec) / (period / 1000.0);
    erreurPrec = erreur;

    float commande = kp * erreur + ki * integral + kd * deriv;

    // saturation
    if (commande > 255) commande = 255;
    if (commande < -255) commande = -255;

    setPWM(commande);

    Serial.print("[PID] consigne = ");
    Serial.print(consigne_rpm);
    Serial.print(" tr/min | vitesse = ");
    Serial.print(vitesse_ticks_s);
    Serial.print(" ticks/s | ");
    Serial.print(vitesse_rpm);
    Serial.print(" tr/min | PWM = ");
    Serial.println(commande);

    return;
  }
}
```