TP – Asservissement en distance du mBot : Correcteur PID

Objectifs

- Comprendre la logique d'un asservissement en boucle fermée
- Utiliser le capteur ultrason pour mesurer une distance réelle
- Étudier séparément les rôles des termes P, I et D
- Analyser la stabilité, le dépassement, le bruit et l'erreur statique
- Interpréter les données fournies dans le moniteur série

Prérequis

Lirairie mBot: bibliotheque_arduino_mbot

Activité

Forcer le mBot) suivre une cible) à 20 cm de distance.

1) Principe général de l'asservissement

Le robot doit se stabiliser à **20 cm** de l'obstacle.

L'erreur est définie par : $e = D_{mes} - D_{cons}$

- si e > 0 → le robot est trop loin → avancer
- si e < 0 → le robot est trop près → reculer

La commande est donnée par un correcteur PID : $v = K_p e + K_i \int e\,dt + K_d \frac{de}{dt}$

Rôle des trois termes

- **P** : corrige proportionnellement à l'erreur
- **I** : supprime l'erreur statique mais peut provoquer des oscillations
- **D** : réduit le dépassement, amortit le mouvement

Filtrage du dérivé

Le capteur ultrason étant bruyant, on utilise un dérivé filtré :

$$d_{filt} = \alpha\, d_{filt} + (1-\alpha)\,\frac{e - e_{prev}}{dt}$$

Anti-windup (limitation de l'intégrale)

$$I = \text{clip}(I + e\,dt,\ -I_{max},\ I_{max})$$

Reset si inversion du signe de l'erreur :

$$e \cdot e_{prev} < 0 \Rightarrow I = 0$$

Saturation moteurs

$$v = \begin{cases} V_{max} & \text{si } v > V_{max} \\ -V_{max} & \text{si } v < -V_{max} \\ v & \text{sinon} \end{cases}$$

Zone de vitesse minimale

$$|v| < V_{min} \Rightarrow v = \begin{cases} V_{min} & \text{si } v > 0 \\ -V_{min} & \text{si } v < 0 \end{cases}$$

---

2) Travail demandé

Partie A — Analyse conceptuelle

- Expliquer la signification physique de l'erreur e.
- Pourquoi un système en boucle fermée est-il nécessaire ?
- Donner le rôle séparé des termes P, I, D.
- Pourquoi la dérivée doit-elle être filtrée ?
- Quel est l'intérêt du mécanisme anti-windup ?

Partie B — Expérimentations

Étape 1 : Action P seule

- Mettre Ki = 0 et Kd = 0
- Faire varier Kp (2, 4, 6, 8…)

Questions :

- Une erreur statique persiste-t-elle ?
- Y a-t-il un dépassement ?
- Le robot oscille-t-il si Kp est trop grand ?

Étape 2 : Action PI

* Garder un Kp raisonnable (ex : 4 ou 6) * Ajouter Ki (0.1 → 0.2)

Questions : * L'erreur statique disparaît-elle ? * Le robot devient-il plus oscillant ? * Que provoque un Ki trop élevé ?

Étape 3 : Action PID

- Ajouter un terme dérivé Kd (0.5 ou 1.0)

Questions :

- Le dépassement diminue-t-il ?
- La stabilité augmente-t-elle ?
- Le dérivé amplifie-t-il le bruit ?
- Le filtre (α) améliore-t-il les tremblements du robot ?

Partie C — Analyse des données série

Les données typiques affichées :

```
D = distance mesurée
e = erreur
v = commande moteur
I = intégrale
d = dérivée filtrée
```

Questions :

- La commande v se rapproche-t-elle de 0 une fois stabilisé ?
- L'intégrale atteint-elle les limites Imax ?
- Le dérivé filtré est-il stable ou bruité ?
- Observe-t-on un dépassement (overshoot) dans la distance D ?

---

3) Synthèse finale

Rédiger un paragraphe répondant aux questions suivantes :

- Pourquoi P seul ne suffit-il pas ?
- En quoi I améliore la précision mais peut dégrader la stabilité ?
- Quel est le rôle du dérivé dans l'amortissement ?
- Quel triplet (Kp, Ki, Kd) est optimal pour VOTRE robot ? (justifier)
- Pourquoi d'un robot à l'autre les réglages diffèrent-ils ?

---

4) Bonus facultatif

- Ajouter une consigne variable via le port série
- Tracer les courbes D, e, v, I dans un tableur
- Tester 10 cm, 20 cm et 30 cm pour analyser la robustesse

```
#include "mCoreLite.h"

mCoreLite bot;

```

```
// ----- CONSIGNE -----
float Dcons = 20.0;      // distance cible (cm)

// ----- PARAMÈTRES PID -----
float Kp = 6.0;          // proportionnel
float Ki = 0.15;         // intégral
float Kd = 0;          // dérivé (amortissement)

// ----- MEMOIRES PID -----
float integral = 0.0;
float e_prev = 0.0;
unsigned long lastTime = 0;

// ----- SECURITE INTEGRALE (ANTI-WINDUP) -----
float Imax = 50.0;       // limite de l'intégrale

// ----- VITESSE -----
int Vmin = 60;           // évite les sifflements
int Vmax = 200;          // saturation

// ----- FILTRE DERIVÉ -----
// (évite les tremblements dus au bruit US)
float alpha = 0.7;       // 0 = pas de filtre, 1 = filtre fort
float d_filtered = 0.0;

void setup() {
  Serial.begin(115200);
  bot.botSetup();
  lastTime = millis();
}

void loop() {

  // ----- 1) MESURE -----
  float Dmes = bot.distanceCM();

  // ----- 2) ERREUR -----
  float e = Dmes - Dcons;

  // ----- 3) TEMPS -----
  unsigned long now = millis();
  float dt = (now - lastTime) / 1000.0;
  if (dt <= 0) dt = 0.001;   // sécurité
  lastTime = now;

  // ----- 4) ZONE MORTE -----
  if (fabs(e) < 1.0) {
    integral = 0;           // détendre l'intégrale
    bot.motors(0,0);
    debug(Dmes, e, 0, integral, 0);
    return;
```

```
  }

  // ----- 5) INTÉGRALE + ANTI-WINDUP -----
  integral += e * dt;
  if (integral >  Imax) integral =  Imax;
  if (integral < -Imax) integral = -Imax;

  // Réinitialisation si inversion de signe de l'erreur
  if (e * e_prev < 0) integral = 0;

  // ----- 6) DÉRIVÉE (!\ capteur bruyant) -----
  float derivative = (e - e_prev) / dt;
  e_prev = e;

  // Filtre passe-bas sur le dérivé
  d_filtered = alpha * d_filtered + (1 - alpha) * derivative;

  // ----- 7) SORTIE PID -----
  float v = Kp * e + Ki * integral + Kd * d_filtered;

  // ----- 8) SATURATION -----
  if (v >  Vmax) v =  Vmax;
  if (v < -Vmax) v = -Vmax;

  // ----- 9) VITESSE MINIMALE -----
  if (fabs(v) > 0 && fabs(v) < Vmin) {
    v = (v > 0) ? Vmin : -Vmin;
  }

  // ----- 10) ACTION -----
  // v > 0 → avancer (cible trop loin)
  // v < 0 → reculer (cible trop proche)
  bot.motors(-v, v);

  // ----- 11) DEBUG -----
  debug(Dmes, e, v, integral, d_filtered);
}

// ================== DEBUG ======================
void debug(float D, float e, float v, float I, float d) {
  Serial.print("D=");
  Serial.print(D);
  Serial.print("  e=");
  Serial.print(e);
  Serial.print("  v=");
  Serial.print(v);
  Serial.print("  I=");
  Serial.print(I);
  Serial.print("  d=");
```

```
  Serial.println(d);
}
```