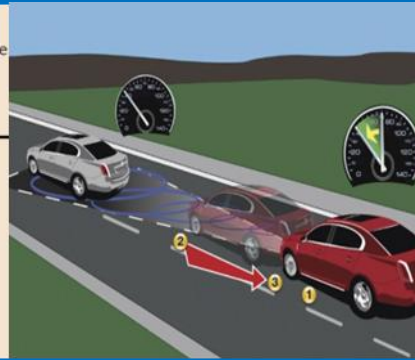
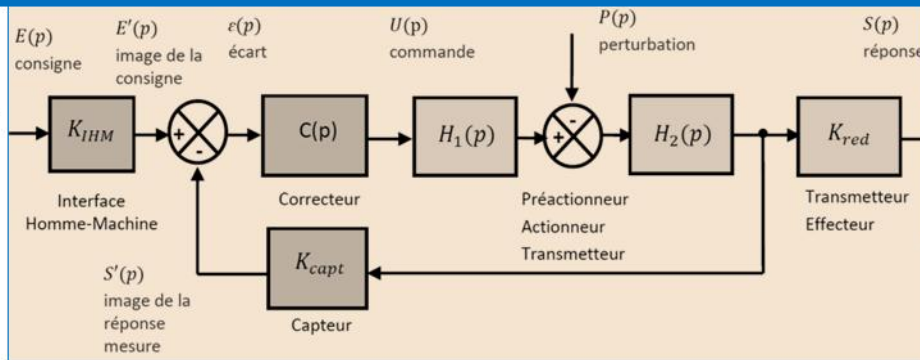


Activité pratique

Mise en évidence du PID et de son influence



4 h 00



1. INTRODUCTION

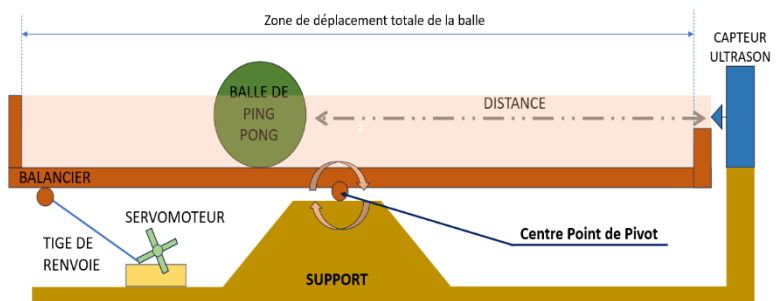
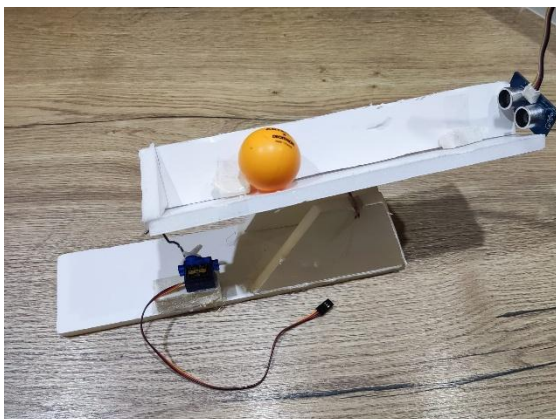
A travers cette activité expérimentale, nous allons mettre en avant comment élaborer un système qui se corrige automatiquement pour répondre à une consigne.

Pour illustrer ses propos, prenons l'exemple d'un **drone**

- **But :** Ce système doit être le plus stable possible lors d'un vol, notamment en phase stationnaire.
 - Si le drone bouge, la distance varie et utilise donc ces hélices pour bouger légèrement dans le sens opposé : cela compense le mouvement et garde le drone stable.
 - C'est comme si l'information issue du capteur ultrason permettait de "corriger" tout le temps sa position pour que le drone reste stable.
- **Comment :**
 - Le **capteur à ultrason** mesure en temps réel la distance entre le drone et le sol.
 - Si le drone descend trop (distance < à la consigne), le système accélère les moteurs pour remonter.
 - Si le drone monte trop (distance > à la consigne), il ralentit les moteurs pour redescendre.

→ On corrige en permanence pour rester à la bonne hauteur !

Dans notre cas, nous allons commander la position d'une balle de ping-pong, mesurée par un capteur ultrason, Cette balle est placée sur un **balancier** commandé par un servomoteur.



- **But :** Nous voulons que la balle reste toujours à une certaine distance : par exemple, à 10 cm du centre du balancier.
- **Comment :** Si la balle s'éloigne trop, le balancier bouge pour la ramener à la bonne distance. Si elle se rapproche trop, le balancier bouge dans l'autre sens pour la repousser. C'est un peu comme si le balancier "corrigeait" tout le temps la position de la balle pour qu'elle reste là où nous voulons.



👉 Dans les deux cas, c'est un système automatique qui compare la mesure (capteur) à l'objectif, puis agit (hélice/balancier) pour corriger l'erreur.

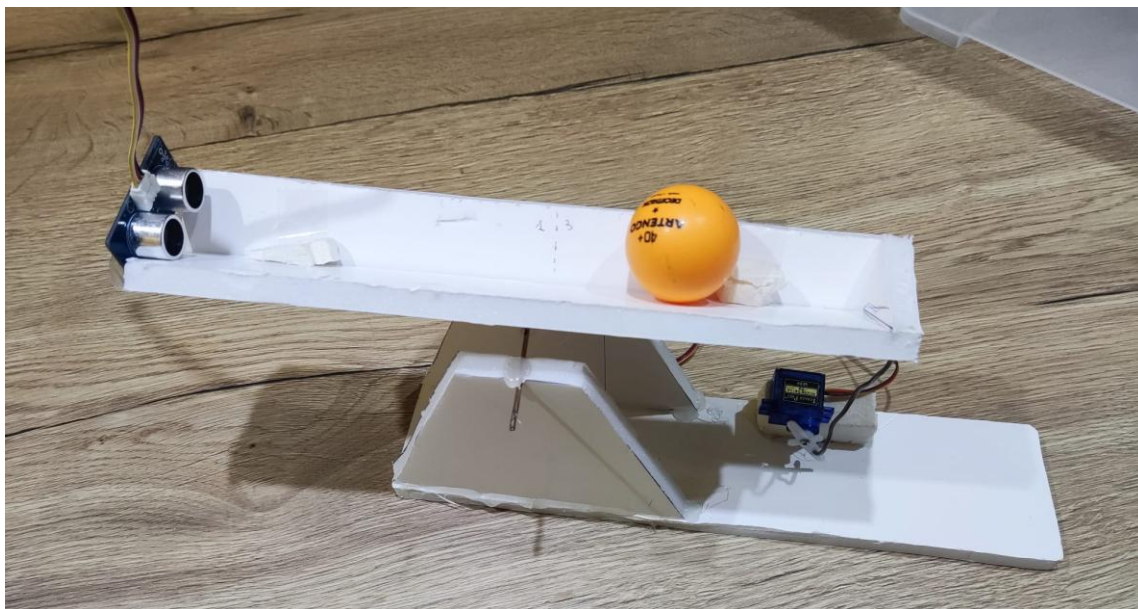
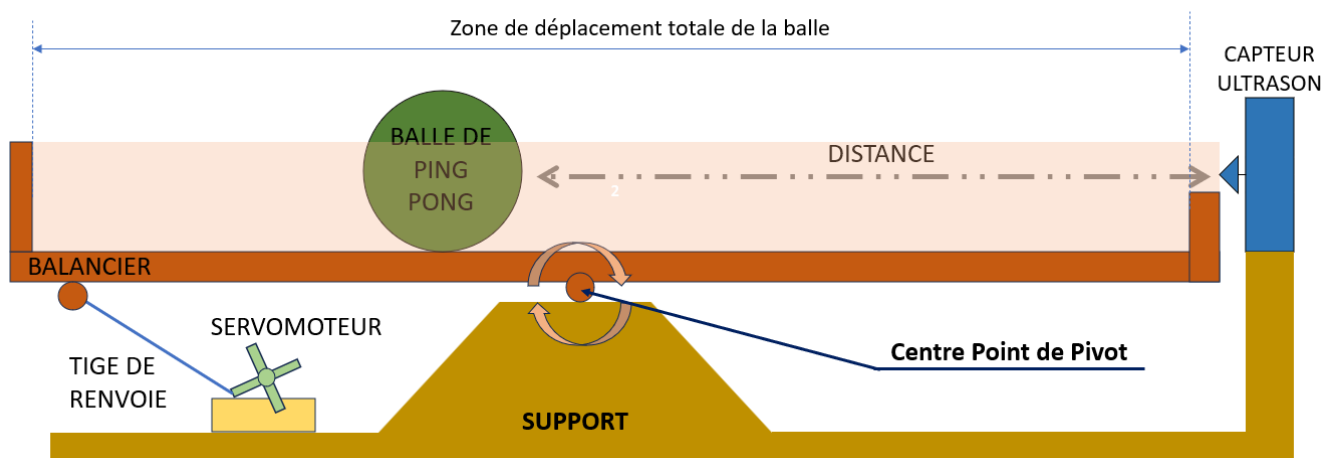
En résumé

- Le balancier et le drone font la même chose : ils **se corrigent en permanence** pour atteindre un objectif (distance ou stabilité). 🚁 ⚡

2. Présentation

- Présentation du montage*

Voici le montage qui servira d'exemple



Ici, nous déposons une balle de ping-pong à une distance de 13 cm du capteur ultrason. Le microcontrôleur pilote un servomoteur. Si on vient perturber la position de la balle, ce servomoteur permet de faire monter ou descendre le balancier et ainsi rapprocher ou éloigner la balle pour la repositionner à 13 cm.



2.1. Programmation du microcontrôleur et script Python

Voici le code pour la partie Arduino – qui pilote le balancier

```
#include <Servo.h>
#include "Ultrasonic.h"
Ultrasonic ultrasonic(7);
Servo servo;

#define kp 10
#define ki 0      On agira sur ces 3 paramètres en leur donnant différentes valeurs
#define kd 0

double priError = 0;
double toError = 0;
void setup() {
    servo.attach(5);
    Serial.begin(9600);
    servo.write(65);
}
void loop()
{
    PID();
}
void PID()
{
    long RangeInCentimeters;
    RangeInCentimeters = ultrasonic.MeasureInCentimeters();
    Serial.println(RangeInCentimeters);
    int setP = 13;
    double error = setP - RangeInCentimeters;
    double Pvalue = error * kp;
    double Ivalue = toError * ki;
    double Dvalue = (error - priError) * kd;
    double PIDvalue = Pvalue + Ivalue + Dvalue;
    priError = error;
    toError += error;
    int Fvalue = (int)PIDvalue;
    Fvalue = map(Fvalue, -135, 135, 135, 0);
    if (Fvalue < 0)
    {
        Fvalue = 0;
    }
    if (Fvalue > 135)
    {
        Fvalue = 135;
    }
    servo.write(Fvalue);
    delay(300);
}
```



Voici le code pour la partie Pýthon – qui nous permettra d'obtenir des courbes en fonction du temps

Vous pouvez personnaliser la représentation graphique ainsi que les paramètres de la communication série en modifiant les variables en début de script :

```
import serial
import matplotlib.pyplot as plt
import time
port_com = 'COM7' # indique le port COM7 comme port série de lecture
valeurs=[]
temps=[]
serie=serial.Serial(port_com ,9600) # ouvre une liaison série en 9600bps
plt.style.use('bmh')
plt.ylabel("valeur")
plt.xlabel("temps en s")
plt.ion()          # on entre en mode interactif
start=time.time() # mesure de l'instant initial
i=0
while (i<60): # on définit le nombre d'itération pour tracer le graphique, par
exemple 60 points. Ce nombre d'itération définit la finesse du tracé
    mesure= int (serie.readline()) # lit la donnée sur la liaison série
    valeurs.append(mesure)        # ajout de mesure à la liste des valeurs
    instant=time.time()-start# calcul du temps écoulé depuis l'instant initial
    temps.append(instant)         # ajout de instant à la liste des temps
    print (mesure,instant)        # affiche dans la console les coordonnées du point
    plt.plot(temps,valeurs,marker='o') # trace la courbe
    plt.draw()                    # affiche la courbe en mode interactif
    i=i+1
plt.ioff()    # on quitte le mode interactif pour rendre la main à l'utilisateur
sur la courbe
plt.show()    # afficher la courbe
```

3. Analyse du modèle fourni et phase de test

Rédigez un compte rendu numérique au fur et à mesure : l'ensemble des courbes obtenues sont à analyser en mettant en perspective les données d'entrées et les résultats obtenus.

Question 1 - Cherchez la définition d'asservissement pour essayer de comprendre ce que nous allons développer dans la suite de l'activité pratique.

L'**asservissement** (ou **régulation automatique**) est un système de contrôle qui permet à un dispositif physique (mécanique, électronique, thermique, etc.) de maintenir une grandeur de sortie (position, vitesse, température, etc.) aussi proche que possible d'une **consigne** (valeur désirée), malgré les perturbations extérieures.

Question 2 - Commentez chaque ligne de code avec un commentaire pour expliciter la fonctionnalité, en observant le code Arduino que vous avez à disposition,



```
#include <Servo.h> // Bibliothèque Servo : Permet de contrôler un servo-moteur avec Arduino.
#include "Ultrasonic.h" // Bibliothèque Ultrasonic : Utilisée pour interfacer le capteur à ultrasons (HC-SR04 ou similaire).
Ultrasonic ultrasonic(7); // Initialisation du capteur ultrason : Crée un objet ultrasonic et connecte le capteur à la broche 7 d'Arduino
Servo servo;
```

- **Initialisation du servo** : Crée un objet servo pour contrôler le moteur.

```
#define kp 10
#define ki 0
#define kd 0
```

- **Constantes PID** :
 - kp (gain proportionnelle) = 10.
 - ki (gain intégrale) = 0 (désactivée).
 - kd (gain dérivée) = 0 (désactivée).
 - Ici, seul le contrôle proportionnel (P) est actif.
-

```
double priError = 0;
```

- **Erreur précédente** : Stocke l'erreur du cycle précédent pour le calcul dérivé (même si kd = 0).
-

```
double toError = 0;
```

- **Erreur totale** : Accumule les erreurs pour le calcul intégral (même si ki = 0).
-

```
void setup() {
  servo.attach(5);
  Serial.begin(9600);
  servo.write(65);
}
```

- **Configuration initiale** :
 - servo.attach(5) : Attache le servo à la broche **5**.
 - Serial.begin(9600) : Initialise la communication série pour le débogage.
 - servo.write(65) : Positionne le servo à **65°** au démarrage.
-

```
void loop() {
  PID();
}
```

- **Boucle principale** : Appelle continuellement la fonction PID().
-

```
long RangeInCentimeters;
```

- **Variable de distance** : Stocke la distance mesurée par le capteur en cm.

```
RangeInCentimeters = ultrasonic.MeasureInCentimeters();
```

- **Mesure de distance** : Récupère la distance de l'obstacle en cm.

```
Serial.println(RangeInCentimeters);
```

- **Affichage de la distance** : Envoie la distance au moniteur série pour débogage.

```
int setP = 13;
```

- **Consigne (Setpoint)** : La distance cible à maintenir (13 cm).

```
double error = setP - RangeInCentimeters;
```

- **Calcul de l'erreur** : Différence entre la consigne et la distance mesurée.

```
double Pvalue = error * kp;
```

- **Terme Proportionnel** : Ajustement basé sur l'erreur actuelle.

```
double Ivalue = toError * ki;
```

- **Terme Intégral** : Ajustement basé sur l'accumulation des erreurs (ici désactivé car ki = 0).

```
double Dvalue = (error - priError) * kd;
```

- **Terme Dérivé** : Ajustement basé sur le taux de changement de l'erreur (ici désactivé car kd = 0).

```
double PIDvalue = Pvalue + Ivalue + Dvalue;
```

- **Sortie PID** : Somme des trois termes (ici réduite à Pvalue).

```
priError = error;
```

- **Mise à jour de l'erreur précédente** : Pour le prochain calcul dérivé.
-




```
toError += error;
```

- **Mise à jour de l'erreur totale** : Pour le prochain calcul intégral.

```
int Fvalue = (int)PIDvalue;
```

- **Conversion en entier** : Convertit la sortie PID en entier pour le servo.

```
Fvalue = map(Fvalue, -135, 135, 135, 0);
```

- **Mapping des valeurs** :

- Convertit la plage $[-135, 135]$ (typique d'un PID) en $[135, 0]$ pour le servo.
- Inversion possible pour corriger le sens de rotation.

```
if (Fvalue < 0) { Fvalue = 0; }
```

- **Limite inférieure** : Empêche les valeurs négatives (0° pour le servo).

```
if (Fvalue > 135) { Fvalue = 135; }
```

- **Limite supérieure** : Empêche les valeurs $> 135^\circ$ (limite physique du servo).

```
servo.write(Fvalue);
```

- **Commande du servo** : Positionne le servo à l'angle calculé.

```
delay(300); // Pause : Attend 300 ms entre chaque mesure pour stabiliser le système.
```

Appeler le professeur pour validation

Expérimentation 1 - Téléversez votre programme et **analysez** ce qui vous obtenez dans le moniteur série d'Arduino.

Expérimentation 2 - Déplacez légèrement la balle de ping-pong et **observez** ce qui se passe.

Expérimentation 3 - Fermez le moniteur série et **lancez** EduPython. **Lancez** le script python mis à votre disposition.

Appeler le professeur pour validation

4. Réglage du gain du correcteur proportionnel - P

Nous allons observer dans cette partie uniquement l'influence du **correcteur proportionnel**. On parlera de **gain**.

Expérimentation 4 - Réalisez 4 tests différents pour les valeurs suivantes : $k_p = 3$; $k_p = 10$; $k_p = 30$; $k_p = 100$.

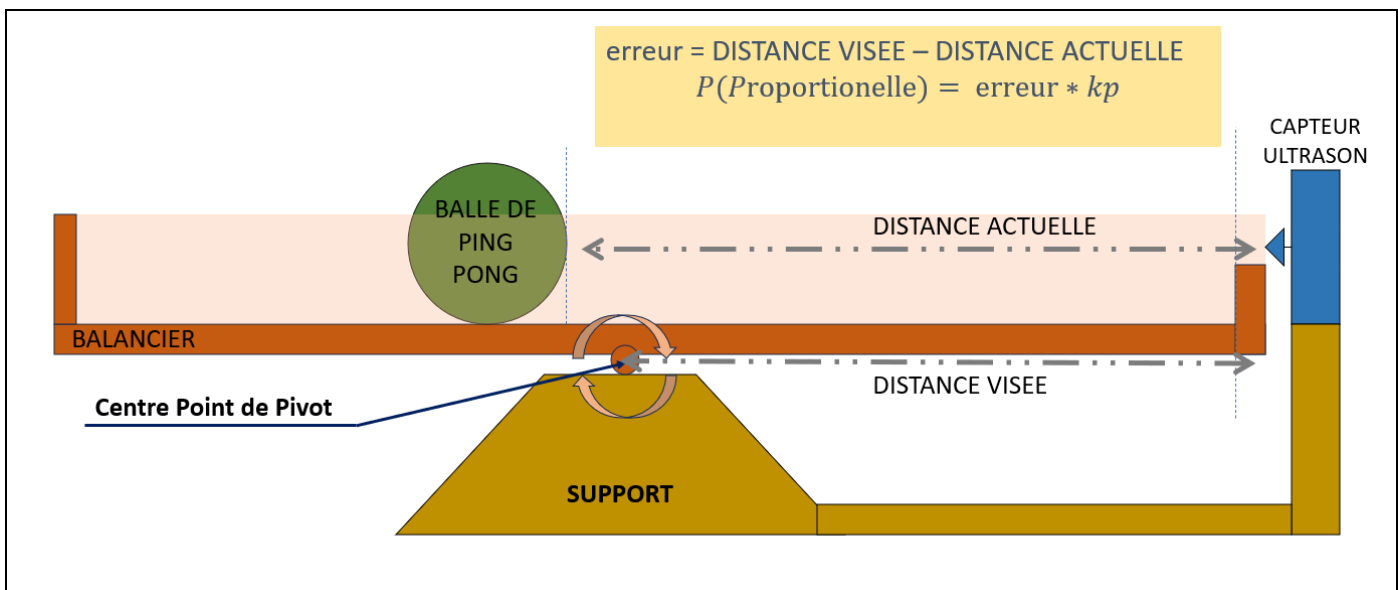
- ***NOTA** : Pour modifier chaque paramètre, il est nécessaire de fermer le script et de le relancer une fois la modification du programme Arduino téléversée.*

Question 3 - A chaque expérimentation, **observez** la réponse du système: **analyser** et **conclure** quant à l'influence de ce paramètre en vous appuyant sur les courbes obtenues grâce au script Python.



Synthèse prof

- **P (Proportionnel)** — Le contrôleur proportionnel (P) ajuste la commande du système en fonction de l'erreur (écart entre la consigne et la mesure). Plus l'erreur est grande, plus la correction appliquée est importante.
- **Principe :**
 - Erreur = Consigne – Mesure réelle
 - Sortie du contrôleur = $K_p \times \text{Erreur}$
 - K_p (gain proportionnel) : Détermine l'intensité de la réaction du système.
- **Réglage de K_p :**
 - Si K_p est trop faible → Réaction lente, système peu précis.
 - Si K_p est trop élevé → Risque d'oscillations ou d'instabilité.
 - Méthode : On teste différentes valeurs de K_p pour trouver le meilleur compromis.



- *Remarque*

Vous avez dû constater que l'erreur entre la distance voulue et la distance obtenue diminue à mesure que le gain P augmente mais génère plus d'instabilité et ne l'annule pas complètement (sauf dans certains cas). **Nous allons donc chercher à réduire cette différence qu'on appelle l'écart statique en jouant sur un effet stabilisateur.**

4.1. Limite de stabilité

Expérimentation 5 - Augmentez le gain jusqu'à obtenir un comportement à la limite de l'instabilité (mais stable quand même !)

Question 4 - Relevez alors l'erreur statique avec ce nouveau gain.

- *NOTA : Il est nécessaire de fermer le script et de le relancer une fois la modification du programme Arduino téléversée.*

4.2. Régime amorti

Dans le cadre d'un système asservi, il est préférable qu'il n'y ait pas de dépassement (et encore moins d'oscillations).

Expérimentation 6 - Modifiez le gain jusqu'à obtenir un comportement sans oscillation ni dépassement, avec le gain le plus grand possible.

Question 5 - Relevez alors l'erreur statique avec ce nouveau gain.



Question 6 - Conclure quant à l'effet du correcteur proportionnel sur le comportement du système.

Appeler le professeur pour validation

5. Utilisation d'un correcteur à action intégrale - I

Afin d'obtenir une meilleure précision, on se propose d'utiliser un correcteur plus performant qu'un simple gain, capable d'appliquer une correction avec une **action intégrale** : ainsi la commande du moteur sera compensée si l'écart persiste dans la durée.

Expérimentation 7 - Configurez l'action **P** (proportionnel) avec la valeur de gain déterminée précédemment.
Mettez la valeur de l'action **I** (intégrale) à 1 (ki dans le programme Arduino).

Question 7 - Lancez l'exécution et **décrivez** le comportement.

Expérimentation 8 - Réalisez plusieurs essais jusqu'à obtenir un comportement satisfaisant. (Il faut agir sur les deux paramètres P et I)

Question 8 - Conclure quant à l'effet du correcteur proportionnel intégral sur le comportement du système.

Synthèse prof

I (intégral) — Le contrôleur intégral (I) permet d'éliminer les erreurs résiduelles en régime permanent (quand le système se stabilise, mais avec un petit écart par rapport à la consigne).

- **Fonctionnement :**

1. **Intégration de l'erreur :**

- On accumule (somme) les erreurs au fil du temps.
- Cela permet de détecter une erreur persistante que le contrôleur proportionnel (P) seul ne corrige pas.

2. **Correction via Ki :**

- La somme des erreurs est multipliée par Ki (gain intégral).
- Plus Ki est élevé, plus le système réagit fortement aux erreurs accumulées.

3. **Équation :**

$\text{Sortie}_I = K_i \times \int (\text{Erreur}) dt$ où \int représente l'intégrale, c'est-à-dire la somme continue des erreurs passées.

- **Réglage de Ki :**

- Ki trop faible → La correction est lente, l'erreur résiduelle persiste.
- Ki trop élevé → Le système devient instable (oscillations ou dépassements).
- Méthode : On teste différentes valeurs pour trouver un équilibre.

- **Exemple dans votre code Arduino :**

```
toError += error; // Accumulation de l'erreur (intégrale)
```

```
double Ivalue = toError * ki; // Correction intégrale et progressive des petits écarts persistants.
```

- **Pourquoi ?**

- Élimine l'erreur statique (ex. : un servo qui reste à 12 cm au lieu de 13 cm malgré le contrôle P).



- Complète l'action du contrôleur P pour plus de précision.

- **Inconvénient :**

Un Ki trop fort peut rendre le système lent à réagir ou provoquer des oscillations.

Comparaison P vs I :

Proportionnel (P)

Réagit à l'erreur instantanée.

Peut laisser une erreur résiduelle.

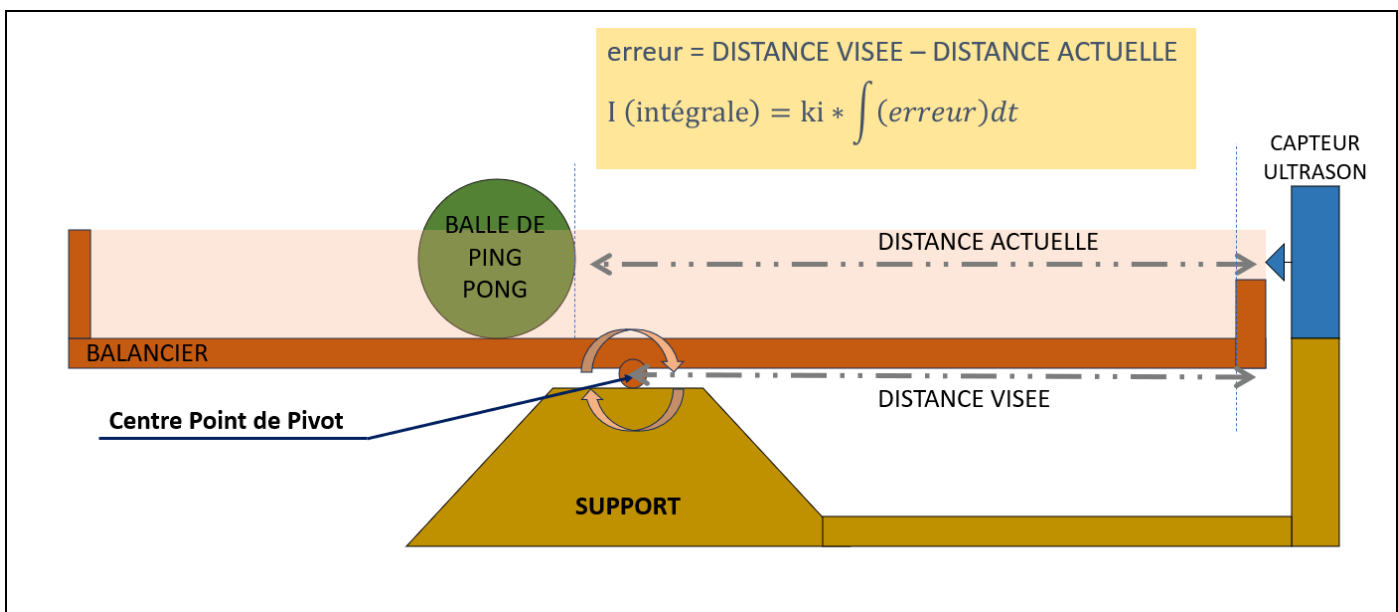
Risque d'instabilité si Kp trop grand.

Intégral (I)

Réagit à l'accumulation des erreurs passées.

Élimine l'erreur résiduelle.

Risque de lenteur ou oscillations si Ki mal réglé.



6. Utilisation d'un correcteur à action dérivée - D

Nous venons de voir que nous sommes en mesure d'avoir un système rapide, précis mais avec une tendance à avoir des dépassements. Nous allons essayer de voir si en agissant sur le paramètre dérivateur, nous pouvons résoudre ce problème.

Expérimentation 9 - Configurez l'action **P** (proportionnel - kp) et l'action **I** (intégrale - ki) avec les valeurs des gains déterminées précédemment. **Mettre** la valeur de l'action **D** (dérivée) à 1 (kd = 1).

Expérimentation 10 - Lancez l'exécution et **décrivez** le comportement.

Expérimentation 11 - Réaliser plusieurs essais jusqu'à obtenir un comportement satisfaisant. (Il faut agir sur les trois paramètres P, I et D)

Question 9 - Conclure quant à l'effet du correcteur proportionnel intégral dérivateur sur le comportement du système.

Synthèse prof

D (Dérivé) — Le contrôleur dérivé (D) agit comme un "frein" pour éviter les à-coups et les oscillations du système. Voici comment il fonctionne :



1. Détection des changements brusques :

- Il mesure la vitesse de variation de l'erreur (à quel point l'erreur augmente ou diminue rapidement).
- C'est comme si votre système pouvait "anticiper" les dépassements.

2. Action stabilisatrice :

- Quand le système approche trop vite de la consigne, le terme D ralentit le mouvement pour éviter de dépasser la cible.
- À l'inverse, si l'erreur augmente rapidement, il donne un "coup de pouce" pour réagir plus vite.

3. Equation :

En pratique, on simplifie souvent en prenant $K_d \times (\text{Erreur} - \text{Erreur_précédente})$

4. Réglage de K_d :

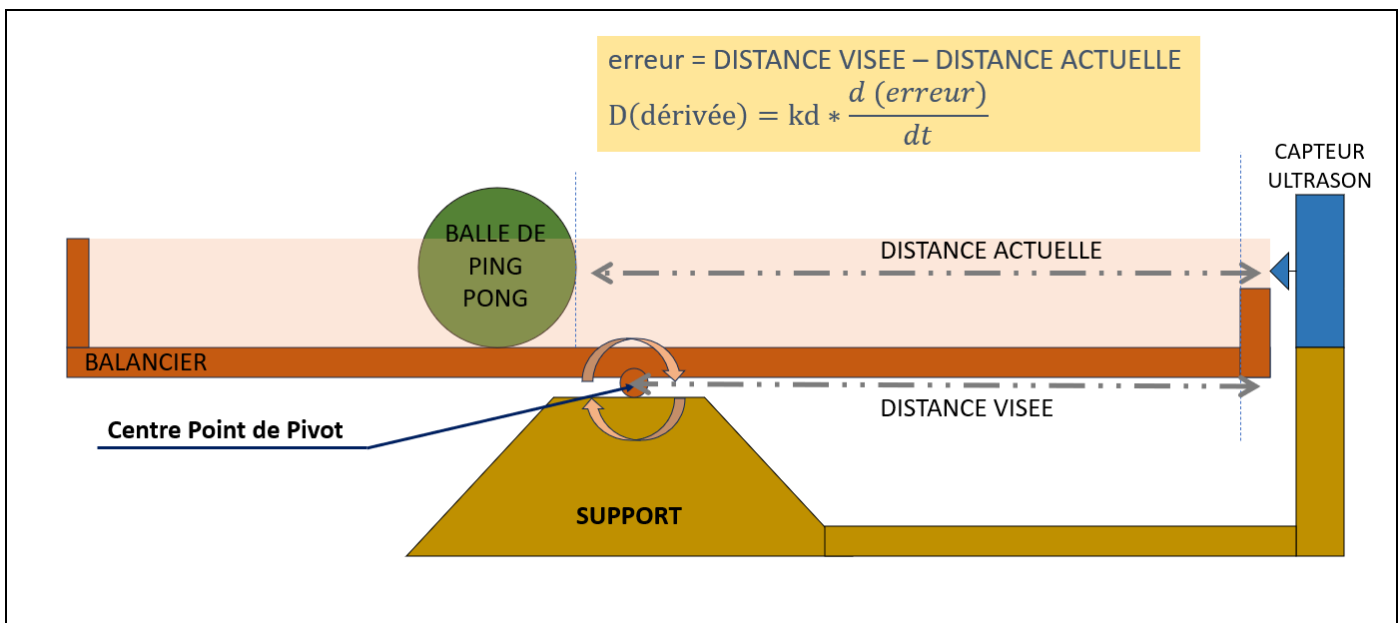
- K_d trop faible : Le système oscille encore un peu avant de se stabiliser
- K_d trop fort : Le système devient lent et "mou", comme sur-amorti
- Méthode : Comme pour K_p et K_i , on teste progressivement

5. Pourquoi l'utiliser ?

- Réduit les oscillations
- Améliore la stabilité
- Permet d'utiliser un K_p plus élevé sans risque de dépassement

6. Inconvénients :

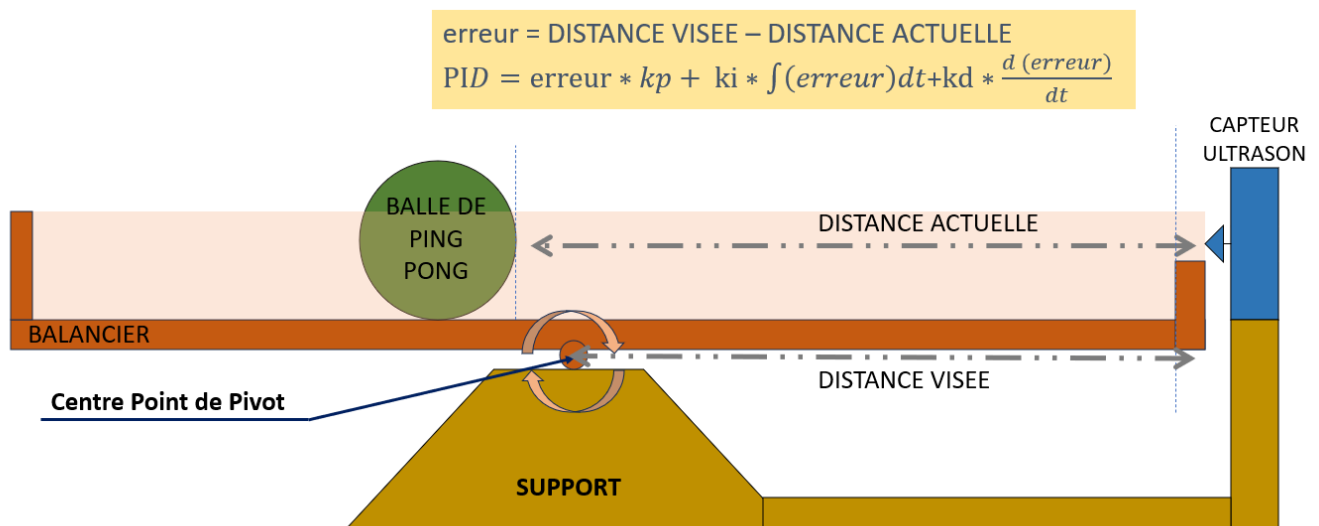
- Très sensible au bruit des capteurs (peut nécessiter un filtrage)
- Peut ralentir excessivement le système si mal réglé

**7. Synthèse**

Nous pouvons également obtenir la valeur PID en ajoutant les trois valeurs du contrôleur ci-dessus.

De plus, l'obtention des trois constantes **kp**, **ki** et **kd** est appelée réglage d'un système de contrôle PID. Les trois valeurs correctes doivent être trouvées en saisissant des valeurs différentes.





Question 10 - Réalisez un tableau de synthèse permettant d'évaluer l'influence positive et négative de chaque critère du PID.

Caractéristique	Proportionnel (P)	Intégral (I)	Dérivé (D)
Rôle principal	Réagit à l'erreur instantanée	Élimine l'erreur persistante	Anticipe et amortit les variations brusques
Avantages	Réponse rapide	Supprime l'erreur résiduelle	Stabilise, réduit les oscillations
Inconvénients	Peut laisser une erreur résiduelle	Peut rendre le système lent ou instable	Sensible au bruit des capteurs
Effet d'un gain trop élevé	Oscillations	Dérive lente ou instabilité	Système trop lent, réponse "molle"
Formule	$P = K_p \times \text{erreur}$	$I = K_i \times \int (\text{erreur}) dt$	$D = K_d \times d(\text{erreur})/dt$
Analogie	J'appuie sur la pédale de frein de mon véhicule inversement proportionnel à la distance face à un obstacle	Je m'aperçois que ma vitesse est toujours inférieure à la vitesse souhaitée pendant un laps de temps, alors j'augmente progressivement l'accélération	J'anticipe une descente en réduisant l'accélération avant que la vitesse ne dépasse 130 km.h ⁻¹