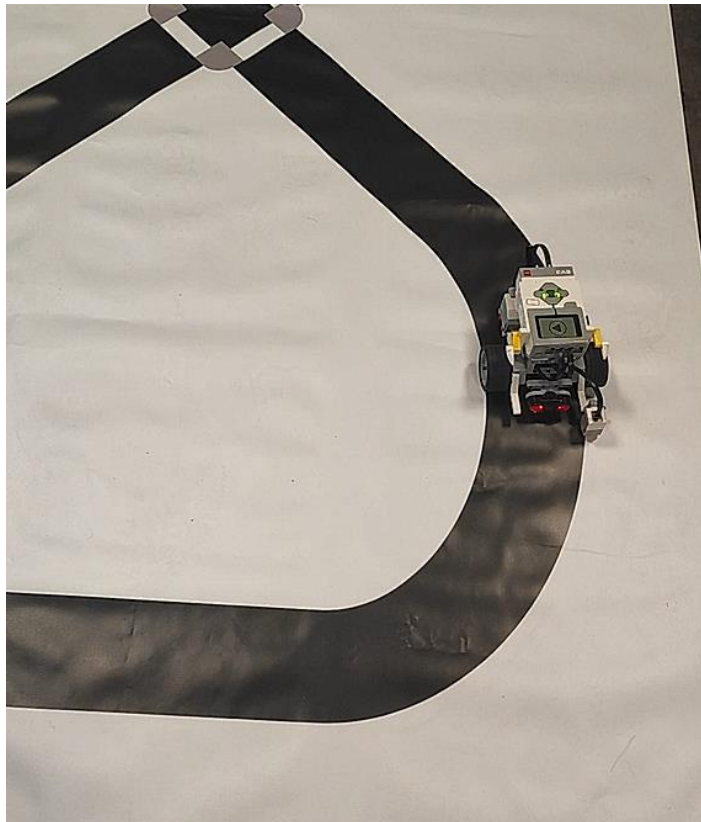


Algorithmique, robotique et systèmes temps réel (RO52)

TP N°2 - SUIVI DE LIGNE

Sébastien DOUTRELUINGNE, Samuel BERNARD

28/04/2023



Introduction

Dans le cadre de notre UV RO52, nous devons réaliser un programme permettant au robot EV3 de LEGO de suivre un tracé noir au sol. Ce travail pratique nous a notamment permis de mettre en application les notions abordées lors de cours magistraux.

Afin de le guider dans sa trajectoire, le robot dispose d'un capteur couleur situé à l'avant. Nous allons commander le robot en fonction des informations reçu par le capteur couleur : si le robot dévie de sa trajectoire, il devra tourner de lui-même d'un angle calculé en fonction de l'erreur calculé (en utilisant les informations du capteur couleur) et d'autres paramètres qui dépendent du correcteur appliqué. Cela lui permettra de rester sur la trajectoire tracée.

Le capteur couleur sera placé entre la ligne noire et blanche de sorte à détecter la répartition entre la couleur noire et de la couleur blanche sur le sol. La consigne sera fixée à 50% (autant de noir que de blanc) pouvoir suivre la ligne correctement. Ainsi, lorsque le capteur couleur détectera une présence du noir à plus de 50%, le robot devra tourner à gauche, et inversement lorsqu'il détectera du blanc.

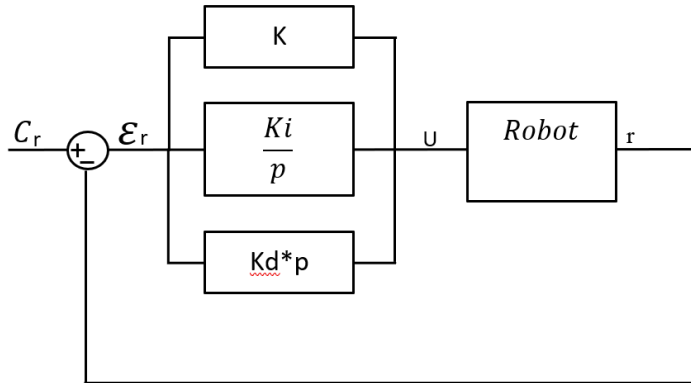
Afin d'effectuer nos tests sur l'asservissement de l'angle du robot, nous disposons de 3 circuits de routes différents afin d'apprendre les commandes les plus optimisées pour un type de circuit donné.

Objectif

L'objectif de ce TP est de mettre en pratique les connaissances théoriques que nous avons acquises durant les séances de cours et de déterminer une commande latérale du robot lui permettant de suivre une ligne noire tout en optimisant la vitesse de déplacement et la stabilité de la trajectoire du robot. Pour cela, nous avons divisé notre démarche en trois étapes et trois algorithmes progressifs : tout d'abord le correcteur proportionnel (P) permettant le suivi de ligne dans une trajectoire rectiligne, le correcteur proportionnel intégral (PI) puis dérivé (PID) afin de suivre correctement le tracé dans des circuits avec de fortes courbures.

Démarche

Ainsi, notre programme respectera la logique et les étapes suivantes afin de corriger continuellement la trajectoire du robot.



C_r : Consigne de réflexion (fixée à 50% : autant de noir que de blanc)

E_r : Erreur entre la réflexion obtenue par le capteur couleur et la consigne ($E_r = r - C_r$).

Correcteurs K, Ki, Kd : Correcteur appliqué sur le système pour corriger la trajectoire du robot.

U : La commande du robot. Ici, on applique un correcteur sur l'angle auquel le robot doit tourner. Dans le rapport, elle sera appelée δ

r : Réflexion de la surface du sol obtenue par le capteur couleur du robot. La réflexion obtenue est un pourcentage entre 0 et 100%. Si la réflexion se rapproche de 0, cela veut dire que la couleur noire est très présente.

Correcteur Proportionnel (P)

Présentation de la commande

Nous souhaitons dans un premier temps que le robot suive une trajectoire rectiligne. Pour cela, nous allons utiliser un correcteur proportionnel. Cette correction consiste simplement à faire tourner le robot d'un angle δ correspondant à une vitesse de rotation, à droite ou à gauche proportionnellement à la valeur de l'erreur instantanée. Pour cela, l'erreur est donc multipliée par un coefficient k permettant d'obtenir une valeur d'angle cohérente : $\delta = k \cdot \varepsilon$.

Ainsi, si une valeur d'erreur sera négative, signifiant une réflexion plus petite que 50% et donc que le robot dévie à droite, alors le correcteur permettra au robot de tourner à droite proportionnellement à la valeur de l'erreur.

Par ailleurs, plus la valeur de l'erreur est importante, plus celle de l'angle le sera également pour corriger la trajectoire du robot.

Implémentation

Pour implémenter le correcteur proportionnel, nous avons défini la fonction *commandeP* au sein de la classe *Command*. Cette fonction prend en paramètre la valeur de réflexion obtenue grâce à la classe *ColorSensorControl* :

```
def commandeP(self, reflection):
    angle = 0
    erreur = abs(reflection-self.consigne)
    if reflection > 50:
        angle = self.k*erreur
    else:
        angle = -self.k*erreur

    return angle
```

Cette fonction calcule tout d'abord la valeur de l'erreur comme la valeur absolue entre la réflexion du capteur de couleur (entre 0% et 100%) et la consigne fixée à 50%.

Ensuite, on détermine la valeur de l'angle de rotation du robot. Si la valeur de réflexion est plus grande que 50%, alors l'erreur est positive et l'angle également. Dans le cas contraire, alors la valeur de l'angle sera négative.

Enfin, la valeur de l'angle est retournée pour permettre au robot de corriger sa trajectoire.

Observations




Nous avons par la suite réalisé des tests du correcteur proportionnel sur différents circuits afin d'ajuster et de déterminer les valeurs de k et de vitesse du robot les plus adaptées.

Le correcteur proportionnel permet de commander l'angle du robot en appliquant un coefficient k à l'erreur. Ainsi, si l'erreur tend vers 0, la rotation du robot diminuera alors que lorsque l'erreur augmente, sa rotation augmente également.

Il est donc nécessaire de déterminer une valeur de k suffisamment grande pour que la correction ait un effet sur la trajectoire du robot. Sinon, le robot se comporterait comme s'il était en boucle ouverte. Cependant, une valeur trop grande de k risquerait de déstabiliser le système avec une correction de l'angle de rotation trop grande, et créant ainsi des oscillations autour de la consigne. Le robot oscillerait donc également autour du tracé.

Il est également important de définir une valeur de vitesse qui soit cohérente par rapport au tracé du circuit, afin que le robot puisse avancer suffisamment rapidement tout en respectant le tracé du circuit.

Après différents tests, voici les différentes valeurs de vitesse et de k que nous avons déterminées, et les observations que nous nous sommes faites :

Circuit			
Correcteur			
K	1	0.5	0.5
Vitesse	200	150	250
Observations	Le robot a du mal à tourner efficacement dans les virages.	Le robot a du mal à tourner efficacement dans les virages. La correction est lente ce qui entraîne une sortie de trajectoire.	Le robot arrive à bien suivre la ligne tout en ayant une vitesse importante.

En effet, nous avons constaté que sur une ligne droite certaines valeurs de k trop élevées engendraient des oscillations du robot autour du tracé. Avec cette valeur de vitesse et de k , le robot est parfaitement capable de suivre la ligne droite correctement.

Toutefois pour les circuits avec des virages, il était nécessaire d'augmenter la valeur de k afin de corriger la trajectoire du robot dans les virages. En effet, dans les virages l'erreur peut être importante. Si la valeur de k est faible, alors le robot prendra beaucoup trop de temps à corriger sa trajectoire et l'erreur ne fera qu'augmenter : le robot sortirait de sa trajectoire. Cependant, si la valeur de k est trop élevée, alors le robot risquerait de surréagir et devenir instable rapidement ce qui ferait augmenter de plus en plus son erreur.

Cependant, nous avons identifié différents avantages pour le correcteur proportionnel dont :

- La simplicité d'implémentation
- Sa capacité à fournir une correction immédiate dès qu'une erreur est détectée.

Correcteur proportionnel intégral (PI)

Présentation de la commande

Afin de s'adapter aux circuits avec des virages à fortes courbures, il est nécessaire de faire évoluer le correcteur proportionnel précédemment présenté. En effet, ce type de circuit peut engendrer un éloignement important du robot sur le trajet initial et nécessite donc l'introduction d'une nouvelle commande se basant sur le calcul de l'intégrale de l'erreur.

Pour se faire, le correcteur intégral (I) calcule une sortie proportionnelle à la somme des erreurs accumulées : plus l'erreur est grande et dure longtemps, plus la vitesse de rotation sera importante (commande du robot). Le correcteur intégral est souvent utilisé pour éliminer l'erreur de positionnement statique et améliorer la précision du système. Cette commande se calcule de la façon suivante :

$$\delta = k_i \cdot \tau \cdot \sum_{j=0}^n \varepsilon(j)$$

Avec :

- Un coefficient k_i dont la valeur doit être déterminée par expérimentation.
- Un paramètre τ qui correspond au temps entre deux applications du correcteur, fixé à 100ms.
- Une variable n correspondant au nombre d'erreurs additionnées depuis que la réflexion est supérieure ou inférieure à 50%.

Ainsi, si le robot dévie à gauche, alors la valeur de réflexion sera supérieure à 50% (détection de blanc majoritairement) et les valeurs d'erreur augmenteront et s'additionneront. La vitesse de rotation du robot sera alors de plus en plus importante pour corriger rapidement sa trajectoire. Ainsi, lorsque le robot aura suffisamment redressé sa trajectoire à droite, d'avantage de noir sera détecté et la valeur de réflexion deviendra inférieure à 50%. À cet instant, la somme des erreurs sera réinitialisée à 0 car le robot aura corrigé sa déviation sur la gauche.

Dans notre cas, ce correcteur intégral (I) sera combiné avec le correcteur proportionnel pour former le correcteur proportionnel intégral (PI), qui permettra de corriger rapidement l'erreur du robot et éviter les instabilités :

$$\delta = k \cdot \varepsilon(n) + k_i \cdot \tau \cdot \sum_{j=0}^n \varepsilon(j)$$

Implémentation

Pour implémenter le correcteur proportionnel intégral, nous avons défini deux fonctions au sein de la classe *Command* : *computeSumError* permettant de calculer la somme des erreurs à chaque itération et

prenant donc en paramètre la valeur de l'erreur actuelle, et *commandePI* implémentant le correcteur PI et prenant en paramètre la valeur de réflexion obtenue grâce à la classe *ColorSensorControl* :

```
def computeSumError(self, erreur):
    self.sumErreur += erreur
    if abs(self.sumErreur) > self.sumBorn:
        if self.sumErreur < 0:
            self.sumErreur = -self.sumBorn
        else:
            self.sumErreur = self.sumBorn
```

Cette première fonction *ComputeSumError* met tout d'abord à jour la valeur de l'attribut *sumErreur* initialisé à 0 à la création d'un objet de classe *Command*, en ajoutant la valeur actuelle de l'erreur.

Nous avons également défini des bornes *sumBorn* de cette somme permettant d'éviter des trop grandes valeurs de corrections. Si, la somme des erreurs va au-delà des bornes définies, alors la valeur de cette somme est ramenée à la valeur de la borne. Ainsi, lors d'une trop grande déviation de trajectoire, la vitesse de rotation du robot sera bornée.

```
def commandePI(self, reflection):
    angle = 0
    erreur = abs(reflection-self.consigne)
    self.computeSumError(erreur)

    angle = self.k*erreur + self.ki*self.tau*self.sumErreur

    if reflection > 50:
        self.currentColor = "White"
    else:
        angle = -angle
        self.currentColor = "Black"

    if self.currentColor != self.previousColor:
        self.sumErreur = 0

    self.previousColor = self.currentColor

    return angle
```

La fonction *commandePI* reprend la logique d'implémentation de la fonction *commandeP*. Ainsi, après avoir calculé l'erreur actuelle et mis à jour la somme des erreurs avec la fonction *computeSumError*, la valeur de l'angle de rotation est calculée en respectant la formule du correcteur PI définie précédemment.

Cependant, il est nécessaire d'évaluer également si le robot a corrigé ou non sa trajectoire. Ainsi, nous avons défini deux attributs *currentColor* et *previousColor* identifiant la couleur majoritairement détectée respectivement à l'instant actuel et à l'instant précédent. Ainsi, si la valeur de réflexion est supérieure à 50%, alors la couleur majoritaire est le blanc et donc *currentColor* vaut White, sinon il vaut Black.

Ensuite, les valeurs de *currentColor* et *previousColor* sont comparées. Si elles sont identiques, cela signifie que le robot est toujours en train de corriger sa trajectoire et qu'il est nécessaire de poursuivre l'addition

des erreurs. Cependant si elles sont différentes, cela indique que le robot a redressé suffisamment sa trajectoire pour détecter davantage de l'autre couleur que celle détectée précédemment. La valeur de la somme de erreurs contenues dans l'attribut *sumErreur* est alors réinitialisée à 0.




La valeur de *currentColor* est ensuite copiée dans *previousColor* pour réaliser cette même comparaison le tour suivant.

Observations

Nous avons par la suite réalisé des tests du correcteur proportionnel intégral sur les mêmes circuits afin d'ajuster et de déterminer cette fois-ci les valeurs de k , k_i , et de vitesse du robot les plus adaptées.

Le correcteur proportionnel intégral permet en plus de la correction calculée par la commande P de corriger des grands écarts de trajectoire grâce au calcul de l'intégral de l'erreur. Ainsi, il est nécessaire de déterminer une valeur de k_i suffisamment grande pour que la trajectoire du robot soit corrigée en cas de grand écart. Sinon, le robot se comporterait comme avec la commande P seulement. Cependant, une valeur trop grande de k_i risquerait également de déstabiliser le système avec une correction de l'angle de rotation trop grande, et créant ainsi des oscillations autour de la consigne.

Après différents tests, voici les différentes valeurs de vitesse, de k et de k_i que nous avons déterminées, et les observations que nous nous sommes faites :

Circuit			
Correcteur			
K	0.5	0.5	0.5
K _i	1	1	0.5
Vitesse	200	150	250
Observations	Le robot appréhende mieux les grands virages sans dévier de sa trajectoire.	Le robot appréhende mieux le grand virage à la moitié du parcours mais dévie de sa trajectoire lors des virages serrés.	Le robot arrive à bien suivre la ligne, sans osciller, tout en ayant une vitesse importante.

En effet, nous avons constaté que sur une ligne droite il n'était pas nécessaire de définir une grande valeur de k_i car le robot n'est pas susceptible de sortir de sa trajectoire dans l'absence de virages. Une valeur trop élevée de k_i risquerait de faire surréagir le robot et qu'il devienne instable. Avec ces valeurs, le robot est parfaitement capable de suivre la ligne correctement.

Toutefois pour le circuit avec des virages, l'introduction de la commande intégrale a permis au robot de corriger rapidement sa trajectoire lors de virages larges. Cependant, une valeur trop grande de k_i engendrerait les mêmes impacts sur le comportement du robot. Nous avons ainsi fixé la valeur de la borne de l'intégrale à 150 pour éviter des trop grandes rotations du robot.

Le robot peut parfois cependant sortir complètement de sa voie. Ce comportement a notamment pu être constaté lors de virages serrés. En effet, lorsque le robot arrive au premier virage, celui-ci sort directement du tracé et le correcteur PI calcule alors une grande vitesse de rotation pour redresser la trajectoire du robot. Cependant, le second virage étant proche, le capteur pourrait dès la première rotation dépasser la ligne et ainsi sortir complètement du tracé.

Correcteur Proportionnel Intégrateur Dérivé (PID)

Présentation de la commande

Avec le correcteur précédent, nous avons remarqué que le robot pouvait sortir de la voie en raison d'une correction trop prononcée. Cela est dû à la somme des erreurs multipliée par k_i qui était devenue trop importante. La commande de l'angle du robot était donc trop importante et le robot sortait de l'autre côté de la voie rendant impossible un suivi de trajectoire.

Pour pallier ce problème, nous avons tout d'abord pensé à augmenter la valeur du k et diminuer celle de k_i . Toutefois, cette solution apportait de nouvelles oscillations. Nous avons donc introduit un nouveau type de correcteur basé sur la dérivée de l'erreur.

Le correcteur dérivé calcule une sortie proportionnelle à la dérivée de l'erreur du système. Cela signifie que le correcteur dérivé répond à la vitesse à laquelle l'erreur change avec le temps. Le correcteur dérivé est utilisé pour améliorer la réponse dynamique du système et réduire les oscillations. Cette commande se calcule de la façon suivante :

$$\delta = \frac{k_d}{\tau} \cdot (\varepsilon(n) - \varepsilon(n-1))$$

Avec :

- Un coefficient k_d dont la valeur doit être déterminée par expérimentation.
- Un paramètre τ qui correspond au temps entre deux applications du correcteur, fixé à 100ms.
- Une variable n correspondant au nombre d'erreurs additionnées depuis que la réflexion est supérieure ou inférieure à 50%.
- $\varepsilon(n-1)$ correspond à l'erreur précédente.

L'avantage d'un correcteur dérivé est son effet stabilisant. En effet, celui-ci permet de réduire les grandes variations de l'erreur (oscillations) qui peut y avoir sur sa trajectoire. Il permet donc de stabiliser le système et d'améliorer le temps de réponse. Toutefois, il ne s'utilise rarement seul puisqu'il amplifie les fluctuations rapides ce qui le rend très sensibles aux perturbations mais aussi parce que si l'erreur est constante, aucune action n'est effectuée.

Si nous combinons les trois correcteurs ensemble, nous obtenons un correcteur PID (proportionnel-intégral-dérivé), permettant d'améliorer la stabilité, la précision et la réponse dynamique du système.

$$\delta = k \cdot \varepsilon(n) + k_i \cdot \tau \cdot \sum_{j=0}^n \varepsilon(j) + \frac{k_d}{\tau} \cdot (\varepsilon(n) - \varepsilon(n-1))$$

Implémentation

Pour implémenter le correcteur proportionnel intégral et dérivé, nous avons créé la nouvelle fonction *commandePID* tout en reprenant le process de la fonction pour le proportionnelle intégrale. En effet, cette fonction utilise *computeSumError* de la classe *Command* afin de calculer la somme des erreurs à chaque itération (voir *commandePI*). Nous avons ajouté à cette fonction l'effet de dérivé qui consiste à multiplier un coefficient *kd* à la différence entre l'erreur actuelle et l'erreur précédente divisé par le temps.

```
def commandePID(self, reflection):
    angle = 0
    erreur = abs(reflection-self.consigne)
    self.computeSumError(erreur)

    angle = self.k*erreur + self.ki*self.tau*self.sumErreur +
            (self.kd/self.tau)*(erreur - self.previousErreur)

    if reflection > 50:
        self.currentColor = "White"
    else:
        self.currentColor = "Black"
        angle = -angle

    if self.currentColor != self.previousColor:
        self.sumErreur = 0

    self.previousColor = self.currentColor
    self.previousErreur = erreur
    return angle
```




Nous avons défini un nouvel attribut *previousErreur* qui identifie l'erreur précédente. Celle-ci permet de détecter un changement rapide du tracé du circuit afin de pouvoir corriger rapidement la trajectoire du robot. Si le tracé est une ligne droite, donc sans changement de trajectoire, la différence entre l'erreur et l'erreur précédente est nulle ce qui permet au robot de continuer normalement sa trajectoire. Toutefois, si le virage est très prononcé, le robot sera capable de corriger avec efficacité sa trajectoire à l'aide de la constante *kd* amplifiée par τ contrairement au correcteur PI qui prenait plus de temps à se corriger. Le PID est donc très efficace pour des circuits qui possèdent des virages à fortes courbures.

Observations

Nous avons fini par réaliser des tests du correcteur proportionnel intégral dérivé sur les mêmes circuits afin d'ajuster et de déterminer les valeurs optimales de k , k_i , k_d .

Le correcteur proportionnel dérivé permet en plus du correcteur PI de corriger rapidement la trajectoire du robot sur des virages très serrés (brusques). Ainsi, il est nécessaire de déterminer une valeur de k_d suffisamment grande pour que la trajectoire du robot soit corrigée rapidement sur des circuits complexes. Cependant, une valeur trop grande de k_d risquerait également de déstabiliser le système avec une correction de l'angle de rotation trop grande et trop rapide (notamment pour les circuits avec de petits virages).

Après différents tests, voici les différentes valeurs de vitesse, de k et de k_i et k_d que nous avons déterminées, et les observations que nous nous sommes faites :

Circuit			
Correcteur			
Kp	0.5	0.8	0.5
Ki	1.5	1.5	0.5
Kd	0.1	0.2	0.1
Vitesse	200	150	250
Observations	Le robot appréhende très bien les grands virages sans dévier de sa trajectoire. Nous pouvons toutefois noter les quelques oscillations sur les lignes droites	Le robot appréhende bien le grand virage à la moitié du parcours et est très efficace sur les virages serrés en fin de parcours.	Le robot arrive à bien suivre la ligne, avec quelques oscillations rapides due au correcteur dérivé.

Nous avons observé que le coefficient k_d doit être assez faible car le correcteur dérivé est très sensible aux variations rapides ce qui peut le rendre instable. En effet, la valeur du coefficient est amplifiée par le fait de la dérivée de l'erreur par rapport au temps. En d'autres termes, puisque le temps entre deux correcteurs est de 0.1ms, cela signifie que le coefficient k_d est multiplié par 10.

De plus, nous avons remarqué que la mise en place du correcteur PID est complexe dans le choix des coefficients étant donné qu'il y a 3 coefficients à paramétrer.

Le correcteur PID a donc permis de corriger le problème de déviation du robot sur le circuit avec de fortes courbures (2^{ème}). Cela peut s'expliquer par la correction rapide des variations de parcours effectuée par la partie dérivée du correcteur.

Conclusion

Pour conclure, ce TP nous a permis de mettre en pratique les connaissances abordées durant les cours magistraux et d'en voir une application direct dans un travail avec une approche ludique qu'a été le suivi de ligne d'un robot.

En effet, les expérimentations nous ont permis de comprendre les différentes commandes proportionnelles, intégrales puis dérivées et leur formule associée.

Enfin, l'approche progressive d'intégration des différents correcteurs nous a permis d'identifier leurs utilités pour ainsi adapter les différentes valeurs de coefficients et concevoir un système de suivi de ligne efficace.