TP1: Contrôle de robots unicycle et bicyclette

Planification et contrôle
ROB316

Yu WANG

December 5, 2021

Remarque : Tous les résultats (code, images, etc.) présentés dans ce rapport peuvent être trouvés sur ce gitHub : https://github.com/Rescon/ROB316

1 Introduction

1.1 Résumé du cours

Ce cours présente un certain nombre de modèles cinématiques et d'algorithmes pour L'algorithme du contrôleur (PID).

Le modèle cinématique est défini par les éléments d'assemblage et les liaisons mécaniques qui, en robotique, relient les vitesses dans le domaine cartésien et les vitesses des articulations du robot. À l'aide du modèle cinématique, il existe de nombreuses applications, dont trois doivent être contrôlées : la stabilisation de la trajectoire, le suivi de la trajectoire et la stabilisation de la configuration.

L'algorithme du contrôleur est le contrôleur PID, qui visent à minimiser l'erreur entre la position cible et la position réelle. En utilisant l'algorithme PID, une convergence rapide peut être obtenue en sélectionnant les composants idéaux (P, I, D) tout en évitant les oscillations d'amplitude excessive. Il est possible de calculer la commande $u(t)$ proportionnelle à l'erreur, son intégrale et sa dérivée, ce qui fait converger $e(t)$ vers zéro.

1.2 description de TP

Dans ce TP, nous allons mettre en oeuvre des méthodes de contrôle très simples, à base de contrôleur PID, pour le contrôle de robots unicycle et bicyclette.

Les programmes fournis permettent de simuler des robots se déplaçant sur le plan en suivant des modèles unicycle et bicyclette en exploitant un contrôle que vous devez programmer. Il y a quatre missions:

- contrôler le robot unicycle.
- contrôler le robot bicyclette vers un point.
- contrôler le robot bicyclette vers une point.
- contrôler le robot bicyclette selon un chemin.

2 Unicycle

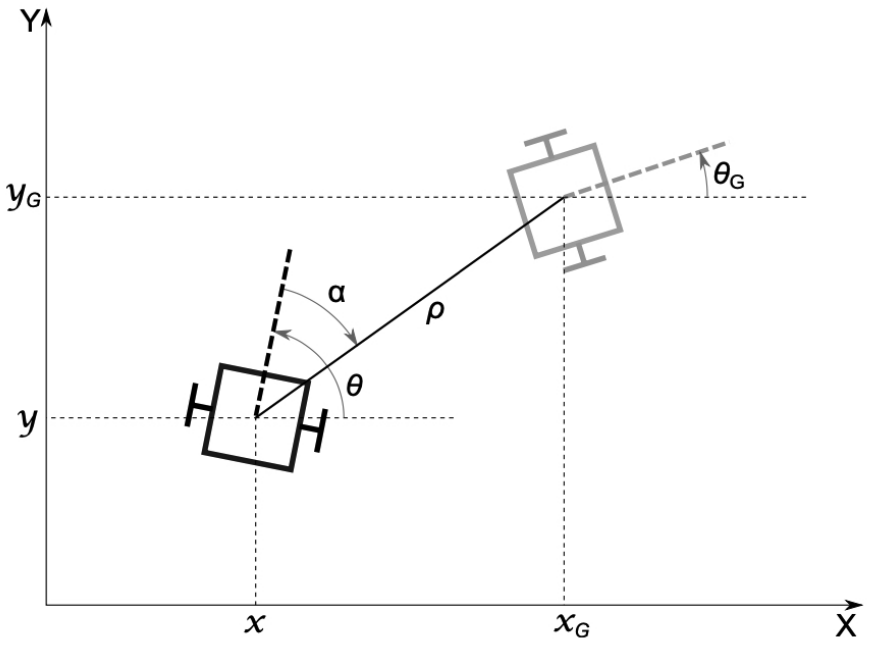


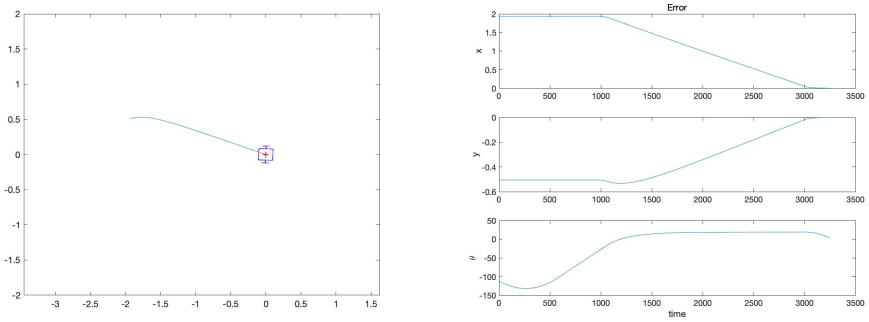
Figure 1: Contrôle d'un unicycle vers une pose

Dans cette section, on va écrire un contrôleur PID qui guide un unicycle vers une pose fixée (Figure 1). La formule associée est présentée comme Eq. (2.1).

$$\begin{aligned} \rho &= \sqrt{(x_G - x)^2 + (y_G - y)^2} \\ \alpha &= \arctan \frac{y_G - y}{x_G - x} - \theta \\ v &= K_\rho \times \rho \\ \varphi &= K_\alpha \times \alpha \end{aligned} \tag{2.1}$$

Lorsque le robot est proche du but ($\rho > 0.05$), pour aller plus vite vers le but, sans faire de grands arcs de cercle, on peut faire tourner le robot sur place si sa direction est trop éloignée de la direction du but : $v = 0, si|\alpha| > \alpha_{max}$.

Lorsque le robot est proche du but ($\rho < 0.05$), le contrôleur utilisera la variable β pour calculer la vitesse de rotation où $\beta = \theta_G - \theta$ et $\omega = K_\beta \times \beta$.



(a) Trajectoire

(b) Évolution d'erreur

Figure 2: Résultat de performance

En choisissant $K_\rho = 15, K_\alpha = 5, K_\beta = 15$ et $\alpha_{max} = \frac{\pi}{4}$, on peut obtenir la trajectoire et l'évolution de l'erreur présentées sur la Figure 2. En utilisant la fonction de benchmark **UnicycleToPoseBenchmark**, on obtient un temps moyen de 1923.7619 ms pour arriver autour de la cible.

K_ρ	K_α	K_β	Temps(ms)
5	5	15	8003.3333
30	5	15	9573.9048
15	3	15	7522.3333
15	15	15	4711.4286
15	5	3	3319.4762
15	5	40	2355.8571

Table 1: Table de paramètre

Le tableau 1 est un exemple qui montre l'impact de différents paramètres sur

les performances (temps d'arrivée) de l'algorithme. On peut analyser l'impact de ces trois paramètres :

- K_ρ déterminera la vitesse de convergence du robot. Si elle est trop petite, le robot mettra trop de temps à atteindre la cible. Si elle est trop grande, le robot oscillera après avoir atteint la zone proche de la cible. (oscillation du petit ρ).
- K_α déterminera la vitesse de rotation . Si elle est trop petite, nous ferons un arc trop grand autour de la cible. Si elle est trop grand, le système réagira de manière excessive à la correction et se dirigera vers la cible.
- K_β, comme K_α, affecte la vitesse de rotation à proximité de la cible. Si elle est trop grand, il y a un risque de manquer la direction de la cible et de faire demi-tour. Si elle est trop petite, le robot tournera trop lentement autour de lui-même. Toutefois, ces modifications n'affectent pas de manière significative le temps moyen pour atteindre l'objectif.

3 Bicyclette

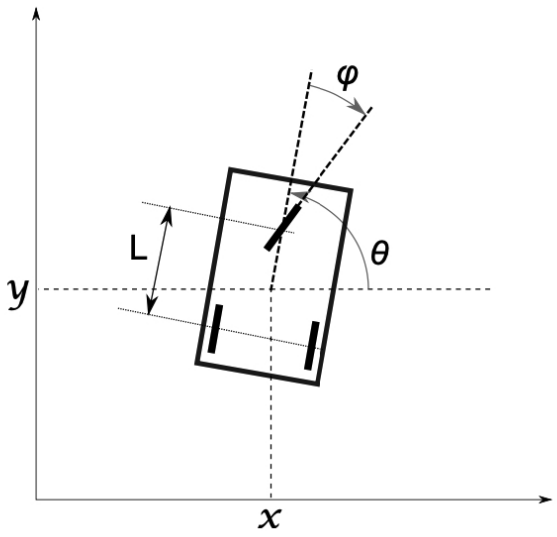


Figure 3: Modèle bicyclette

Dans cette section, on va écrire un contrôleur PID qui contrôler le modèle bicyclette (Figure 3). Les équations du modèle sont comme Eq. 3.1.

$$\begin{aligned} \dot{x} &= v\cos\theta\cos\varphi \\ \dot{y} &= v\sin\theta\cos\varphi \\ \dot{\theta} &= \frac{v}{L}\sin\varphi \\ \dot{\varphi} &= \omega \end{aligned} \tag{3.1}$$

3.1 Question 2 - contrôle de bicyclette vers un point

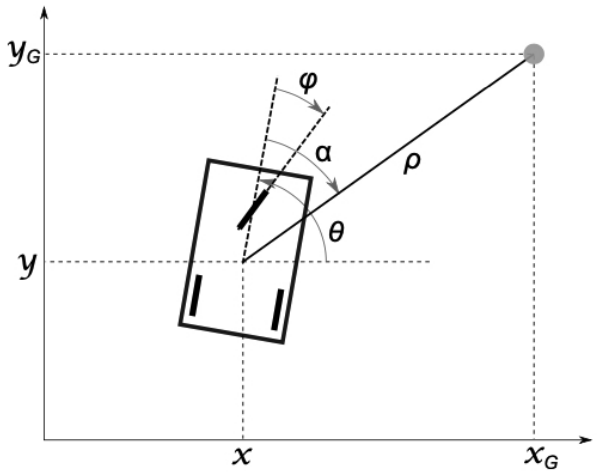
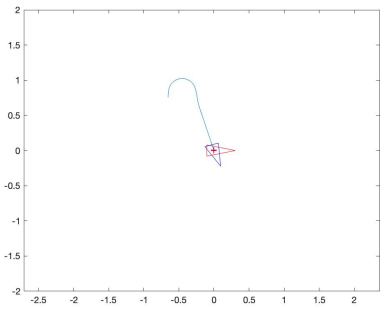


Figure 4: Contrôle d'un modèle bicyclette vers un point

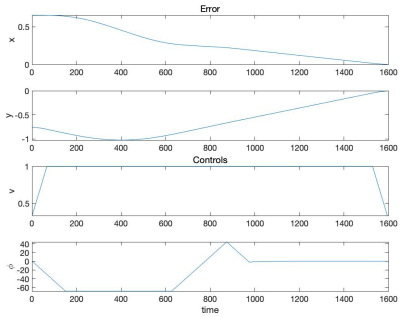
Tout d'abord, on écrit un contrôleur proportionnel qui guide le robot vers un point (Figure 4). Pour cela, on écrit un contrôleur à partir des formules comme Eq. 3.2.

$$\begin{aligned} \rho &= \sqrt{(x_G - x)^2 + (y_G - y)^2} \\ \alpha &= \arctan\frac{y_G - y}{x_G - x} - \theta \\ v &= K_\rho \times \rho \\ \varphi &= K_\alpha \times \alpha \end{aligned} \tag{3.2}$$

En choisissant $K_\rho = 20$ et $K_\alpha = 10$, on peut obtenir la trajectoire et l'évolution de l'erreur présentées sur la Figure 5. En utilisant la fonction de benchmark **BicycleToPointBenchmark**, on obtient un temps moyen de 1370.1429 ms pour arriver autour de la cible.



(a) Trajectoire



(b) Évolution d'erreur et de contrôles

Figure 5: Résultat de performance

K_ρ	K_α	Temps(ms)
10	10	1540.619
40	10	1353.1429
20	5	1369.3333
15	20	5601.3333

Table 2: Table de paramètre

Le tableau 2 est un exemple qui montre l'impact de différents paramètres sur les performances (temps d'arrivée) de l'algorithme. On peut contrôler cette modèle avec K_ρ pour contrôler la vitesse et K_α pour contrôler la vitesse angulaire.

3.2 Question 3 - contrôle de bicyclette vers une position

On écrit un contrôleur proportionnel qui guide le robot vers une position (Figure 6). Pour cela, on écrit un contrôleur à partir des formules comme Eq. 3.3.

$$\begin{aligned} \rho &= \sqrt{(x_G - x)^2 + (y_G - y)^2} \\ \alpha &= \arctan \frac{y_G - y}{x_G - x} - \theta \\ \beta &= \theta_G - \arctan \frac{y_G - y}{x_g - x} \\ v &= K_\rho \times \rho \\ \varphi &= K_\alpha \times \alpha + K_\beta \times \beta \\ avec K_\beta &< 0 \end{aligned} \tag{3.3}$$

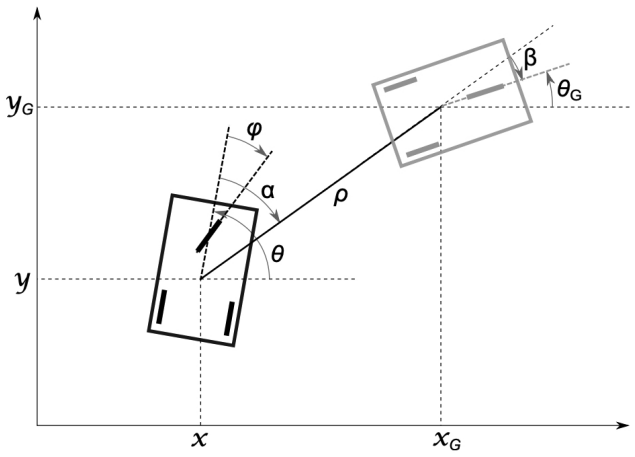
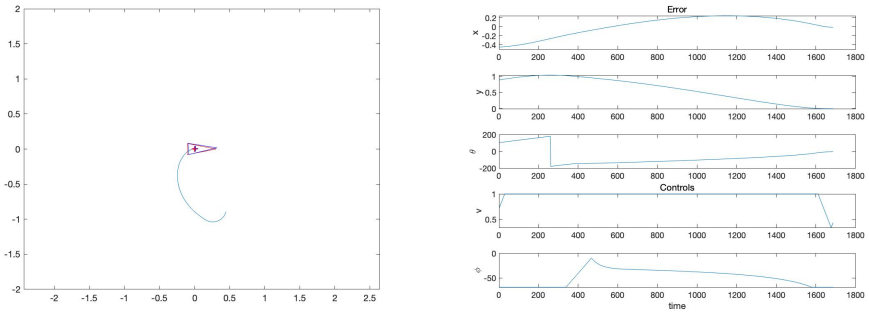


Figure 6: Contrôle d'un modèle bicyclette vers une position



(a) Trajectoire

(b) Évolution d'erreur et de contrôles

Figure 7: Résultat de performance

En choisissant $K_\rho = 30, K_\alpha = 10$ et $K_\beta = -5$, on peut obtenir la trajectoire et l'évolution de l'erreur présentées sur la Figure 7. En utilisant la fonction de benchmark **BicycleToPoseBenchmark**, on obtient un temps moyen de 1576.6667 ms pour arriver autour de la cible. Le modèle de bicyclette ne permet pas de tourner sur place ni de revenir en arrière : pour atteindre la cible de la meilleure façon avec la bonne orientation, le robot doit être ajusté à la bonne orientation dès qu'il a une distance suffisante de la cible. Pour une trop grande valeur de K_α, on arrivera à la cible avec la mauvaise direction et on tournera à l'infini pour trouver la bonne combinaison de position et de direction. Comme le système ne peut pas se translater, K_β est la correction d'orientation une fois l'orientation correcte est obtenue pour atteindre la position. Si elle est trop petite, il n'y a pas de correction, si elle est trop grande, le système sera trop sensible et arrivera avec la mauvaise direction.

K_ρ	K_α	K_β	Temps(ms)
20	10	-5	1641.8571
40	10	-5	1565.7619
30	5	-5	8232.7143
30	15	-5	6178.0476
30	10	-3	6179.2381
30	10	-8	2238.4762

Table 3: Table de paramètre

Le tableau 3 est un exemple qui montre l'impact de différents paramètres sur les performances (temps d'arrivée) de l'algorithme.

3.3 Question 4 - contrôle de bicyclette selon un chemin

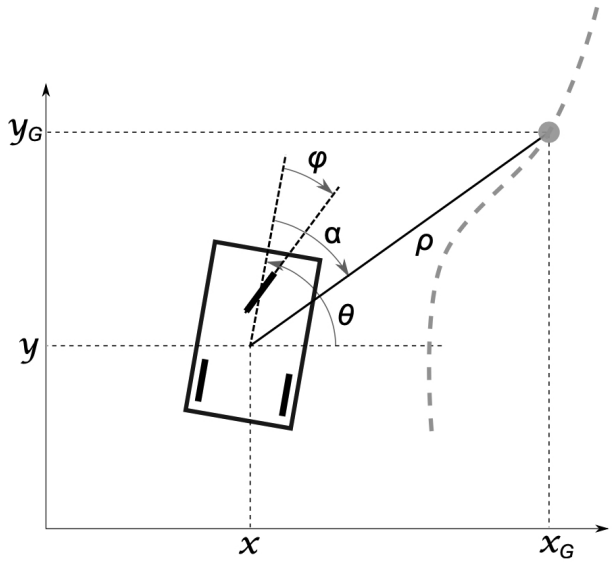
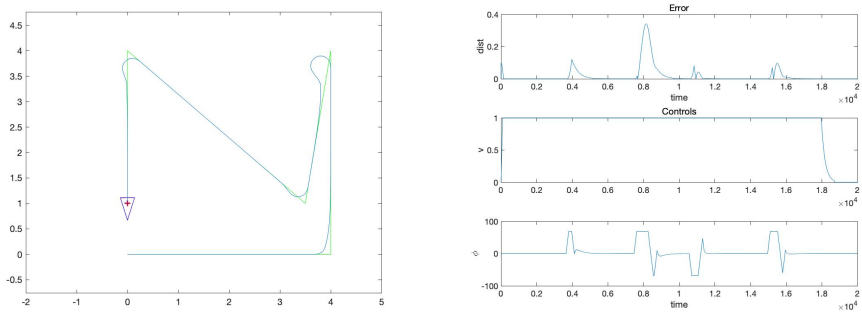


Figure 8: Contrôle d'un modèle bicyclette selon un chemin

On écrit un contrôleur proportionnel qui guide le robot selon un chemin (Figure 8).

L'idée est de définir le premier point de la trajectoire comme un point cible temporaire. Ensuite, nous fixons une valeur seuil et lorsque la distance entre le robot et ce point de la trajectoire est inférieure à ce seuil, nous considérons qu'il a atteint ce point et nous mettons à jour le point cible temporaire à ce point. Nous calculons ensuite le vecteur unitaire du point cible au point suivant



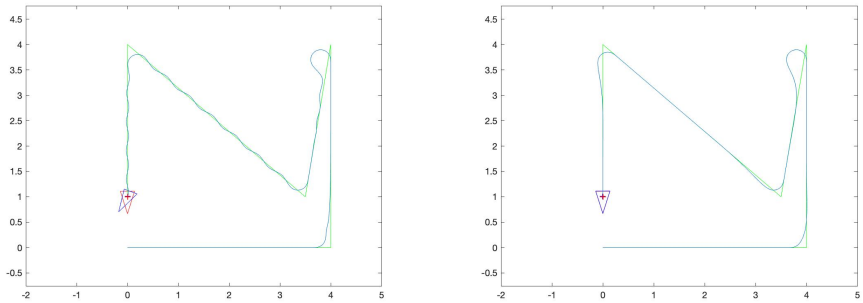
(a) Trajectoire (b) Évolution d'erreur et de contrôles

Figure 9: Résultat de performance

de la trajectoire et l'ajoutons au point cible actuel pour obtenir le nouveau point cible. Pour chaque point cible, nous le contrôlons en utilisant le même contrôleur PID que précédemment.

En choisissant $K_\rho = 5, K_\alpha = 8$ et $seuil = 0.4$, le robot suit avec succès la trajectoire vers la cible avec une erreur de 357.7656.

En choisissant une grande K_α, la sensibilité du système signifie qu'il y aura de nombreuses oscillations dans chaque partie de la trajectoire avant que le système n'atteigne la stabilité. Cependant, pour un petit K_α, le robot aura des arcs très larges lorsqu'il se déplace vers de nouvelles sections (comme la Figure 10). K_ρ joue un rôle dans l'erreur et la vitesse de convergence du système pendant le virage ; lorsque K_ρ est trop petite, le robot n'atteindra pas le point cible final à temps, et lorsque K_ρ est trop grande, il augmente l'amplitude des oscillations dans une certaine mesure, mais a peu d'effet sur l'erreur totale.



(a) $K_\alpha = 20$, erreur = 441.1961 (b) $K_\alpha = 3$, erreur = 383.2351

Figure 10: la sensibilité de K_α