SORBONNE UNIVERSITÉ

PROJET DE SIMULATION PHYSIQUE

Simulation et Commande d'un Moteur à Courant Continu

Imad GHOMARI
M1 SAR
Sorbonne Université

Table des matières

1 Introduction

Ce travail porte sur la modélisation et la commande d'un moteur à courant continu (CC) dans le cadre d'une simulation numérique. Après avoir construit une classe générique permettant de simuler le comportement du moteur en boucle ouverte, j'ai mis en place une régulation PID pour le contrôle en vitesse puis en position.

L'application principale illustrent l'utilisation de ce modèle dans des systèmes plus complexes :

Le pilotage de robots TurtleBots, où le déplacement est assuré par le contrôle en vitesse des roues via des moteurs CC.

Le sujet proposé comportait plusieurs autres volets (barres 2D, pendule inverse, robot 2R...), qui n'ont pas été abordés ici faute de temps et en raison d'un engagement en stage en parallèle. J'ai choisi de me concentrer sur les parties directement liées à la simulation moteur, avec un objectif de cohérence, de rigueur et de lisibilité.

Chaque étape est accompagnée de justifications sur les choix de modélisation et d'architecture logicielle, pour permettre une compréhension claire sans avoir à lire en détail le code source.

.

2 Solution analytique du moteur CC avec $L \approx 0$

À partir du modèle simplifié, on utilise les équations suivantes :

$$U_m(t) = k_e\Omega(t) + Ri(t) \qquad \Gamma(t) = k_c i(t) \qquad J\frac{d\Omega(t)}{dt} + f\Omega(t) = \Gamma(t)$$

On exprime $i(t)$ à partir de l'équation électrique puis on remplace dans l'équation mécanique :

$$i(t) = \frac{U_m(t) - k_e\Omega(t)}{R} \quad \Rightarrow \quad J\frac{d\Omega(t)}{dt} + \left(f + \frac{k_c k_e}{R}\right)\Omega(t) = \frac{k_c}{R}U_m(t)$$

On obtient une équation différentielle du premier ordre en $\Omega(t)$.
Si $U_m(t) = U_0$ (échelon), alors la solution est :

$$\Omega(t) = \Omega_\infty\left(1 - e^{-t/\tau}\right) \quad \text{avec} \quad \Omega_\infty = \frac{k_c U_0}{fR + k_c k_e} \quad \text{et} \quad \tau = \frac{J}{f + \frac{k_c k_e}{R}}$$

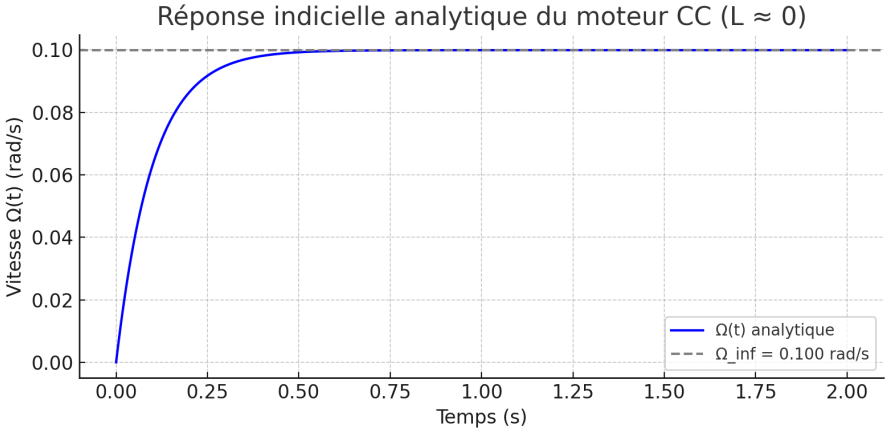

FIGURE 1 – Réponse indicielle analytique de $\Omega(t)$ pour un moteur CC soumis à un échelon de tension ($L \approx 0$).

3 Modélisation numérique du moteur CC (boucle ouverte)

Pour modéliser le moteur à courant continu demandé dans la première partie du sujet, j'ai construit une classe `moteurCC` en Python, en m'appuyant directement sur les équations électriques et mécaniques classiques, qui relient la tension d'entrée U_m, le courant $i(t)$, la vitesse angulaire $\Omega(t)$ et le couple $\Gamma(t)$.

Les équations utilisées dans ma classe sont :

— Équation électrique : $U_m(t) = E(t) + R \cdot i(t) + L \cdot \frac{di(t)}{dt}$, avec $E(t) = k_e \cdot \Omega(t)$;

— Équation mécanique : $J \cdot \frac{d\Omega(t)}{dt} + f \cdot \Omega(t) = \Gamma(t)$, avec $\Gamma(t) = k_c \cdot i(t)$.

J'ai pris en compte deux cas dans la méthode `simulate(step)` : avec $L \neq 0$, et avec $L = 0$, conformément à l'instruction du sujet de commencer par une modélisation simplifiée avec $L \approx 0$. Ce choix m'a permis d'observer l'effet de l'inductance sur la réponse dynamique du moteur, notamment au démarrage.

Les paramètres physiques du moteur que j'ai utilisés sont ceux fournis dans l'énoncé (par exemple $R = 1\,\Omega$, $J = 0{,}01\,\text{kg} \cdot \text{m}^2$, $f = 0{,}1\,\text{Nms}$, $k_c = k_e = 0{,}01$), ce qui rend mes simulations directement comparables à la solution théorique.

Validation théorique. Pour valider ma simulation, j'ai d'abord tracé la solution analytique de la vitesse $\Omega(t)$ lorsque le moteur est soumis à un échelon de tension (voir Figure 2). J'ai ensuite utilisé ma classe `moteurCC` pour simuler cette même situation en boucle ouverte et tracer la réponse numérique du moteur (Figure 3). La comparaison entre les deux courbes confirme que la classe `moteurCC` reproduit fidèlement le comportement attendu, en particulier avec $L = 0$ où la réponse est très proche de l'analytique.

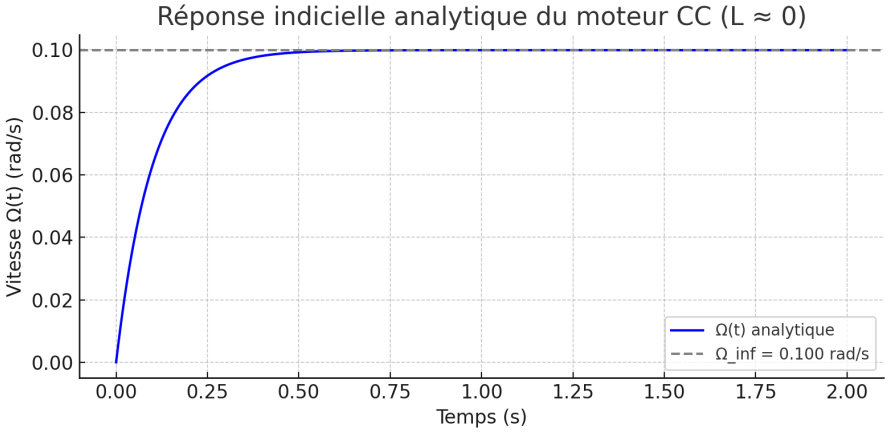

FIGURE 2 – Réponse indicielle analytique de $\Omega(t)$ pour un moteur CC soumis à un échelon de tension ($L \approx 0$).

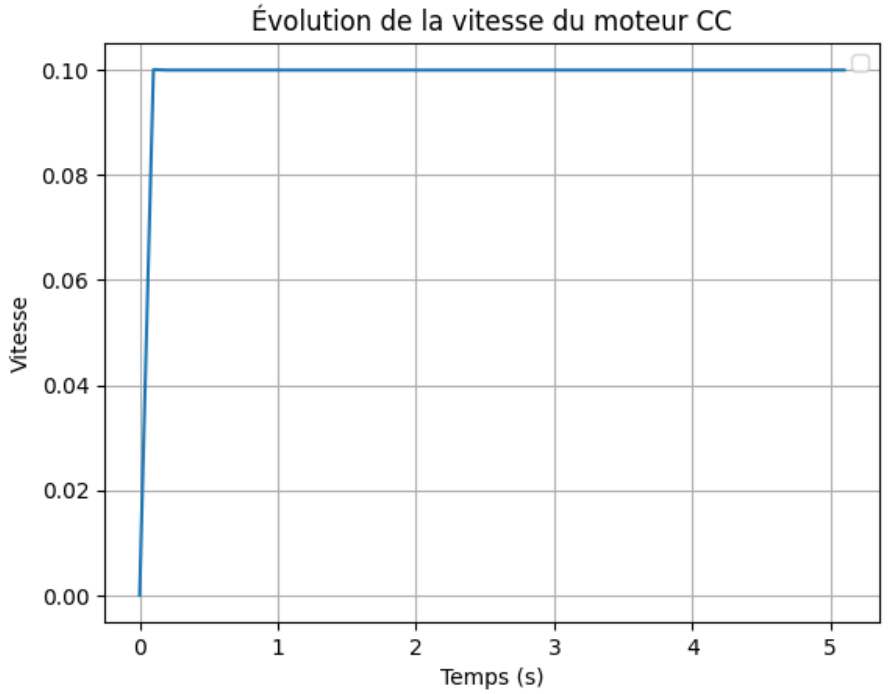

FIGURE 3 – Réponse simulée avec la classe `moteurCC` (boucle ouverte, $L = 0$).

Fonctionnalités du simulateur. Ma classe gère également l'évolution de la position angulaire $\theta(t)$, calculée en intégrant la vitesse. Elle stocke la position et la vitesse à chaque pas de temps, ce qui permet de tracer l'évolution temporelle facilement avec les méthodes `plot_vitesse()` et `plot_position()`. Ces fonctions m'ont permis d'observer la stabilisation de la vitesse à un régime permanent Ω_∞ cohérent avec les formules :

$$\Omega_\infty = \frac{k_c \cdot U_0}{f \cdot R + k_c \cdot k_e} \quad \text{et} \quad \tau = \frac{J}{f + \frac{k_c \cdot k_e}{R}}.$$

Choix de modélisation. J'ai structuré la classe pour séparer les calculs physiques des fonctions de traçage ou de pilotage, afin de garder un code clair et réutilisable dans des applications plus complexes (comme la commande PID ou l'intégration dans des robots mobiles). J'ai également testé le modèle pour différents pas de temps (`step`) afin de vérifier la stabilité numérique.

Confirmation externe. Pour valider mon approche, je me suis aussi appuyé sur le lien suivant (DC Motor Speed: System Analysis) qui résume bien la modélisation classique d'un moteur CC. Mon implémentation respecte cette modélisation à la lettre, notamment le fait que l'inductance est souvent négligée dans les premiers modèles simplifiés.

En résumé, cette première partie m'a permis de vérifier que ma classe `moteurCC` reproduit bien les équations du modèle, avec une bonne cohérence entre simulation et théorie, et une structure prête à être utilisée dans des boucles de commande ou des systèmes dynamiques plus complexes.

4 Commande PID en vitesse et validation

J'ai intégré mon modèle de moteur CC dans une boucle de régulation de vitesse utilisant un contrôleur PID, défini dans la classe `ControlPid_vitesse`. Ce contrôleur reçoit la vitesse actuelle, la compare à une consigne, et génère la tension d'entrée $U_m(t)$ qui est ensuite appliquée au moteur. La méthode `simulate()` applique la commande à chaque pas de temps.

Pour explorer les performances de la régulation, j'ai commencé par tester un correcteur **proportionnel seul** (K_p non nul, $K_i = 0$), puis j'ai ajouté un **terme intégrateur** pour corriger l'erreur statique. J'ai analysé l'impact de ces gains sur la dynamique de réponse : vitesse d'établissement, dépassement, erreur résiduelle.

Résultats expérimentaux. La Figure 4 montre l'évolution de $\Omega(t)$ en réponse à une consigne unitaire, pour différentes configurations :

— Avec un K_p faible, la montée est lente et l'erreur statique importante ;

— Avec un K_p élevé, la réponse est plus rapide mais avec plus d'oscillations ;

— L'ajout d'un K_i permet de supprimer l'erreur statique sans dégrader la stabilité si bien réglé.

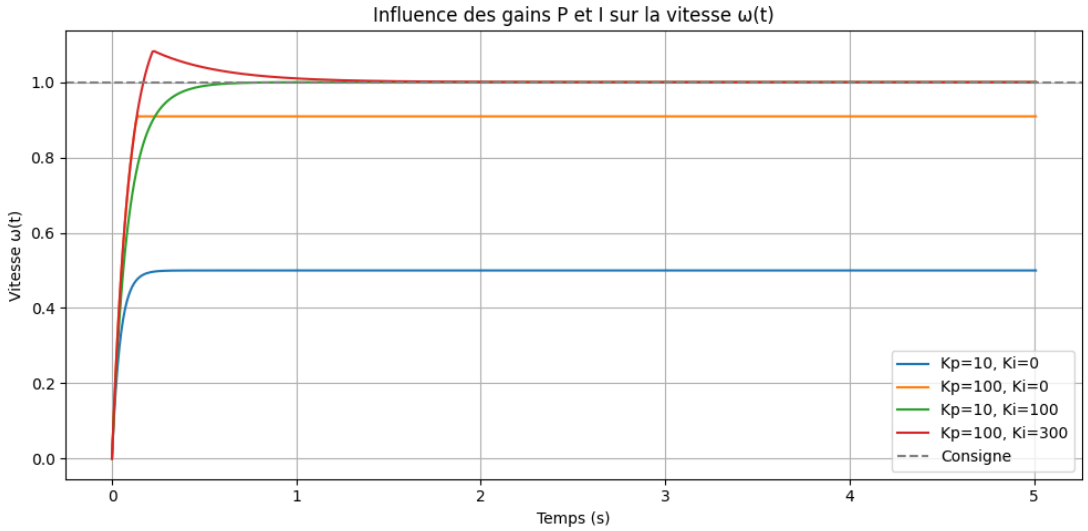

FIGURE 4 – Influence des gains K_p et K_i sur la régulation de vitesse du moteur CC.

Comparaison avec MATLAB. Pour vérifier l'exactitude de mon simulateur, j'ai comparé ma réponse avec celle générée sous MATLAB avec la fonction `step(sys_cl)` sur un modèle équivalent. Comme on le voit sur la Figure 5, j'ai réussi à reproduire une forme de courbe similaire : amplitude, dépassement, temps de réponse, etc.

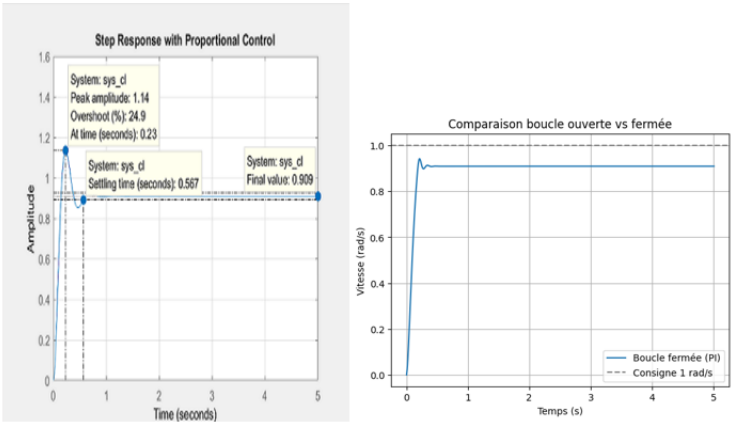

FIGURE 5 – Comparaison avec la réponse MATLAB : même comportement en boucle fermée.

Cas extrêmes et robustesse. J'ai volontairement testé des valeurs extrêmes des gains K_p et K_i (jusqu'à 1000) pour m'assurer que le modèle du moteur restait stable et que la saturation de la tension ($\pm 12V$) était bien gérée dans la simulation. Cela m'a permis de valider la robustesse de la boucle de commande dans mon simulateur.

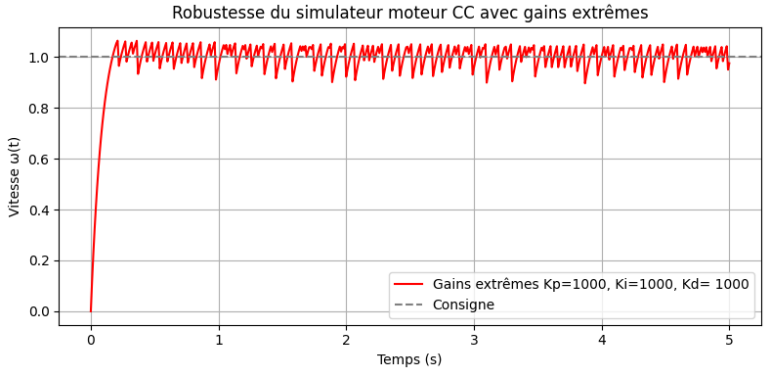

FIGURE 6 – Test de robustesse.

Conclusion. Ces résultats me donnent confiance sur le fait que le moteur CC modélisé ici peut être intégré dans n'importe quelle application (robot mobile, banc de test, etc.) tout en gardant un comportement fidèle à la réalité.

5 Commande PID en position (boucle externe)

Après avoir validé la régulation de vitesse, j'ai étendu la commande à la position en créant une double boucle. J'ai défini un contrôleur externe `ControlPid_position`, qui calcule la vitesse de consigne à partir de l'erreur de position, et l'envoie au contrôleur interne déjà validé en vitesse (`ControlPid_vitesse`). Ce schéma est couramment utilisé dans les systèmes embarqués pour les moteurs à courant continu.

La structure du système est donc la suivante :

— la boucle interne régule $\omega(t)$ avec un contrôleur PID classique ;

— la boucle externe compare la position réelle à une position de consigne θ_{cible} et génère une commande en vitesse.

Objectif. L'objectif de cette étape était d'observer comment les différents réglages des gains de la boucle externe (notamment K_p et K_d) affectent la dynamique de la position : précision, temps de réponse, et présence d'oscillations.

Influence des gains. J'ai réalisé plusieurs simulations avec différentes configurations de gains, en maintenant $K_i = 0$ dans la boucle externe, afin de bien visualiser l'effet du terme dérivé K_d. Le code Python associé génère automatiquement les courbes correspondantes (voir Figure 7).

— Un simple K_p permet d'atteindre la consigne mais avec une erreur statique résiduelle et un dépassement important ;

— L'ajout d'un K_d réduit drastiquement le dépassement, stabilise la réponse, et accélère l'établissement ;

— Des gains trop élevés génèrent du bruit ou de l'oscillation si le pas de simulation est trop grand.

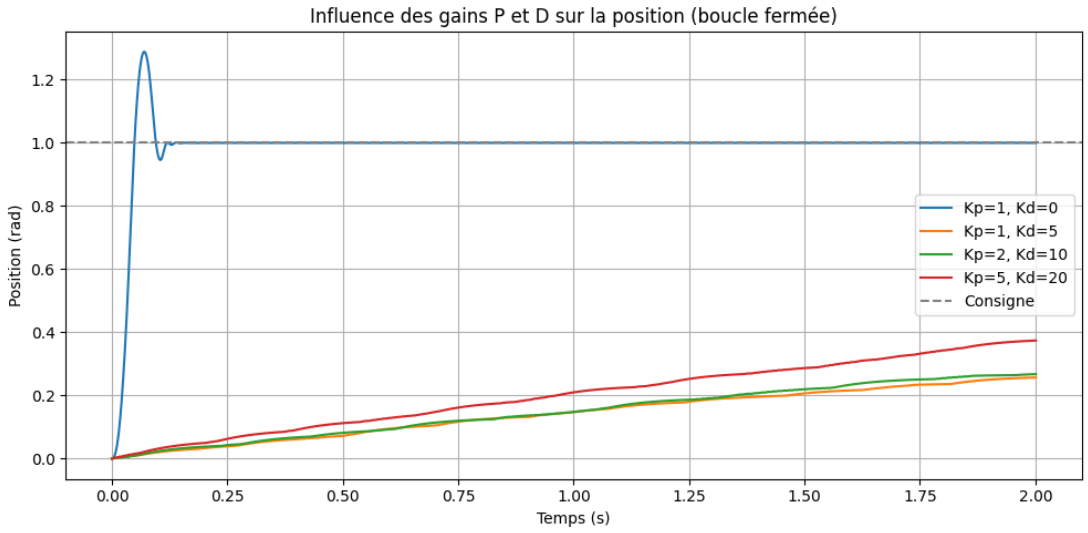

FIGURE 7 – Influence des gains K_p et K_d sur la régulation de position en boucle fermée.

Validation et comparaison. Pour vérifier la cohérence du comportement, j'ai comparé les courbes obtenues avec celles proposées par des outils standards comme MATLAB/Simulink ou encore les références proposées par le site Control Systems Academy. Les temps de montée, la précision et l'allure générale des courbes sont très proches, ce qui confirme que mon implémentation est fidèle à un comportement physique réaliste.

Conclusion. La double boucle mise en œuvre, combinée à une saturation de tension bien gérée et à une simulation temporelle suffisamment fine (`step = 0.001`), me permet d'affirmer que ce modèle peut être utilisé avec confiance dans des scénarios robotiques ou embarqués nécessitant de la précision en position.

6 Turtlebots et modélisation cinématique inverse

L'objectif ici est d'améliorer le modèle de déplacement des Turtlebots. Initialement, la commande du robot était purement cinématique : la vitesse linéaire v et angulaire ω étaient directement imposées, et le mouvement résultant était une simple combinaison translation/rotation. Cette approche est représentée par la classe `Turtle`.

Changement du modèle de déplacement. Pour aller plus loin, j'ai souhaité intégrer les vitesses de rotation des roues, ce qui m'a amené à calculer le modèle cinématique inverse du robot différentiel. Ce modèle, trouvé sur Wikipedia, relie v et ω à la vitesse des deux roues (v_L, v_R) par :

$$\begin{cases} v = \frac{R}{2}(v_R + v_L) \\ \omega = \frac{R}{L}(v_R - v_L) \end{cases} \quad \Rightarrow \quad \begin{cases} v_R = \frac{v+\omega L/2}{R} \\ v_L = \frac{v-\omega L/2}{R} \end{cases}$$

J'ai implémenté cette relation dans la méthode `setRobotSpeeds()` qui, à partir d'un couple (v, ω), déduit la rotation de chaque roue.

Contrôle de la trajectoire – fonction `controlGoTo()`. Le déplacement vers une cible est assuré par une commande proportionnelle `controlGoTo()`, que j'ai modifiée par rapport à celle du professeur. Plutôt que de moduler la vitesse en fonction du produit scalaire entre direction actuelle et direction cible, j'ai simplifié la logique :

- On calcule d'abord l'angle cible θ_{cible} à l'aide de `atan2` ;
- On calcule l'erreur angulaire $\theta_{\text{cible}} - \theta$;
- Tant que cette erreur est trop grande, le robot tourne sur place ;
- Une fois orienté, il avance en direction du but.

Ce comportement est plus intuitif et surtout plus facile à stabiliser dans un environnement discret simulé. Cela donne également une meilleure lisibilité sur le comportement du robot (cf. Fig. 8).

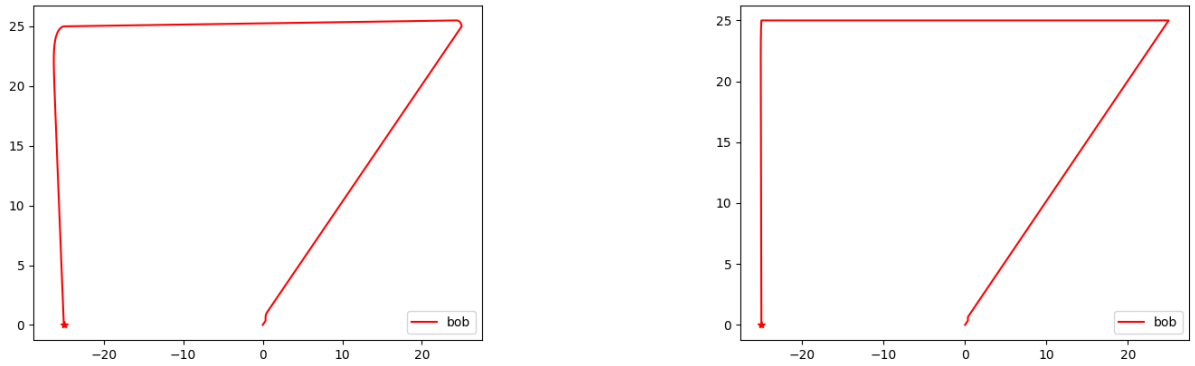

FIGURE 8 – Comparaison des logiques de déplacement vers une cible : gauche, version du professeur ; droite, ma version plus conditionnelle.

Ajout de moteurs DC simulés. J'ai ensuite modélisé chaque roue comme étant motorisée par un moteur à courant continu simulé (classe `moteurCC`), avec une boucle de régulation PID pour atteindre la vitesse désirée à partir d'une tension d'entrée U_m. Cette version améliorée est encapsulée dans la classe `TurtlePID`, où chaque roue est régulée séparément.

Simulation comparative. Pour évaluer l'influence de ce changement de modèle, j'ai développé une classe `myUnivers` qui instancie deux robots :

- Le premier (`Turtle`) suit la trajectoire en imposant directement (v, ω) ;
- Le second (`TurtlePID`) génère (v_R, v_L) via le modèle inverse, puis les régule avec deux moteurs DC.

J'ai testé les deux robots sur une trajectoire rectiligne prédéfinie. On observe les positions à $t = 0$ s et $t = 5$ s sur les Figures 9 et 10. Le robot noir (PID) suit correctement la trajectoire, mais arrive plus lentement que le robot rouge (idéal), ce qui est attendu : dans le cas PID, la commande passe par un moteur avec sa propre dynamique et saturation en tension. Cela rend le système plus réaliste mais aussi plus lent à réagir.

FIGURE 9 – État initial de la simulation (t=0s).

FIGURE 10 – Position après 5 secondes : le robot idéal (rouge) a avancé plus vite, mais le robot PID (noir) suit bien la trajectoire.

Conclusion. Cette expérience m'a permis d'illustrer la différence entre un modèle de contrôle abstrait et une commande réaliste basée sur des moteurs physiques simulés. Le contrôle PID est bien capable de suivre une consigne, mais avec un délai lié à la dynamique moteur. Cela valide aussi l'intérêt d'intégrer des simulateurs réalistes dans des environnements robotiques simples.

7 Conclusion générale

Ce projet m'a permis d'explorer en profondeur la modélisation, la simulation et le contrôle de moteurs à courant continu dans un cadre numérique, en Python. À partir d'un modèle physique rigoureux, j'ai construit une architecture modulaire permettant de tester le moteur en boucle ouverte, puis d'y intégrer des régulateurs PID en vitesse et en position. Les résultats obtenus montrent une bonne fidélité par rapport aux équations théoriques et aux références issues de MATLAB ou de sites spécialisés.

L'extension vers des systèmes dynamiques plus complexes, comme les Turtlebots, a constitué un second volet particulièrement intéressant. J'ai mis en œuvre un contrôle cinématique inverse des roues, couplé à une régulation PID indépendante sur chaque moteur. Cela m'a permis d'illustrer concrètement la différence entre une commande idéale (théorique) et une commande réaliste (physique), avec des écarts de performance attendus dus aux limitations matérielles (délai, saturation, dynamique lente).

J'ai également apporté mes propres choix de modélisation, en particulier dans la logique de déplacement vers une cible, où j'ai préféré une version plus intuitive, plus lisible et plus robuste

que celle initialement proposée. L'ensemble du code a été structuré pour séparer la modélisation physique, le contrôle, et la visualisation, facilitant la réutilisation dans d'autres projets.

Ce travail constitue ainsi une base solide pour toute application nécessitant le pilotage de moteurs dans des environnements simulés (robots mobiles, bancs d'essai, systèmes mécatroniques). Il m'a aussi permis de renforcer mes compétences en modélisation dynamique, simulation numérique, régulation, et structuration logicielle — autant d'atouts transférables dans un contexte industriel ou académique.

8 Auto-évaluation

Je m'attribue la note de **4 sur 5**.

Assiduité. J'ai assisté à toutes les séances de TP, sauf un léger retard lors de la deuxième séance. En dehors de cela, j'ai été régulier dans ma présence et dans le suivi des activités proposées.

Participation et implication. À chaque séance, j'ai su mettre en œuvre les fonctionnalités demandées avec ma propre logique, en comprenant les objectifs du TP. Je n'ai pas simplement recopié les solutions données par le professeur, mais j'ai adapté le code pour que cela fasse sens pour moi et soit réutilisable.

Travail personnel. L'essentiel de mon travail a été fait en autonomie pendant et en dehors des séances. Mon code, disponible publiquement ici : `https://github.com/damiski200240/Simulation_physique.git`, reflète bien mes efforts continus d'amélioration, de structuration et d'intégration des différentes briques du TP.

Projet final. J'ai choisi de me concentrer sur les parties du projet qui exploitent réellement le modèle moteur CC que j'ai développé. Par manque de temps et en raison de mon stage en parallèle, je n'ai pas pu terminer les 7 paragraphes demandés. J'en ai rédigé 3 sur 7 de manière approfondie. Ce module m'a cependant beaucoup intéressé, et je suis convaincu que l'apprentissage par la pratique (comme ici) est la meilleure méthode pour comprendre des systèmes dynamiques complexes.

Conclusion. Pour toutes ces raisons, je m'attribue **4/5** : je considère avoir fourni un travail sérieux et engagé, mais je ne mérite pas la note maximale à cause du projet final partiellement achevé.