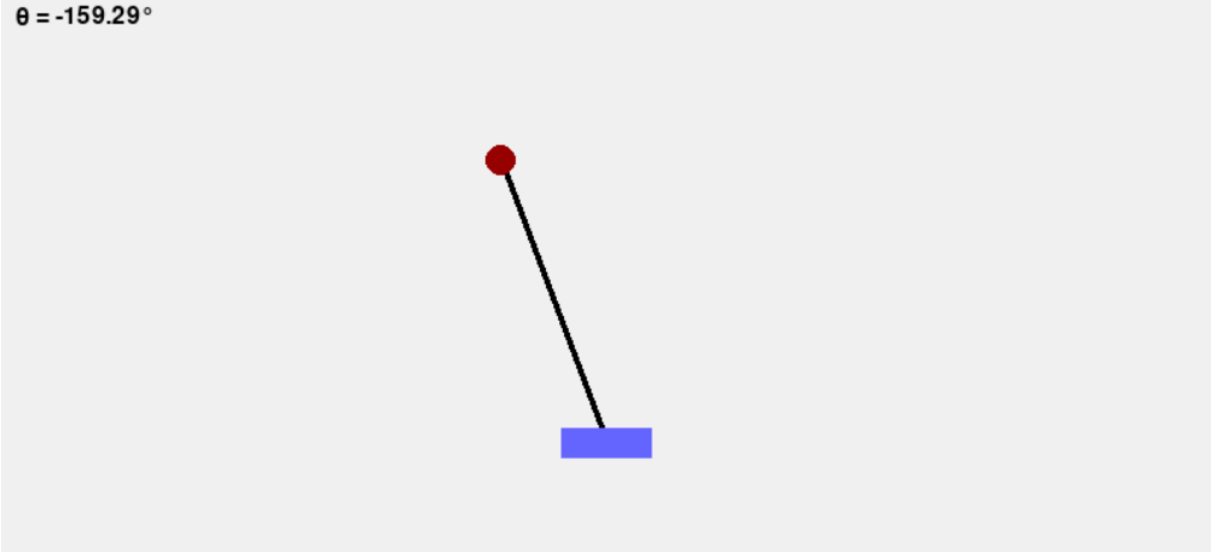
Sorbonne Université

Master 1 SAR

Projet de Simulation Physique



Réalisé par :
AHADDAD Amir
Année universitaire : 2024–2025

Table des matières

1 Introduction

La simulation physique joue un rôle essentiel dans le domaine de la robotique et des systèmes dynamiques. Elle permet de modéliser et de tester le comportement des systèmes mécaniques ou électromécaniques dans un environnement numérique, sans avoir besoin de prototypes physiques. C'est un outil puissant pour anticiper les performances, optimiser les paramètres de commande et mieux comprendre les phénomènes dynamiques.

Dans le cadre de ce projet, l'objectif est de développer un simulateur physique interactif en Python, utilisant la bibliothèque `Pygame`, permettant de modéliser, visualiser et commander différents systèmes dynamiques (moteur à courant continu, pendule, robot, etc.), tout en intégrant des contrôleurs PID et des interactions en temps réel.

Le travail s'est concentré principalement sur la simulation du moteur à courant continu (CC), composant de base dans de nombreux systemes. Le moteur a été modélisé puis simulé numériquement, testé en boucle ouverte, et contrôlé en boucle fermée à l'aide d'un correcteur PID. Il a ensuite été intégré dans plusieurs systèmes plus complexes tels qu'une centrifugeuse, un robot mobile de type TurtleBot, ou encore un pendule inversé.

Ce projet m'a permis d'acquérir un ensemble de compétences variées :

— modélisation physique de systèmes électromécaniques ;
— programmation orientée objet en Python ;
— simulation interactive avec `Pygame` ;
— conception et réglage de contrôleurs PID ;
— analyse des performances dynamiques et validation par simulation.

Le reste de ce rapport décrit les différentes étapes de ce travail, les choix techniques effectués, les résultats obtenus, ainsi que les pistes d'amélioration.

2 Simulation d'un moteur à courant continu (CC)

L'objectif de cette partie est de créer un modèle numérique du moteur à courant continu (CC), en s'appuyant sur sa modélisation physique. Le but est de simuler son comportement dynamique en considérant la tension $U_m(t)$ comme entrée, et en observant en sortie la vitesse de rotation $\Omega(t)$ ainsi que le couple moteur $\Gamma(t)$.

Nous allons d'abord établir le modèle mathématique du moteur à partir des équations électriques et mécaniques fondamentales. Une simplification sera ensuite appliquée afin d'obtenir une solution analytique exploitable. Cette modélisation sera implémentée numériquement à l'aide d'un simulateur Python, puis testée en boucle ouverte, avant d'être enrichie et régulée à l'aide d'un contrôleur PID.

2.1 Solution analytique

Pour étudier le comportement dynamique du moteur à courant continu, on part des équations fondamentales qui le régissent. On distingue deux parties : le modèle électrique et le modèle mécanique.

Équation électrique :

$$U_m(t) = E(t) + Ri(t) + L\frac{di(t)}{dt}$$

où :

— $U_m(t)$ est la tension appliquée au moteur,
— $E(t) = k_e \cdot \Omega(t)$ est la force contre-électromotrice,
— R est la résistance de l'induit,
— L est l'inductance (qu'on va négliger ici),

— $i(t)$ est le courant.

Sous l'hypothèse $L \approx 0$, l'équation devient :

$$i(t) = \frac{U_m(t) - k_e \cdot \Omega(t)}{R}$$

Équation mécanique :

$$J\frac{d\Omega(t)}{dt} + f \cdot \Omega(t) = \Gamma(t) \quad \text{avec} \quad \Gamma(t) = k_c \cdot i(t)$$

En remplaçant $i(t)$, on obtient :

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

Ce qui se réécrit :

$$\frac{d\Omega(t)}{dt} + \left(\frac{k_c k_e}{JR} + \frac{f}{J}\right)\Omega(t) = \frac{k_c}{JR}U_m(t)$$

Soit en posant :

$$a = \frac{k_c k_e}{JR} + \frac{f}{J}, \quad b = \frac{k_c}{JR}$$

On obtient une équation différentielle du premier ordre :

$$\frac{d\Omega(t)}{dt} + a\Omega(t) = bU_m(t)$$

Solution à un échelon de tension : On considère $U_m(t) = U_0$ constant. La solution analytique est alors :

$$\Omega(t) = \Omega_\infty \left(1 - e^{-at}\right), \quad \text{où} \quad \Omega_\infty = \frac{bU_0}{a}$$

Cette solution traduit une réponse de type exponentiel vers un régime permanent, typique d'un système du premier ordre.

2.2 Implémentation et validation du modèle numérique

Pour simuler le comportement du moteur à courant continu, une classe `MoteurCC` a été développée en Python. Elle modélise les équations électriques et mécaniques vues précédemment, en appliquant la méthode d'Euler explicite pour la discrétisation temporelle.

Le moteur est simulé avec les paramètres suivants :

— résistance de l'induit $R = 1\,\Omega$;

— inductance $L = 0.001\,\text{H}$ (négligée dans le calcul) ;

— constante de couple $k_c = 0.01\,\text{Nm/A}$;

— constante de force contre-électromotrice $k_e = 0.01\,\text{V.s/rad}$;

— inertie du rotor $J = 0.01\,\text{kg.m}^2$;

— frottement visqueux $f = 0.1\,\text{Nm.s}$.

Structure de la classe `MoteurCC` :

— Attributs : tension U_m, courant i, vitesse Ω, couple Γ, position θ ;

— Méthodes : `setVoltage`, `simule(step)`, `getSpeed`, `getTorque`, `getIntensity`, etc.

Une première simulation a été effectuée en appliquant un échelon de tension de 1 V, pour comparer la réponse numérique avec la solution analytique obtenue précédemment.

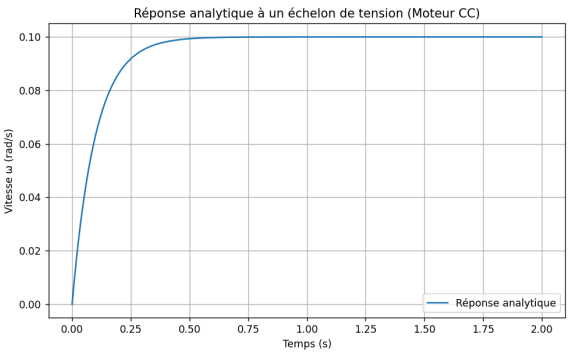


FIGURE 1 – Réponse théorique à un échelon

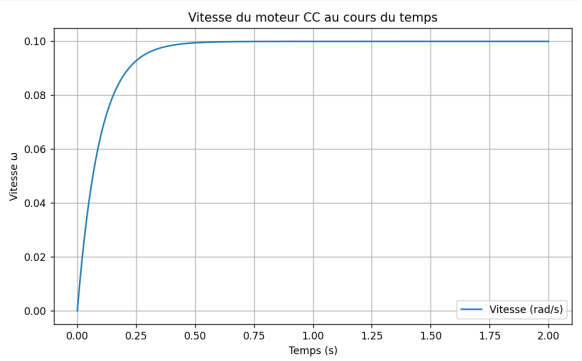


FIGURE 2 – Réponse analytique à un échelon (modèle)

Comparaison des réponses : Les deux courbes obtenues (figure 1 et figure 2) montrent une excellente concordance. La vitesse angulaire monte de manière exponentielle jusqu'à atteindre une valeur de régime permanent identique dans les deux cas, confirmant la validité du modèle numérique implémenté.

Influence de l'inductance L :

Influence de l'inductance L : Une étude complémentaire pour comparer deux modèles : l'un avec $L \approx 0$, l'autre avec $L = 0.1\,\text{H}$. Comme prévu, l'inductance introduit une légère inertie supplémentaire : la réponse est plus lente, le courant augmente plus progressivement, et le couple moteur met plus de temps à s'établir.

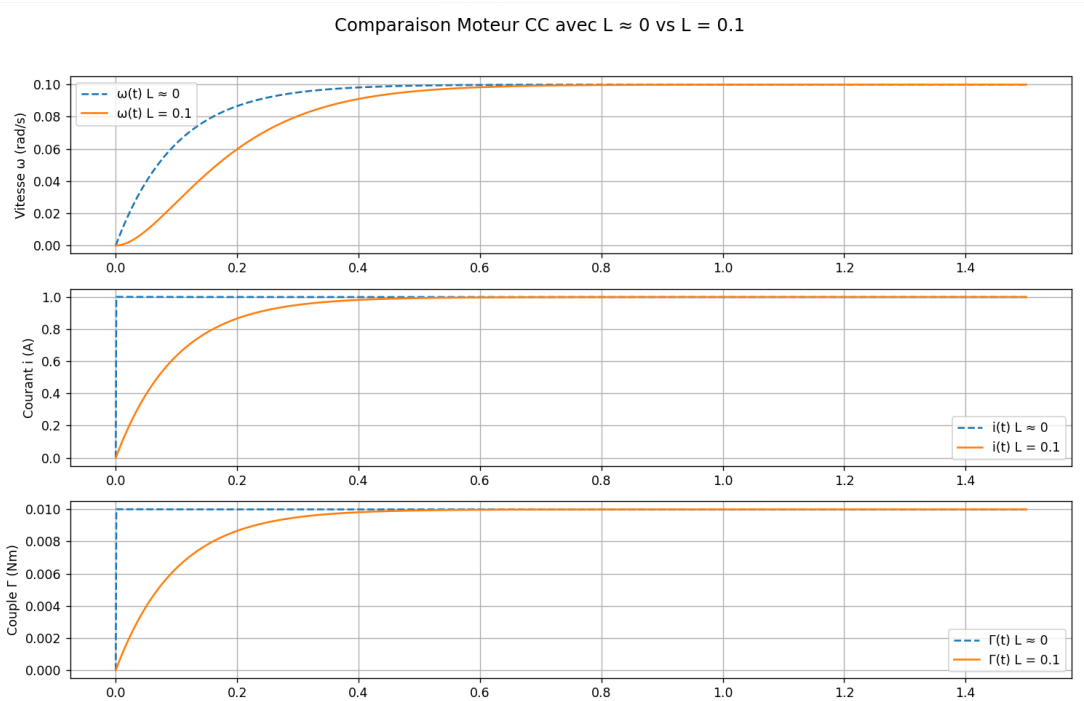


FIGURE 3 – Comparaison des réponses du moteur CC avec et sans inductance ($L \approx 0$ vs $L = 0.1\,\text{H}$)

Analyse des courbes : La figure 3 présente trois courbes comparatives :

- **Vitesse angulaire $\omega(t)$** : sans inductance, le moteur atteint plus rapidement sa vitesse de régime. Avec $L = 0.1\,\text{H}$, l'augmentation est plus progressive, indiquant une dynamique légèrement ralentie.
- **Courant $i(t)$** : le courant présente une discontinuité au démarrage dans le cas $L \approx 0$, car il réagit instantanément à l'échelon. En revanche, pour $L = 0.1$, la montée du courant est amortie, suivant une croissance exponentielle typique d'un circuit RL.
- **Couple moteur $\Gamma(t)$** : puisque le couple est directement proportionnel au courant ($\Gamma = k_c \cdot i$), son évolution suit exactement celle du courant. Là encore, la version sans inductance produit un couple instantané, tandis que le modèle réaliste introduit un retard.

Ces résultats confirment que l'approximation $L \approx 0$ peut être utilisée pour des simulations rapides, notamment en régime permanent. Toutefois, pour modéliser fidèlement les transitoires électriques, notamment au démarrage, il est préférable de conserver l'inductance dans le modèle.

2.3 Enrichissement du modèle

Pour rendre la simulation du moteur à courant continu plus réaliste, le modèle de base a été enrichi en intégrant des éléments extérieurs qui influencent la dynamique du système. Ces ajouts permettent de mieux représenter les conditions de fonctionnement réelles dans des systèmes mécaniques complexes.

Trois types d'enrichissements ont été introduits :

- **Inertie de la charge** : une charge mécanique reliée au moteur ajoute une inertie supplémentaire J_{charge}, qui vient s'ajouter à l'inertie du rotor. La dynamique globale du système devient plus lente, car plus d'énergie est nécessaire pour faire varier la vitesse :

$$J_{\text{total}} = J + J_{\text{charge}}$$

- **Couples extérieurs** : des perturbations ou résistances mécaniques (gravité, frottement sec, couple de freinage) peuvent être modélisées par un couple externe Γ_{ext} opposé au mouvement. L'équation mécanique devient :

$$J\frac{d\Omega}{dt} + f \cdot \Omega = \Gamma_{\text{moteur}} - \Gamma_{\text{ext}}$$

- **Viscosité du milieu** : un frottement visqueux supplémentaire peut représenter l'effet d'un fluide ou d'un matériau résistant. Ce terme est intégré dans le coefficient total de frottement f_{total}.

Ces enrichissements ne changent pas la structure de base du modèle, mais permettent d'évaluer la réponse du moteur dans des contextes plus complexes, tels qu'une centrifugeuse, un robot mobile ou un pendule inversé. Ils renforcent la fidélité de la simulation et ouvrent la voie à des analyses plus approfondies de robustesse ou de contrôle.

2.4 Simulation interactive avec Pygame :

Pour compléter la simulation, une interface interactive a été réalisée à l'aide de la bibliothèque `Pygame`. Elle permet de visualiser en temps réel la rotation du moteur en fonction de la tension appliquée au clavier. L'utilisateur peut augmenter ou diminuer la tension via les touches directionnelles, et observer immédiatement l'évolution de la vitesse de rotation à l'écran.

Ce mini-simulateur, basé sur le fichier `moteurCC_clavier.py`, illustre de manière intuitive la dynamique du moteur et rend l'expérimentation plus accessible et visuelle.

3 Commande PID

3.1 Commande PID : Régulation de la vitesse du moteur CC

Pour améliorer la précision et la stabilité du moteur à courant continu, un régulateur PID (Proportionnel-Intégral-Dérivé) a été mis en œuvre. Ce type de commande agit sur la tension appliquée au moteur en fonction de l'écart entre la vitesse désirée (consigne) et la vitesse réelle mesurée.

Le contrôleur PID génère la tension selon la loi suivante :

$$U_m(t) = K_P \cdot e(t) + K_I \int_0^t e(\tau)\, d\tau + K_D \cdot \frac{de(t)}{dt}$$

où $e(t) = \Omega_{\text{cible}}(t) - \Omega(t)$ est l'erreur de vitesse.

— Le terme proportionnel $K_P \cdot e(t)$ agit instantanément pour corriger l'erreur.
— Le terme intégral $K_I \cdot \int e(t)$ élimine l'erreur statique sur le long terme.
— Le terme dérivé (non utilisé ici) permettrait d'amortir les variations rapides.

Cette commande permet ainsi de réguler dynamiquement la vitesse de rotation du moteur, même en présence de perturbations ou de changements de consigne.

Implémentation du PID : Le PID a été intégrée via une classe `ControlPID_vitesse`, qui agit en boucle fermée. À chaque itération, la vitesse réelle du moteur est mesurée, l'erreur est calculée, puis une tension est générée et appliquée au moteur.

Le contrôleur PID est initialisé avec les gains K_P, K_I, et K_D, et dispose des méthodes suivantes :

— `setTarget(vitesse)` : définit la vitesse cible ;
— `getVoltage()` : calcule la tension selon la loi PID ;
— `simule(step)` : applique la tension au moteur, simule une itération, et enregistre les données ;
— `plot()` : affiche l'évolution de la vitesse et de la tension.

Deux moteurs identiques ont été utilisés pour comparer le comportement en boucle ouverte (BO) et en boucle fermée (BF).

3.2 Comparaison des correcteurs P et PI

Deux simulations ont été menées pour analyser l'influence des gains du contrôleur :
— un correcteur proportionnel seul (P) avec $K_P = 2.0$, $K_I = 0.0$;
— un correcteur proportionnel-intégral (PI) avec $K_P = 1.0$, $K_I = 2.0$.

Analyse des résultats : Correcteur P seul (figure 4) :
— La boucle fermée atteint une vitesse stable autour de $\omega \approx 0.18\,\text{rad/s}$, bien inférieure à la consigne (1 rad/s) : une **erreur statique** importante est présente.
— Le temps de montée est court, avec une réponse rapide.
— La tension sature brièvement puis se stabilise rapidement.
— La boucle ouverte (tension fixe) atteint aussi une vitesse basse mais constante.

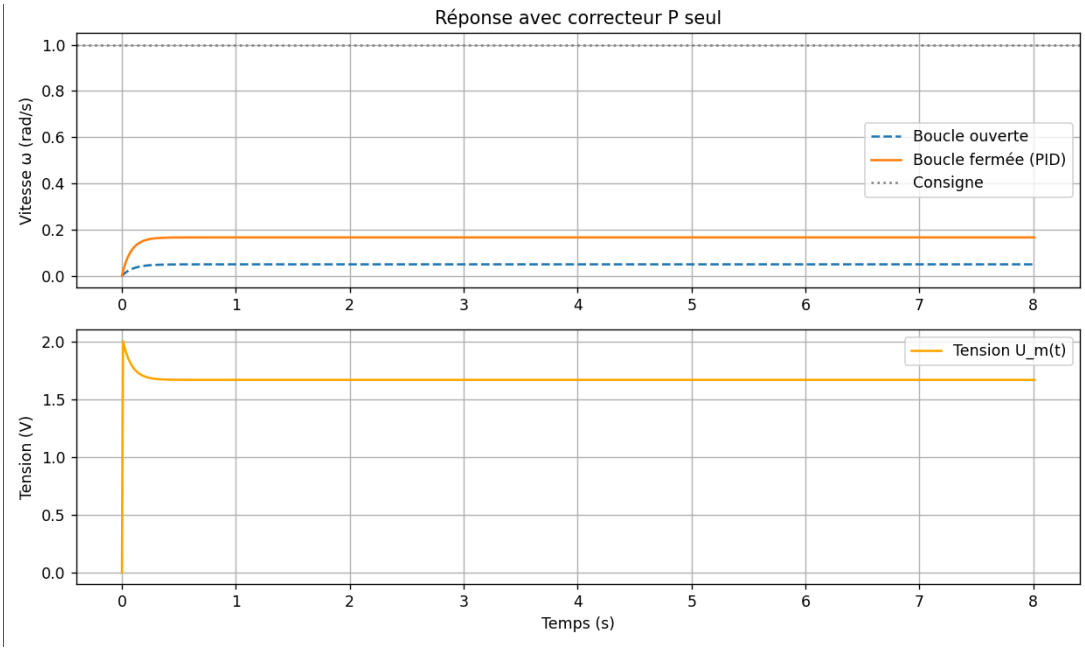


FIGURE 4 – Réponse du moteur avec correcteur P seul

Correcteur PI (figure 5) :
— La vitesse en boucle fermée augmente progressivement vers la consigne sans erreur statique.
— Le terme intégral corrige progressivement l'erreur, mais le temps de réponse est plus long.
— La tension de commande continue d'augmenter avec le temps, illustrant l'accumulation de l'intégrale de l'erreur.
— La boucle ouverte reste stable à une vitesse bien inférieure, illustrant l'intérêt du PID.

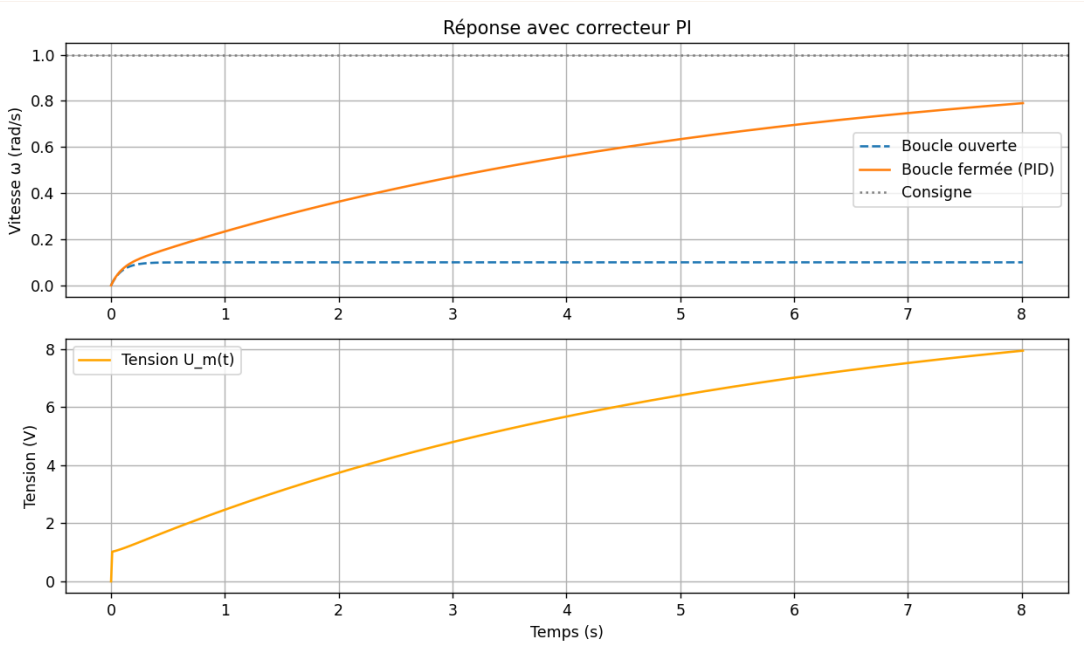


FIGURE 5 – Réponse du moteur avec correcteur PI

Conclusion : Cette comparaison met en évidence les rôles distincts des gains :

— K_P détermine la rapidité de réaction du système, mais ne permet pas d'éliminer l'erreur statique.

— K_I permet de supprimer cette erreur, au prix d'une réponse plus lente et d'un risque de dépassement si mal réglé.

Le contrôleur PI offre ainsi une régulation plus précise et conforme à la consigne, au prix d'un ajustement plus délicat et d'une dynamique plus lente.

4 Commande en position du moteur CC

Après la régulation en vitesse, l'objectif est d'asservir la position angulaire $\theta(t)$ du moteur. L'objectif est d'atteindre une position cible (ex : $\theta = 1.5\,\text{rad}$) en minimisant le temps de réponse, l'erreur statique et les dépassements.

Le contrôleur utilisé ici est de type PD (proportionnel + dérivé), qui calcule la tension d'entrée du moteur en fonction de l'erreur de position et de sa dérivée :

$$U_m(t) = K_P \cdot (\theta_{\text{cible}} - \theta(t)) + K_D \cdot \frac{d}{dt}(\theta_{\text{cible}} - \theta(t))$$

— Le terme proportionnel corrige l'erreur actuelle.

— Le terme dérivé anticipe les variations de l'erreur, permettant de freiner le système et limiter les dépassements.

Une classe dédiée `ControlPID_position` a été implémentée pour piloter le moteur dans ce mode.

4.1 Résultats expérimentaux

Deux tests ont été menés :

— avec un correcteur P seul ($K_P = 5.0, K_D = 0.0$) ;

— avec un correcteur PD ($K_P = 5.0, K_D = 1.0$).

Analyse du P seul :

- La position augmente progressivement mais n'atteint pas la consigne (erreur statique).
- La tension diminue au fil du temps car l'erreur se réduit.
- Réponse lente et sans dépassement, mais peu précise.

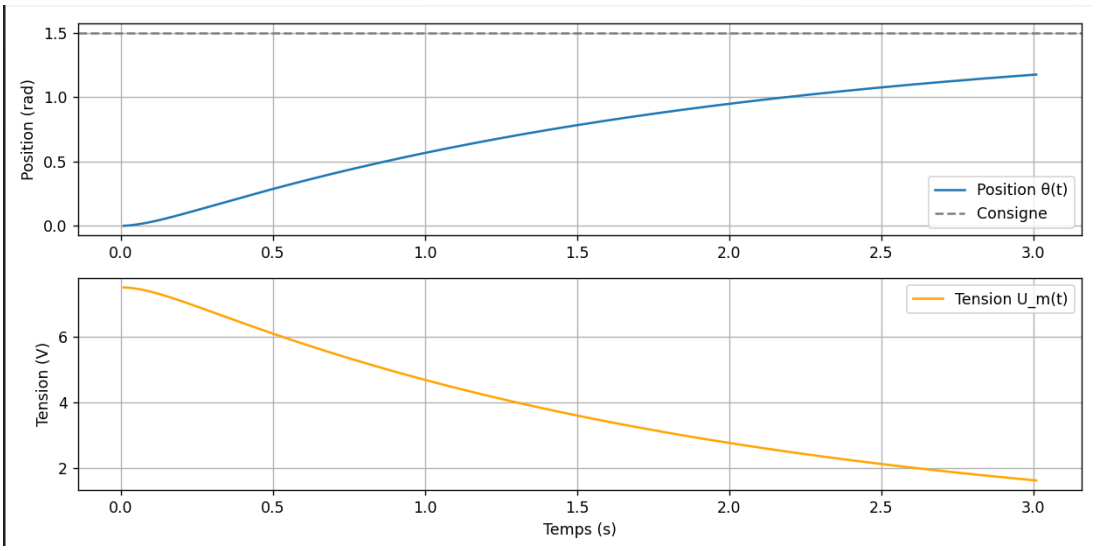


FIGURE 6 – Commande en position avec correcteur P seul

Analyse du PD :

- La réponse initiale est beaucoup plus rapide.
- Un pic de tension très élevé est généré (supérieur à 150 V), ce qui augmente brutalement le couple.
- La position progresse plus rapidement mais reste incomplète à $t = 3\,\mathrm{s}$ (encore une erreur statique).
- Le terme dérivé freine le moteur en fin de trajectoire.

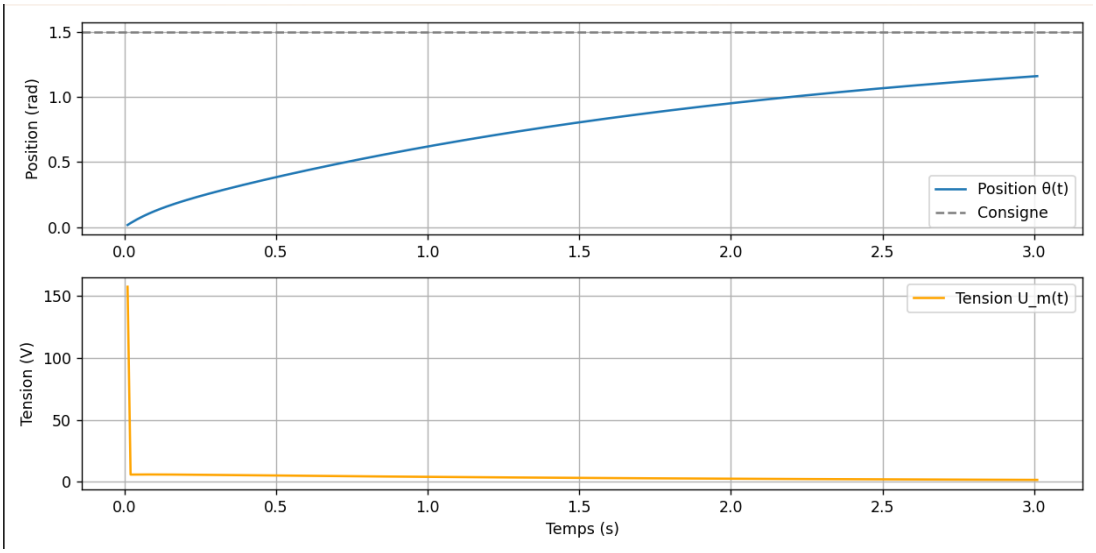


FIGURE 7 – Commande en position avec correcteur PD

4.2 Influence des gains K_P et K_D

— **Augmenter** K_P améliore la rapidité initiale, mais génère plus de dépassement et d'oscillations.
— **Ajouter** K_D permet de stabiliser le système, amortir les oscillations, et réduire le dépassement. Il joue un rôle de frein numérique.
— En revanche, des valeurs trop élevées de K_D peuvent causer des tensions transitoires très fortes, potentiellement dangereuses pour le matériel.

Conclusion : La commande en position avec un correcteur P seul est simple, mais souffre d'erreur statique. L'ajout du terme dérivé améliore significativement la stabilité et la dynamique. Pour un comportement optimal, un réglage fin des gains est indispensable.

Les résultats obtenus montrent que ni le correcteur P, ni le correcteur PD ne permettent à eux seuls d'atteindre la consigne précisément. L'un laisse une erreur statique, l'autre génère un pic de tension trop important sans pour autant éliminer l'erreur finale.

Pour obtenir une commande réellement optimale, il est donc recommandé d'utiliser un correcteur PID complet. Le terme intégral K_I permettrait de supprimer l'erreur statique, tandis que le terme dérivé K_D limiterait les dépassements. Le choix des gains K_P, K_I, K_D reste essentiel pour garantir la stabilité et la performance du système.

5 Turtlebots

5.1 Principe du robot différentiel

Le TurtleBot est un robot mobile à deux roues motrices, dont le déplacement est piloté par les vitesses indépendantes de chaque roue gauche (v_g) et droite (v_d). Ce type de configuration, appelé robot différentiel, est très répandu en robotique mobile.

La cinématique du robot s'exprime par les équations suivantes :

$$v = \frac{v_d + v_g}{2}, \quad \omega = \frac{v_d - v_g}{L}$$

où :
— v est la vitesse linéaire du centre du robot ;
— ω est la vitesse angulaire de rotation autour de son axe vertical ;
— L est la distance entre les deux roues.

Deux classes ont été développées :
— `TurtleDifferential` : version simplifiée, sans moteurs, où les vitesses des roues sont imposées directement.
— `TurtleDifferentialMotor` : version réaliste, intégrant deux moteurs à courant continu contrôlés par PID, un par roue.

Un premier test interactif a été réalisé avec la classe `TurtleDifferential`, permettant de piloter le robot à l'aide du clavier (touches fléchées) en imposant manuellement les vitesses des roues. Cela permet d'explorer le principe du contrôle différentiel de manière intuitive.

5.2 Modèle idéal vs modèle réel

Deux versions du robot ont ensuite été simulées :
— **Robot idéal** : les vitesses v_g et v_d sont imposées directement.
— **Robot réel** : les vitesses sont générées par deux moteurs CC, commandés par des contrôleurs PID en vitesse.

Les trajectoires testées sont composées de trois phases successives :

1. Avance rectiligne : $v_g = v_d = 0.4\,\mathrm{m/s}$

2. Virage à gauche : $v_g = 0.2, v_d = 0.4$
3. Virage à droite : $v_g = 0.4, v_d = 0.2$

Chaque phase dure environ 3 secondes, suivie d'un arrêt.

5.3 Commande PID des moteurs

Chaque roue du robot réel est entraînée par un moteur CC simulé, avec sa dynamique propre (inertie, frottements, couple moteur...). Ces moteurs sont commandés via des régulateurs PID qui ajustent en temps réel la tension d'alimentation pour suivre la consigne de vitesse.

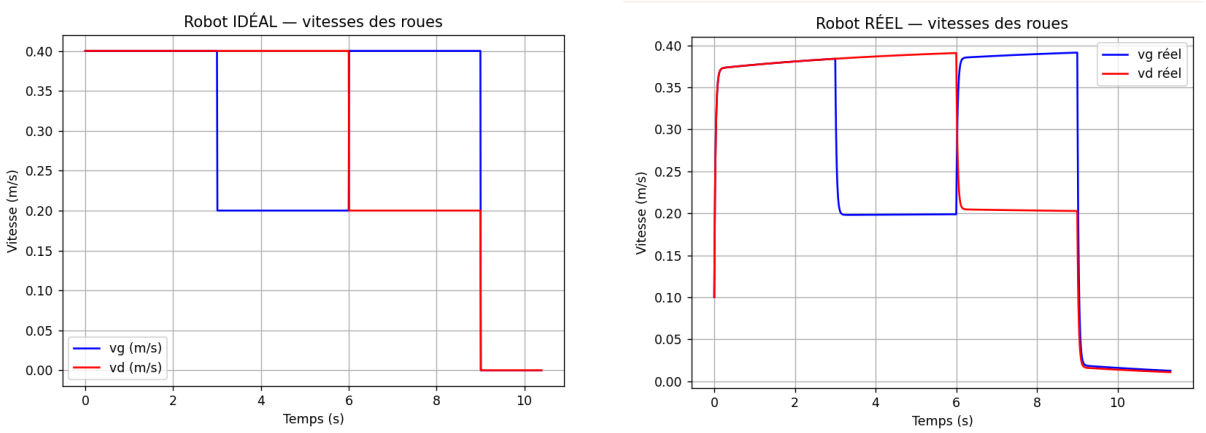
Le contrôleur PID utilisé pour chaque moteur est de type PI, avec :

$$U(t) = K_P \cdot e(t) + K_I \cdot \int e(t)dt$$

Des valeurs typiques utilisées sont $K_P = 8.0$, $K_I = 3.0$, ajustées pour garantir un bon compromis entre rapidité et stabilité.

5.4 Analyse comparative des vitesses des roues

Les figures suivantes comparent les vitesses des roues pour le robot idéal et le robot réel.



(a) Robot IDÉAL — vitesses imposées

(b) Robot RÉEL — vitesses via PID

FIGURE 8 – Comparaison des vitesses des roues pour les modèles idéal (a) et réel (b)

Interprétation :

— Le robot idéal réagit instantanément aux changements de vitesse.
— Le robot réel montre une montée progressive en régime, avec un léger retard au démarrage et à chaque changement de phase.
— Le PID compense efficacement les écarts, mais laisse apparaître une légère dérive ou erreur transitoire selon le réglage.

5.5 Test d'une trajectoire complexe

Pour tester la robustesse du système, une trajectoire plus complexe en forme de « huit » a été simulée, avec les deux robots (idéal et moteur) côte à côte dans un environnement Pygame. La trajectoire cible est recalculée en temps réel, et chaque robot tente de la suivre via les vitesses de ses roues.

— Le robot idéal suit la trajectoire de manière fluide et parfaite.

— Le robot réel parvient à suivre la trajectoire avec une bonne précision, mais montre parfois un léger retard dans les virages ou des arrondis dans les courbes serrées.

Cette simulation permet de visualiser de manière très concrète l'impact de la dynamique moteur et des limitations physiques sur la trajectoire réelle d'un robot.

6 Simulation [moteur centrifugeuse]

6.1 Objectif de la simulation

Le but de cette simulation est d'étudier le comportement dynamique d'une particule soumise à la force centrifuge dans un système tournant. La particule est reliée à l'axe de rotation par un ressort et un amortisseur, et l'ensemble est animé par un moteur à courant continu (CC) en boucle ouverte.

On cherche à tracer la distance radiale d de la particule par rapport à l'axe, en fonction de la vitesse de rotation ω du moteur.

6.2 Modèle physique

Le système se compose de :
— un moteur CC qui fournit une rotation selon la tension appliquée ;
— une particule mobile glissant librement sur un bras radial ;
— un ressort linéaire ramenant la particule vers l'axe ;
— un amortisseur linéaire pour stabiliser les oscillations.

Les forces appliquées à la particule sont :
— Force centrifuge : $F_c = m \cdot \omega^2 \cdot d$
— Force de rappel du ressort : $F_r = -k \cdot d$
— Force d'amortissement : $F_a = -c \cdot \dot{d}$

L'équation différentielle du mouvement s'écrit :

$$m \cdot \ddot{d} = m \cdot \omega^2 \cdot d - k \cdot d - c \cdot \dot{d}$$

En régime permanent ($\ddot{d} \approx 0, \dot{d} \approx 0$), on a :

$$m \cdot \omega^2 \cdot d = k \cdot d \quad \Rightarrow \quad \omega^2 = \frac{k}{m}$$

La distance d'équilibre s'écrit :

$$d = \frac{m \cdot \omega^2}{k - m \cdot \omega^2}$$

Il existe donc une **vitesse critique** $\omega_c = \sqrt{\frac{k}{m}}$ au-delà de laquelle le système devient instable : la force centrifuge dépasse la force de rappel, ce qui provoque une croissance rapide de d vers l'extérieur.

6.3 Implémentation

La simulation est implémentée dans `centrifugeuse_simulation.py`, et utilise les modules :
— `MoteurCC` pour la rotation ;
— `Particule`, `Vector3D`, `Univers` pour la dynamique ;
— Pygame pour la visualisation interactive.

L'utilisateur peut augmenter la tension moteur en temps réel via le clavier (touche A), et observer l'effet sur la vitesse ω et la distance d.

6.4 Résultats et tests

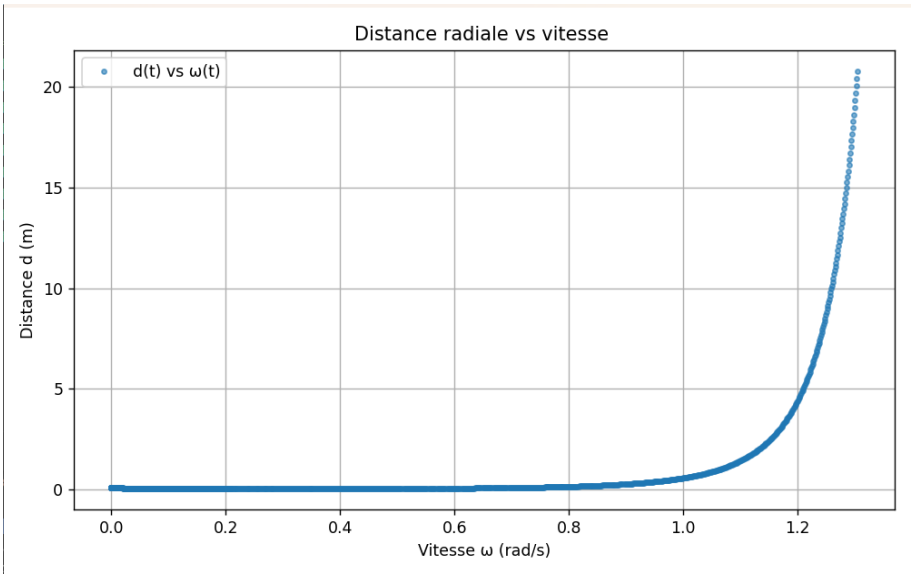


FIGURE 9 – Distance radiale d en fonction de la vitesse angulaire ω

Analyse :

- Pour les faibles vitesses, la distance d reste faible et stable.
- À l'approche de la vitesse critique $\omega_c = \sqrt{k/m}$, la distance croît rapidement : le système devient instable.
- Le ressort contrebalance la force centrifuge jusqu'à une certaine limite.
- L'amortisseur permet d'atteindre plus rapidement le régime stable en amortissant les oscillations.

Conclusion : Après avoir simulé l'ensemble du système composé du moteur CC, du ressort, de l'amortisseur et de la particule, le graphique $d(\omega)$ a été tracé. Il permet de visualiser clairement l'évolution de la distance radiale en fonction de la vitesse de rotation, validant ainsi le comportement physique attendu du système centrifuge.

Cette simulation met en évidence :

- l'effet de la force centrifuge sur une particule en rotation ;
- la limite de stabilité liée au rapport k/m ;
- l'utilité d'un amortissement pour réguler la réponse dynamique ;
- la pertinence d'un simulateur interactif pour étudier ce type de phénomène.

7 Asservissement de l'équilibre d'un pendule inverse

7.1 Validation du comportement du systeme

L'objectif de cette partie est de modéliser et simuler le comportement naturel d'un pendule inversé, dans deux situations :

- lorsque la base est fixée au sol ;
- lorsque la base est libre de se déplacer horizontalement.

Ces simulations permettent de valider le comportement dynamique attendu du système avant d'implémenter un asservissement.

7.2 Simulation 1 : base fixe

Le pendule est initialisé dans une position légèrement décalée de la verticale avec une vitesse angulaire nulle. On observe le mouvement pendulaire non amorti.

Le modèle utilisé est :

$$\ddot{\theta} = -\frac{g}{l} \cdot \sin(\theta)$$

Une solution analytique linéarisée autour de $\theta = \pi$ est également calculée pour valider la simulation :

$$\theta(t) \approx \pi - \theta_0 \cdot \cos\left(\sqrt{\frac{g}{l}} \cdot t\right)$$

7.3 Simulation 2 : base mobile libre

Dans cette configuration, la base horizontale peut se déplacer librement sur un axe. La dynamique devient couplée :

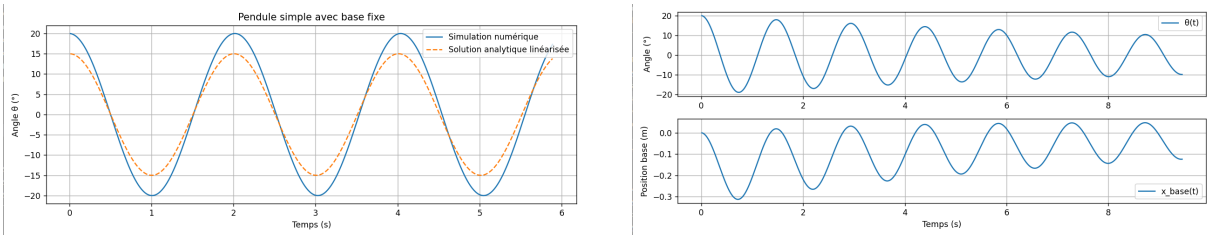
$$\ddot{\theta} = -\frac{g}{l} \cdot \sin(\theta) + \frac{\ddot{x}}{l} \cdot \cos(\theta)$$

où $\ddot{x}$ est l'accélération de la base, elle-même induite par la force de réaction du pendule :

$$\ddot{x} = \omega^2 \cdot \sin(\theta) - g \cdot \cos(\theta) \cdot \sin(\theta)$$

Un léger amortissement visqueux est ajouté sur la base pour éviter une dérive infinie.

7.4 Résultats comparés



(a) Pendule inversé avec base fixe

(b) Pendule inversé avec base mobile libre

FIGURE 10 – Comparaison du comportement du pendule inversé avec base fixe (a) et mobile (b)

Analyse :

- Dans le cas **base fixe**, le pendule effectue des oscillations régulières et conserve son énergie. La courbe simulée est proche de la solution théorique.
- Avec une **base mobile**, les oscillations du pendule génèrent des déplacements latéraux couplés de la base. Celle-ci oscille naturellement en réponse aux forces horizontales.
- Le système reste non amorti mais conserve une dynamique cohérente et réaliste.

7.5 Validation du modèle

Les résultats obtenus confirment que :

- le modèle de pendule inversé simule correctement les effets gravitationnels et inertiels ;
- la prise en compte de la base mobile introduit une dynamique couplée réaliste ;
- la réponse numérique est conforme aux équations analytiques dans le cas linéarisé.

Cette étape permet de valider le fonctionnement du module physique avant de passer à une commande active (PID).

7.6 Commande automatique du pendule inversé à base mobile

Après avoir validé le comportement libre du pendule, cette partie vise à maintenir automatiquement le pendule inversé en position verticale grâce à un système de commande asservie.

L'objectif est de synthétiser un correcteur PID permettant de stabiliser un pendule inversé sur une base mobile. La base est actionnée par un moteur à courant continu et peut se déplacer uniquement selon l'axe horizontal.

Le contrôleur calcule une tension U_m appliquée au moteur CC, dans le but de produire un déplacement adapté de la base, afin de compenser les déséquilibres du pendule et le maintenir proche de l'axe vertical ($\theta = \pi$).

Structure de la commande :

- La variable mesurée est l'angle θ du pendule par rapport à la verticale.
- Le PID agit sur l'erreur $\theta - \pi$ pour générer une tension U_m.
- Le moteur CC convertit cette tension en mouvement horizontal de la base ($x_b(t)$).
- Une perturbation (pichenette) peut être injectée par l'utilisateur via les touches ← / →, ce qui agit directement sur la vitesse angulaire du pendule.

Validation : Les résultats montrent que le système est capable de :

- ramener le pendule à la verticale après une perturbation ;
- compenser dynamiquement les déséquilibres en déplaçant la base ;
- saturer naturellement la tension du moteur pour rester dans les limites physiques.

Réglage du PID et observations : Afin de déterminer des valeurs cohérentes pour les paramètres du correcteur PID (K_p, K_i, K_d), plusieurs essais ont été réalisés en ajustant les coefficients de manière empirique. L'objectif était d'obtenir une réponse stable, sans oscillations excessives ni dérive. Le réglage retenu ($K_p = 250$, $K_d = 20$, $K_i = 2$) permet d'illustrer correctement le principe de stabilisation d'un pendule inversé

Le système permet de visualiser clairement la logique du PID et le rôle de chaque composante dans la régulation.

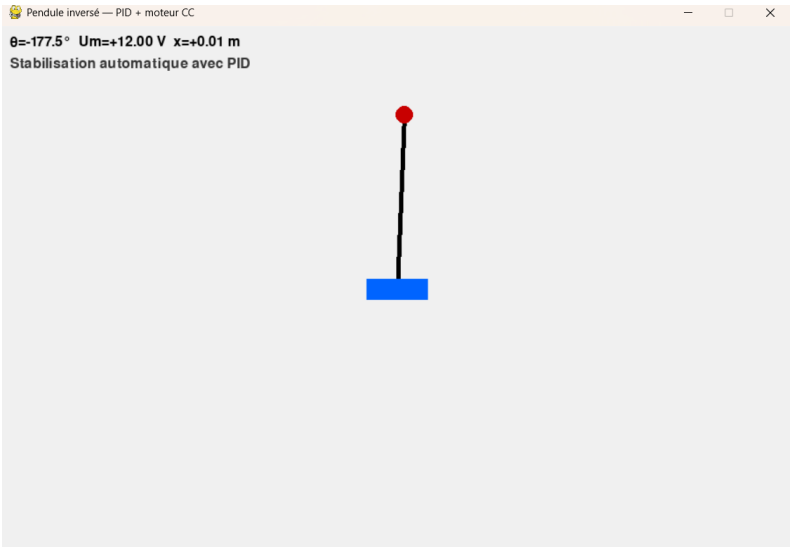


FIGURE 11 – Stabilisation du pendule inversé par un PID agissant sur un moteur CC

8 Simulation des barres 2D

Dans un premier temps, j'ai développé une classe nommée `Barre2D`, représentant un solide rigide en deux dimensions. Cette classe modélise les caractéristiques physiques d'une barre (position, orientation, vitesse, masse, inertie), et permet d'y appliquer des torseurs (forces et moments). Elle inclut également une méthode de simulation basée sur les équations du mouvement, avec un amortissement simplifié pour stabiliser les trajectoires.

Une fois cette classe implémentée et testée, je l'ai intégrée dans une structure d'univers dynamique (`Univers.py`) permettant de simuler plusieurs objets (barres, particules, forces, liaisons, etc.) en interaction. Le moteur de simulation inclut la gestion des liaisons mécaniques (pivot, prismatique, etc.) et permet d'interagir en temps réel avec le clavier pour appliquer des forces et moments à des objets contrôlables.

En parallèle, j'ai implémenté plusieurs modules complémentaires :

— `forces.py` pour gérer les générateurs de forces comme la gravité ;
— `liaison.py` et `liaison_mixte.py` pour simuler des connexions mécaniques entre les objets ;
— `torseur.py` pour représenter les efforts mécaniques appliqués ;
— `vector3D.py` pour manipuler des vecteurs 3D dans l'espace.

Malgré cette base fonctionnelle, je n'ai pas pu aboutir à une simulation complète

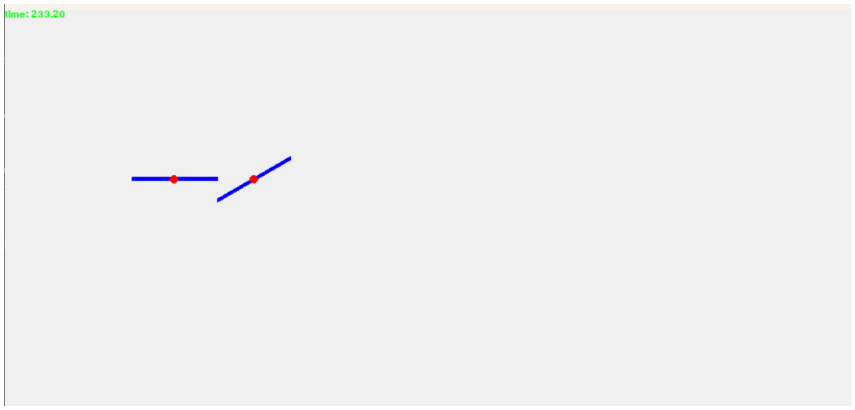


FIGURE 12 – Affichage graphique d'un système avec barres, liaisons et forces simulées en temps réel

9 Conclusion

Ce projet de simulation physique m'a permis d'explorer en profondeur les principes fondamentaux de la modélisation dynamique et du contrôle de systèmes électromécaniques, à travers des cas concrets et variés.

J'ai pu concevoir et valider numériquement un modèle de moteur à courant continu, l'intégrer dans des architectures complexes comme la centrifugeuse ou le pendule inversé, et expérimenter différents types de commandes (P, PI, PID) pour en analyser les performances.

L'utilisation de Python et Pygame m'a offert un cadre flexible et interactif pour tester mes implémentations. Par ailleurs, l'approche orientée objet et la structuration du code en modules m'ont permis d'acquérir des compétences solides en développement logiciel pour la simulation.

Malgré quelques limitations de temps qui m'ont empêché de traiter toutes les parties avancées du projet (comme la simulation complète des barres 2D ou du robot 2R), je suis satisfait du travail réalisé, de sa qualité, et de sa pertinence vis-à-vis des objectifs du module.

Enfin, cette expérience confirme mon intérêt fort pour la robotique et le contrôle dynamique, domaines que je souhaite approfondir dans la suite de mon parcours académique et professionnel.

Annexe : Code source sur GitHub

L'ensemble du code source développé dans le cadre de ce projet est disponible sur le dépôt GitHub suivant :

`https://github.com/amirahaddad06/Projet_Simulation_physique.git`

10 Auto-évaluation

Note proposée : 4.5 / 5

Je m'attribue la note de 4.5 sur 5, car j'ai été très sérieux et impliqué tout au long de ce projet. J'ai été présent de manière régulière à toutes les séances de TP, et j'ai toujours avancé de façon autonome sur le travail demandé.

À chaque nouvelle tâche, je faisais de mon mieux pour la réaliser rapidement et vous présenter mes résultats directement en séance, ce qui m'a permis d'avoir des retours et validations en continu.

J'ai trouvé ce module particulièrement motivant : simuler des systèmes dynamiques et robotiques m'a beaucoup intéressé, car cela m'a aidé à mieux comprendre leur fonctionnement. Je suis convaincu que ces compétences me seront utiles pour la suite de mes études et, plus tard, dans ma vie professionnelle, notamment dans le domaine de la robotique.

Dans la réalisation de ce projet, j'ai vraiment souhaité produire un travail complet et de qualité. Même si je n'ai pas pu traiter toutes les simulations et problématiques posées, j'ai mis l'accent sur la clarté et le professionnalisme du rapport, en utilisant notamment LaTeX et GitHub pour assurer une bonne structuration et une présentation propre.

Je tiens également à m'excuser pour le léger retard dans la remise de ce travail. J'ai commencé mon stage immédiatement après la dernière semaine de cours, ce qui a entraîné une charge importante dès le début et a légèrement repoussé la finalisation du rendu. Malgré cela, j'ai travaillé avec rigueur et sérieux jusqu'au bout.