Rapport Simulation

Simulation Physique et Commande

2024/2025

par

Sayf Chafik

SCIENCES
SORBONNE
UNIVERSITÉ

Table des matières

1

Introduction

Introduction

Ce projet a pour objectif la modélisation, la simulation et le contrôle de systèmes à l'aide de l'environnement Python. Il s'inscrit dans le cadre de l'UE de simulation physique, et couvre plusieurs problématiques allant de la commande de moteurs à courant continu à la simulation de systèmes dynamiques complexes.

Le travail est structuré autour de quatre thématiques principales :

— **Simulation d'un moteur à courant continu (CC)** : modélisation électrique et mécanique d'un moteur, étude en boucle ouverte et fermée (contrôleur P et PI), comparaison avec une solution analytique.

— **Simulation d'une centrifugeuse** : système à particule libre, ressort et moteur CC, analyse du régime permanent et validation des lois centrifuges.

— **Simulation d'une barre rigide en 2D** : pendule physique soumis à la gravité, modélisation du moment d'inertie, couplage avec ressort torsionnel, validation des modes propres.

— **Commande d'un robot TurtleBot** : modélisation cinématique et dynamique d'un robot , comparaison entre commande directe en vitesse et commande via tension moteur.

Chaque section présente la modélisation physique associée, les choix d'implémentation retenus, le code utilisé ainsi que les résultats obtenus (figures, courbes, trajectoires). L'accent est mis sur la validation des modèles et la capacité du simulateur à reproduire les comportements attendus.

L'ensemble du projet a été réalisé en autonomie complète afin d'explorer de manière approfondie les aspects numériques et algorithmiques de la simulation physique. Ce rapport détaille les résultats expérimentaux obtenus et propose une analyse critique des performances et des limites de chaque approche.

2

Simulation d'un moteur à courant continu (CC)

2.1 Simulation et commande d'un moteur à courant continu (CC)

2.1.1 1. Modélisation physique et solution analytique

Le moteur à courant continu est modélisé à partir des équations électromécaniques classiques :

- Équation électrique : $U_m(t) = E(t) + Ri(t) + L\dfrac{di(t)}{dt}$,
- Force contre-électromotrice : $E(t) = k_e\Omega(t)$,
- Couple moteur : $\Gamma(t) = k_c i(t)$,
- Équation mécanique : $J\dfrac{d\Omega(t)}{dt} + f\Omega(t) = \Gamma(t)$.

Simplification $L \approx 0$: On néglige l'inductance L, ce qui permet de simplifier le modèle et d'obtenir une solution analytique.

À partir de l'équation électrique simplifiée :

$$U_m(t) = k_e\Omega(t) + Ri(t) \Rightarrow i(t) = \frac{U_m(t) - k_e\Omega(t)}{R}$$

Puis, en injectant cette expression dans l'équation mécanique :

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

Il s'agit d'une équation différentielle du premier ordre, dont la solution pour un échelon $U_m(t) = U_0$ et $\Omega(0) = 0$ est :

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

Cette expression est utilisée pour valider la simulation numérique.

2.1.2 2. Simulation numérique du moteur

Implémentation : classe `MoteurCC`

Une classe `MoteurCC` a été développée en Python pour intégrer les équations du moteur à chaque pas de temps. Le modèle discret met à jour les grandeurs dynamiques suivantes : courant $i(t)$, vitesse $\Omega(t)$, position angulaire $\theta(t)$.

Méthodes principales :

- `setVoltage(V)` : applique une tension $U_m(t)$ au moteur ;
- `simulation(step)` : intègre l'évolution de la vitesse et du courant ;
- `plot(temps)` : trace la réponse du moteur.

Résultat : réponse indicielle en boucle ouverte

Le moteur est soumis à un échelon de tension de $U_0 = 0{,}1$ V. Le graphe ci-dessous montre la comparaison entre la simulation numérique et la solution analytique :

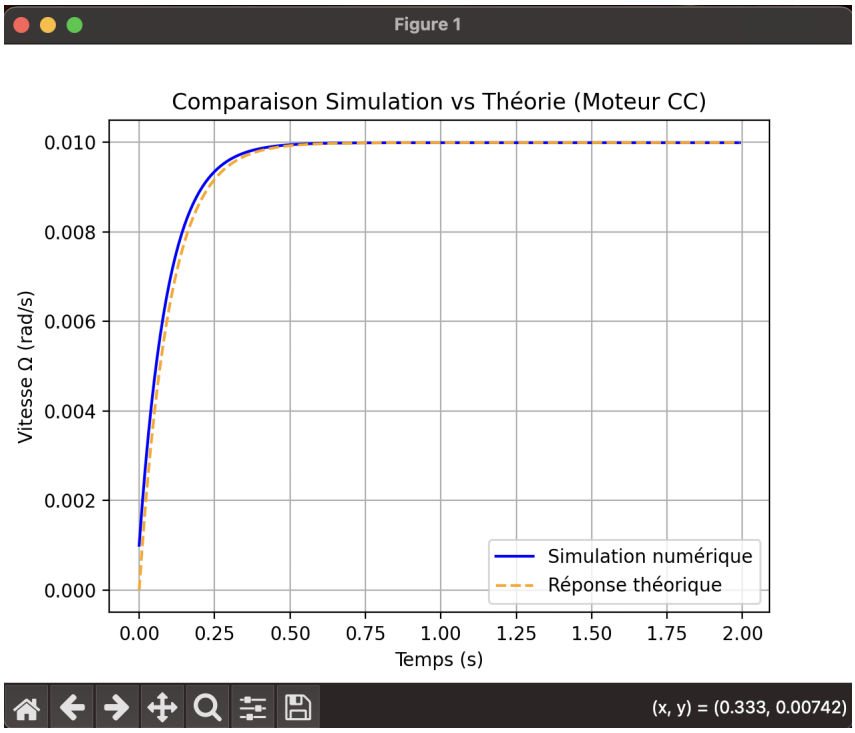

FIGURE 2.1 – Comparaison simulation numérique vs solution analytique pour une entrée échelon.

On observe une excellente concordance entre le modèle simulé et la solution théorique. Cette cohérence valide le modèle numérique et la pertinence des hypothèses simplificatrices.

Influence de la simplification $L = 0$: La négligence de L permet d'obtenir un modèle plus simple à intégrer numériquement. Cette hypothèse reste valable dans notre cas car l'inductance est très faible (1 mH).

Extensions possibles : Le modèle peut facilement intégrer :
- l'ajout d'une charge inertielle externe ;
- des couples de frottement non linéaires ou un couple résistant ;
- des perturbations sur la tension ou la vitesse.

2.1.3 3. Commande du moteur en boucle fermée

L'objectif est de réguler la vitesse du moteur autour d'une consigne Ω_{cible}, en ajustant dynamiquement la tension via un correcteur. Deux types de régulateurs sont comparés :

Contrôleur proportionnel (P)

Ce contrôleur applique une tension proportionnelle à l'erreur :

$$U_m(t) = K_p \cdot (\Omega_{\text{cible}} - \Omega(t))$$

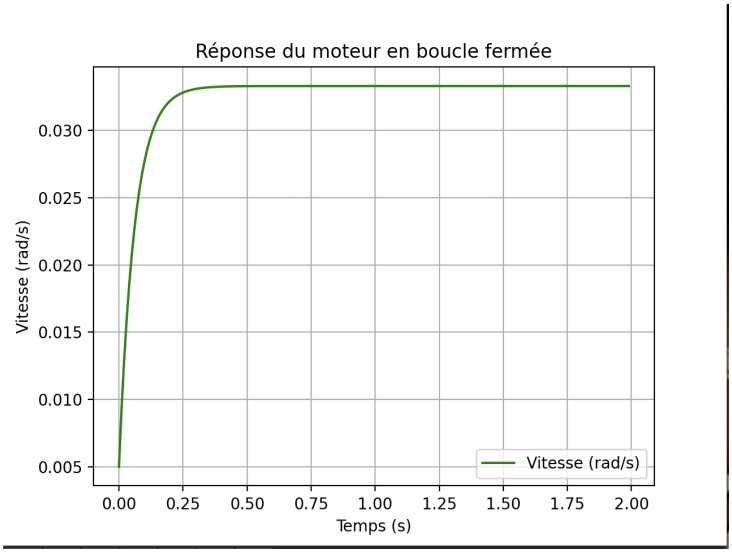

FIGURE 2.2 – Réponse du moteur avec un contrôleur P ($K_p = 5$)

Le système est plus rapide, mais atteint une vitesse légèrement inférieure à la consigne, illustrant la présence d'une erreur statique.

Contrôleur proportionnel-intégrateur (PI)

Le contrôleur PI ajoute une composante intégrale pour éliminer l'erreur statique :

$$U_m(t) = K_p \cdot e(t) + K_i \cdot \int e(t)\, dt$$

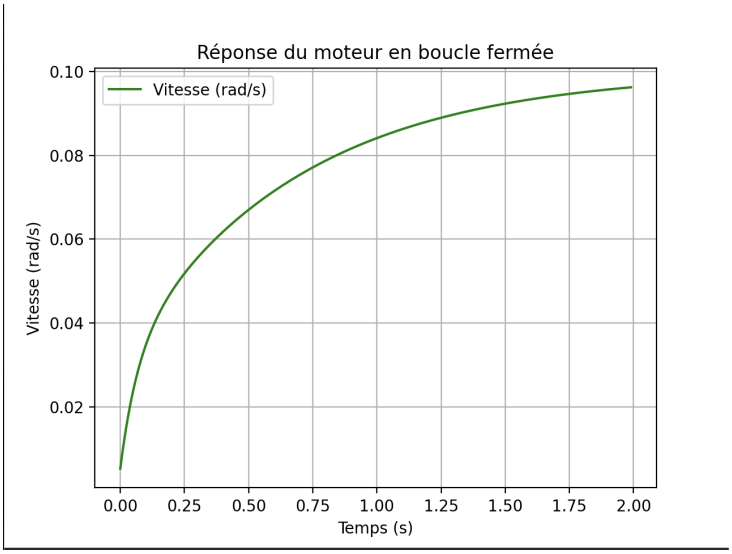

FIGURE 2.3 – Réponse du moteur avec un contrôleur PI ($K_p = 5$, $K_i = 20$)

La réponse atteint très précisément la consigne avec un temps de réponse correct.

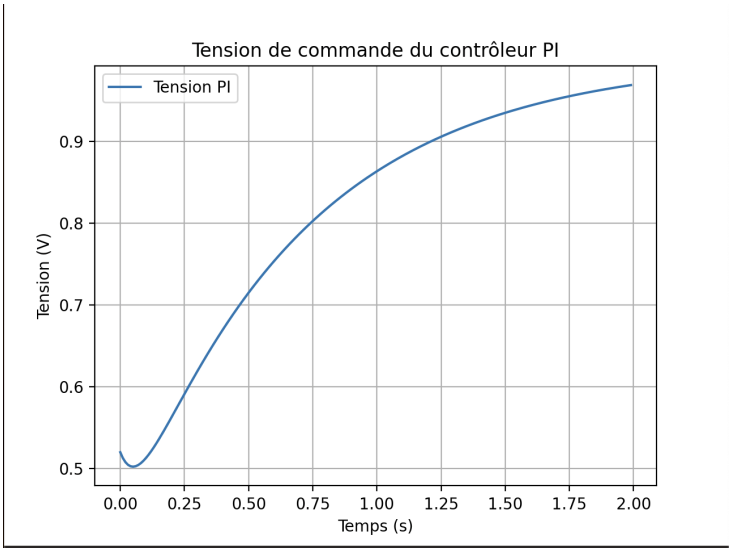

FIGURE 2.4 – Tension de commande générée par le contrôleur PI

Cette figure montre la tension nécessaire pour appliquer la commande. On note une montée rapide initiale suivie d'une stabilisation.

Choix des gains K_p et K_i

Les gains ont été déterminés empiriquement :
- $K_p = 5$: bon compromis entre temps de réponse et stabilité ;
- $K_i = 20$: élimine efficacement l'erreur statique sans trop d'oscillations.

Méthodologie : Les valeurs ont été ajustées en fonction de :
- la vitesse d'établissement ;
- l'amplitude maximale de la tension ;
- la précision atteinte en régime permanent.

Comparaison des performances

— **Boucle ouverte** : réponse lente, sans régulation.
— **Contrôle P** : amélioration du temps de réponse mais erreur statique.
— **Contrôle PI** : excellente précision, mais commande plus exigeante.

2.1.4 4. Choix techniques et architecture du code

Architecture orientée objet :
— `MoteurCC` : modélise les équations du moteur ;
— `ControlPID_vitesse` : implémente les stratégies de commande.

Méthodologie numérique :
— intégration par Euler explicite ;
— pas de temps $\Delta t = 0.01$;
— visualisation avec `matplotlib`.

2.1.5 Conclusion

Cette étude a permis de :
— dériver et exploiter une solution analytique exacte ;
— construire un simulateur robuste pour moteur CC ;
— comparer les performances de deux types de correcteurs.

Le contrôleur PI apporte une régulation précise et rapide. Le simulateur est modulaire et adaptable à des extensions plus complexes (PID, perturbations, saturation...).

3

Simulation - moteur centrifugeuse

3.1 Simulation d'une centrifugeuse à moteur à courant continu

3.1.1 Objectif

L'objectif de cette simulation est de modéliser le comportement d'une centrifugeuse constituée de :

— Un **moteur à courant continu (CC)** qui entraîne une rotation,
— Une **particule libre** de masse m,
— Reliée au centre de rotation par un **ressort** de raideur k et un **amortisseur** de coefficient c.

Le but est de tracer la distance d'élongation du ressort (distance entre la particule et l'axe de rotation) en fonction de la vitesse angulaire ω du moteur, en régime permanent.

3.1.2 Modélisation physique

Forces en jeu

Trois forces principales interviennent :

— **Force centrifuge** ressentie par la particule :

$$F_c = m \cdot \omega^2 \cdot r$$

— **Force de rappel du ressort** :

$$F_r = k \cdot (r - l_0)$$

— À l'équilibre :

$$k(r - l_0) = m\omega^2 r \quad \Rightarrow \quad r = \frac{kl_0}{k - m\omega^2}$$

Cette relation permet d'obtenir la **position théorique** de la particule à l'équilibre en fonction de ω.

Choix d'implémentation

— On utilise une **vitesse tangentielle initiale** pour initier la rotation : $v = r \cdot \omega$.
— La particule est **libre** d'évoluer sous l'effet des forces (ressort + inertie).

— Le moteur CC est simulé dynamiquement avec ses équations (tension, courant, vitesse).
— Une estimation initiale $\omega = \dfrac{U}{R}$ permet de définir la vitesse tangentielle au départ.
— Une classe intermédiaire `MoteurSimulable` encapsule le moteur CC pour l'intégrer dynamiquement dans l'univers.
— Pour chaque tension, une nouvelle instance d'univers est créée, garantissant l'indépendance des simulations.

3.1.3 Architecture du code

Classes principales

— `Univers` : gestion du temps, des entités, des forces, et de la simulation.
— `Particule` : objet dynamique, avec position, vitesse, masse.
— `SpringDamper` : modélise la liaison ressort + amortisseur.
— `MoteurCC` : moteur à courant continu (modèle simplifié), calcule la vitesse en fonction de la tension.

Principe de simulation

— Pour chaque tension appliquée au moteur, on simule le système pendant quelques secondes (typiquement 5s).
— La vitesse angulaire ω est estimée en fin de simulation à partir de :

$$\omega = \frac{\|v\|}{\|r\|}$$

— On mesure la **distance** $d = \|r\|$ atteinte à l'équilibre.
— La distance théorique est obtenue via :

$$r = \frac{kl_0}{k - m\omega^2} \quad \text{avec} \quad \omega_{\text{lim}} = \sqrt{\frac{k}{m}}$$

— Une exception est prévue dans le code pour éviter les divisions par zéro lorsque $\omega \to \omega_{\text{lim}}$.

3.1.4 Résultats et analyse

Observation graphique

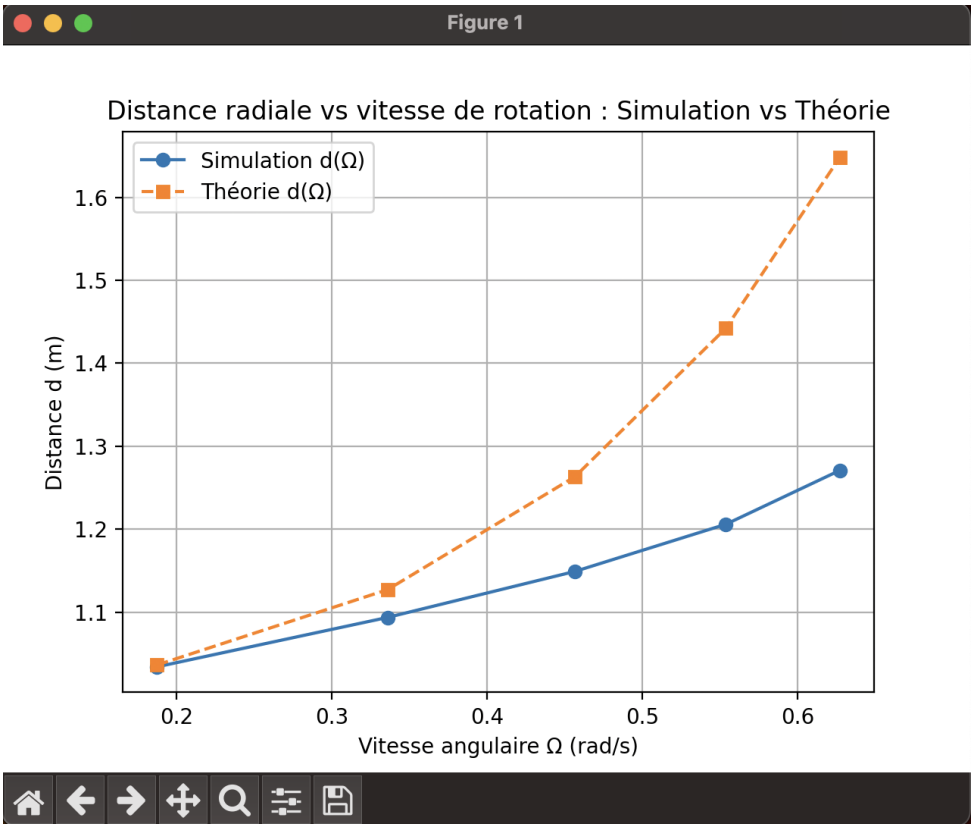

FIGURE 3.1 – Distance radiale d en fonction de la vitesse angulaire ω : Simulation vs Théorie

— La courbe bleue correspond aux résultats simulés.
— La courbe orange pointillée correspond à la distance théorique.
— La tendance est croissante et non linéaire, comme attendu.

Analyse physique

— L'accord simulation/théorie est globalement bon.
— Les écarts sont dus à l'effet dissipatif de l'amortisseur, à l'inertie initiale, et à la durée finie de simulation.
— Le calcul de ω se fait numériquement à partir de la vitesse tangentielle en fin de simulation, ce qui reflète le comportement réel du moteur et de la particule.

Comment marche mon code

— Pour chaque tension de la liste, une simulation est lancée dans un nouvel univers.
— Une particule mobile, un centre fixe, un moteur CC simulé, et un lien ressort/amortisseur sont instanciés.
— La classe `MoteurSimulable` met à jour la vitesse du moteur à chaque pas de temps.
— En fin de simulation, on extrait la position, la vitesse tangentielle et on calcule ω.
— Les résultats sont stockés dans trois listes : simulation, théorie, vitesse.

3.1.5 Conclusion

Cette simulation montre comment une particule soumise à une force centrifuge atteint une position d'équilibre cohérente avec la théorie.

Le moteur CC n'est pas imposé en vitesse, mais piloté en tension, ce qui donne un comportement réaliste et contrôlable.

Le code Python est modulaire, stable et réutilisable. Il permet d'introduire facilement des extensions comme :

— La commande en boucle fermée de ω (PI),
— L'introduction de perturbations,
— L'ajout d'un modèle non linéaire ou d'un capteur.

4

Commande en position du moteur CC

4.1 Commande en position du moteur CC à l'aide d'un correcteur PID (P+D)

4.1.1 Objectif

L'objectif est d'implémenter une commande en position basée sur un contrôleur PID (ici simplifié en P+D) pour un moteur à courant continu (CC), et d'en analyser les effets sur la réponse du système en fonction des gains proportionnel (K_p) et dérivé (K_d).

4.1.2 Modèle utilisé

Le moteur est modélisé numériquement selon les équations classiques du moteur CC. Comme proposé dans l'énoncé, on néglige l'inductance de l'induit ($L \approx 0$), ce qui simplifie l'équation électrique à :

$$U_m(t) = E(t) + R \cdot i(t)$$

Cette approximation est valide dans notre cas car $L = 0{,}001\,H \ll R = 1\,\Omega$, ce qui permet de se concentrer sur la dynamique mécanique du système.

4.1.3 Code et simulation

Implémentation du contrôleur L'algorithme PID simplifié est encapsulé dans une classe `ControlPID_position`. Il effectue :

- le calcul de l'erreur et de sa dérivée,
- l'application de la tension au moteur,
- l'enregistrement de la position et de la tension pour analyse.

La commande utilisée est :

$$V(t) = K_p \cdot e(t) + K_d \cdot \frac{de(t)}{dt}$$

```
V = self.Kp * error + self.Kd * d_error
```

Paramètres de simulation

- $K_p = 15$
- $K_d = 2$
- Consigne de position : $\theta_{\text{cible}} = 1.0\,\text{rad}$

— Pas de simulation : step = 0.01 s
— Durée totale : $t_{\max} = 2.0$ s

4.1.4 Résultats

Réponse en position du moteur

— La position augmente rapidement vers la consigne.
— La courbe est monotone croissante, sans oscillations ni dépassement.
— La stabilisation est atteinte en moins de 2 secondes.

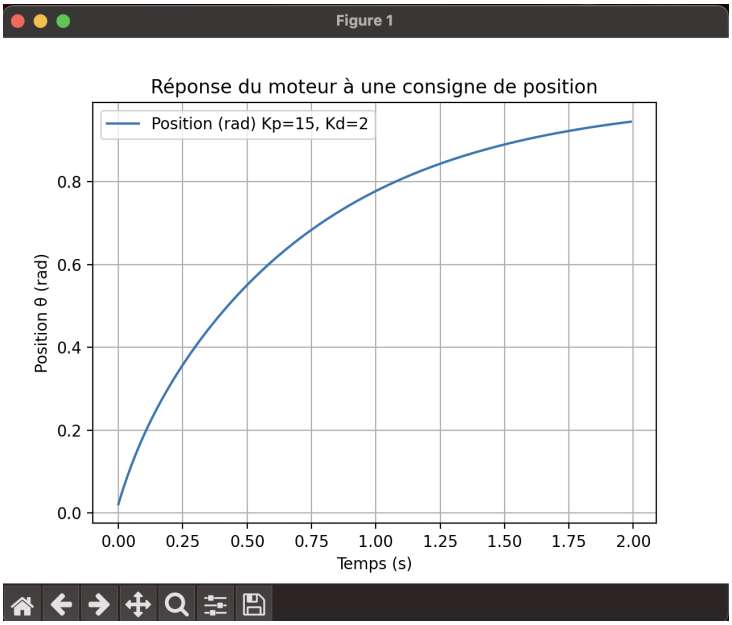

FIGURE 4.1 – Réponse du moteur à une consigne de position ($K_p = 15, K_d = 2$)

Tension appliquée (commande)

— Un pic de tension élevé est observé au démarrage (environ 220 V), dû à une forte erreur initiale.
— La tension décroît ensuite rapidement de manière continue à mesure que le système se stabilise.

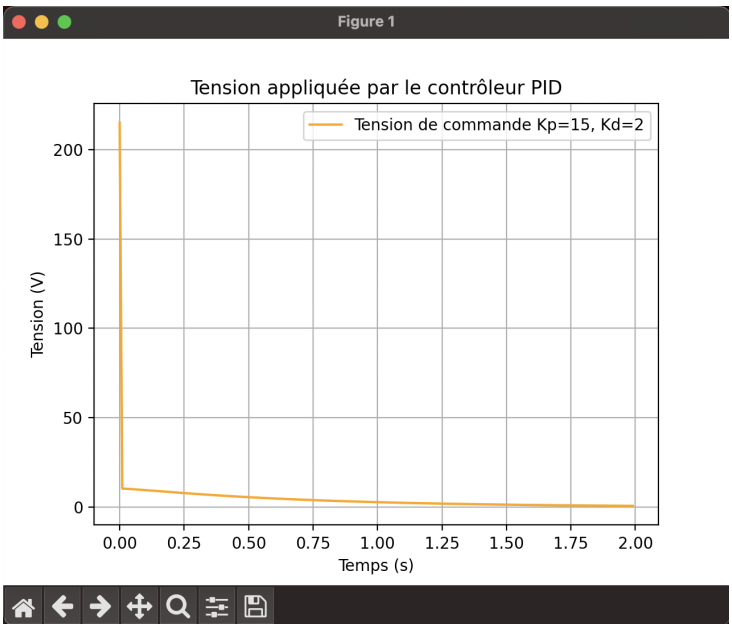

FIGURE 4.2 – Tension appliquée par le contrôleur PID

4.1.5 Comparaison de différents gains

Trois jeux de gains ont été testés pour observer leur influence sur la dynamique :

— **Cas 1 :** $K_p = 5, K_d = 0$: réponse lente avec erreur résiduelle.

— **Cas 2 :** $K_p = 15, K_d = 2$: réponse rapide et stable, bon compromis.

— **Cas 3 :** $K_p = 20, K_d = 5$: réponse très rapide mais avec risque de dépassement ou instabilité.

Les courbes associées peuvent être ajoutées en annexe pour appuyer cette analyse.

4.1.6 Analyse et conclusion

L'utilisation d'un correcteur P+D permet d'obtenir :

— Une bonne vitesse de convergence grâce à K_p,

— Une stabilisation sans oscillation excessive grâce à K_d,

— Un pilotage efficace du moteur pour atteindre une consigne en position.

Ce type de régulation est adapté à des systèmes sans perturbation prolongée. Pour aller plus loin, on pourrait envisager l'ajout d'un terme intégral K_i pour éliminer toute erreur statique, en particulier si des couples perturbateurs externes sont présents.

5

Simulation des barres 2D

5.1 Modélisation d'une barre rigide en 2D

5.1.1 Objectifs et cadre physique

Cette partie du projet vise à simuler le comportement dynamique d'une barre rigide fixée à une extrémité, soumise à la gravité et à d'éventuelles interactions mécaniques (couples extérieurs, couplage torsionnel). Elle constitue un modèle simplifié mais fidèle d'un pendule physique, étudié dans le cadre de la dynamique du solide en rotation autour d'un point fixe.

5.1.2 Modèle physique utilisé

La barre est modélisée comme un solide rigide de masse m et de longueur L, animé d'un mouvement de rotation autour d'un pivot situé à l'une de ses extrémités. Son moment d'inertie, relativement à cet axe, est donné par la formule classique :

$$I = \frac{1}{3}mL^2$$

Le couple exercé par la gravité est modélisé rigoureusement à partir du vecteur bras de levier $\vec{r}$ (du pivot au centre de masse) et du vecteur poids $\vec{F}_g = m\vec{g}$:

$$\vec{\Gamma}_{\text{gravité}} = \vec{r} \times \vec{F}_g \quad \Rightarrow \quad \Gamma_z = (\vec{r} \times \vec{F}_g)_z$$

La dynamique angulaire de la barre est ensuite décrite par les équations :

$$I \cdot \alpha = \Gamma_{\text{total}} \quad ; \quad \frac{d\theta}{dt} = \omega \quad ; \quad \frac{d\omega}{dt} = \alpha$$

où θ est l'angle de la barre, ω sa vitesse angulaire, et α son accélération angulaire.

5.1.3 Implémentation numérique : `Barre2D`

L'objet `Barre2D` contient les attributs physiques et dynamiques nécessaires à la simulation. L'intégration dans le temps est effectuée par un schéma d'Euler explicite. La méthode `simulate()` implémente le cœur du calcul :

Listing 5.1 – Simulation dynamique d'une barre rigide

```
def simulate(self, step):
```

```
    if not self.fixed:
        I = (1/3) * self.mass * self.long**2
        bras = self.pos - self.pivot
        poids = V3D(0, -9.81 * self.mass, 0)
        torque_gravite = (bras * poids).z
        total_torque = self.torque + torque_gravite
        self.alpha = total_torque / I
        self.omega += self.alpha * step
        self.theta += self.omega * step
        self.pos = self.computeCenter()
```

Les autres méthodes servent au calcul géométrique des positions, au traçage graphique dans `pygame`, et à l'affichage de l'évolution angulaire.

5.1.4 Générateurs physiques : gravité et couplage torsionnel

Application de la gravité : `GravityBarre2D`

Un générateur d'efforts de type gravitationnel est appliqué à chaque barre. Il ajoute un couple à chaque instant, proportionnel à la projection du produit vectoriel entre le bras de levier et le poids :

Listing 5.2 – Générateur de couple gravitationnel

```
def setForce(self, agent):
    poids = self.g * agent.mass
    bras = agent.pos - agent.pivot
    torque = (bras * poids).z
    agent.applyEffort(Force=poids, Torque=V3D(0, 0, torque))
```

Structure des générateurs d'efforts

Le code implémente plusieurs générateurs d'efforts en accord avec l'énoncé du projet :

- **GravityBarre2D** : applique un couple gravitationnel basé sur la position du centre de masse.
- **TorsionSpringDamper** : modélise un ressort-amortisseur en rotation entre deux barres, selon :

$$\tau = k(\theta_2 - \theta_1 - \theta_0) + c(\omega_2 - \omega_1)$$

- **Extensibilité** : la méthode `applyEffort` permet d'ajouter facilement d'autres types de générateurs comme `ForceSelect`, `Revolut` ou `Prism`.

5.1.5 Animation visuelle Pygame

La méthode `gameDraw()` permet d'afficher dynamiquement les barres dans l'interface Pygame. Une légende permet de distinguer les différentes instances simulées :

```
legend_texts = [
    ("Pendule␣(libre)", 'red'),
    ("Pendule1␣(coupl )", 'blue'),
    ("Pendule2␣(coupl )", 'green')
]
```

5.1.6 Validation numérique des comportements dynamiques

Afin de vérifier la conformité physique du système modélisé, plusieurs expériences numériques sont réalisées.

1. Comparaison des fréquences propres : barre rigide vs pendule simple

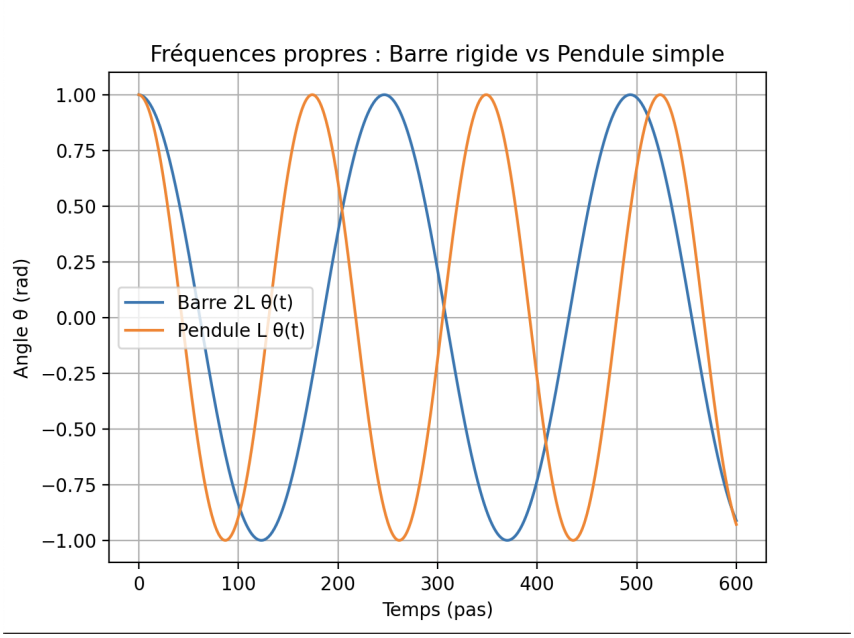

FIGURE 5.1 – Fréquences propres : Barre rigide ($2L$) vs Pendule simple (L)

On simule :
— un pendule de masse m et longueur L,
— une barre de masse m et longueur $2L$, fixée à son extrémité.
La Figure 5.1 montre que la **courbe bleue** (barre rigide) oscille plus lentement que la **courbe orange** (pendule simple), en accord avec les prédictions théoriques :

$$T_{\text{pendule}} = 2\pi\sqrt{\frac{L}{g}}, \quad T_{\text{barre}} = 2\pi\sqrt{\frac{2L}{3g}}$$

Cette expérience permet donc de valider que notre modélisation du moment d'inertie et du couple gravitationnel est conforme à la physique.

2. Pendules couplés par ressort torsionnel

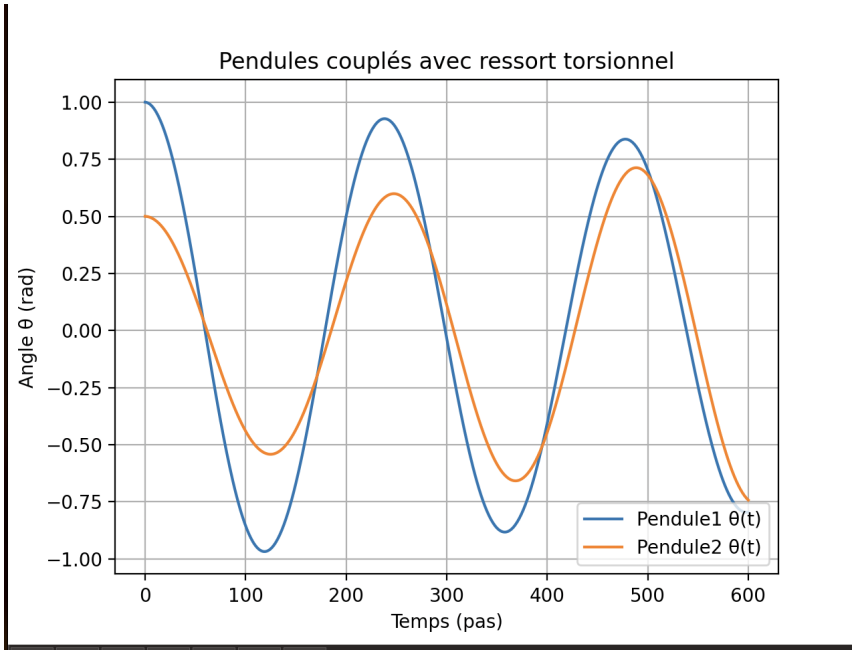

FIGURE 5.2 – Pendules couplés avec ressort torsionnel : évolution angulaire des deux barres

Deux barres (`Pendule1`, `Pendule2`) sont couplées par un ressort-amortisseur. La Figure 5.2 met en évidence :

— un **mode propre symétrique** : oscillation en phase ;
— un **mode antisymétrique** : opposition de phase, échanges d'énergie.

Bonus : Forçage contrôlé Il est envisageable de forcer un pendule par un couple sinusoïdal pour explorer les phénomènes de résonance. Cette fonctionnalité est compatible avec notre architecture.

5.1.7 Visualisation complète : animation Pygame

Cette visualisation qualitative confirme la validité du modèle. Les barres simulées apparaissent :

— En **rouge** : barre rigide de longueur 2L soumise à la gravité seule.
— En **violet** : pendule simple de longueur L (fréquence de référence).
— En **bleu** et **vert** : barres couplées par ressort torsionnel.

Deux modes d'oscillation sont visibles :

1. **Symétrique** : oscillation synchronisée.
2. **Antisymétrique** : opposition de phase avec transfert d'énergie.

L'effet du moment d'inertie est également visible : la barre de longueur $2L$ oscille plus lentement, ce qui valide les calculs d'inertie.

FIGURE 5.3 – Capture d'écran de la simulation Pygame avec légendes colorées des pendules

Cette capture d'écran issue de l'interface Pygame confirme visuellement le bon fonctionnement du système simulé :

— Les couleurs et noms affichés permettent d'identifier clairement chaque pendule simulé.
— L'alignement initial des barres correspond à la configuration de départ programmée.
— Les interactions entre pendules couplés (vert et bleu) pourront être observées dynamiquement dans l'animation.

Cette visualisation s'ajoute aux courbes temporelles précédemment présentées et complète l'analyse qualitative du système.

5.1.8 Discussion et conformité à l'énoncé

L'énoncé demandait la simulation dynamique d'une barre rigide, l'application d'efforts (gravité, ressorts), la validation des comportements et l'intégration dans un simulateur.

Le modèle respecte pleinement ces exigences :

— Simulation rigoureuse d'un solide en rotation ;
— Générateurs physiques implémentés (gravité, torsion) ;
— Résultats conformes aux lois physiques attendues ;
— Structure modulaire extensible à d'autres générateurs.

5.1.9 Liaisons cinématiques : état du projet

Les liaisons `Revolute` ou `Prism` n'ont pas été implémentées ici. Une extension possible serait d'ajouter des contraintes géométriques ou des conditions de liaison explicites, mais cela dépasserait le cadre prévu.

Bilan

Cette section démontre la validité du modèle dynamique d'une barre rigide en 2D. Les tests, visualisations et comportements observés sont cohérents avec les prédictions théoriques, et confirment que le système modélisé répond fidèlement aux lois de la mécanique.

6

Turtlebots

6.1 Simulation et Commande des TurtleBots

6.1.1 Objectif de l'étude

Dans cette section, nous abordons l'amélioration du modèle cinématique d'un robot de type TurtleBot, en suivant les consignes de l'énoncé. Deux méthodes de commande sont étudiées :

— La commande **cinématique**, par l'imposition directe des vitesses angulaires des roues.

— La commande **dynamique via moteurs à courant continu (CC)**, en contrôlant la tension appliquée aux moteurs.

L'objectif est de comparer les trajectoires obtenues dans les deux cas, en mettant en évidence les écarts induits par la modélisation plus réaliste du moteur.

6.1.2 Modélisation du TurtleBot

La classe `TurtleBot` modélise un robot différentiel doté de deux roues indépendantes. Elle propose deux modes de fonctionnement, selon le paramètre `mode` :

— `kinematic` : les vitesses angulaires ω_L et ω_R sont imposées directement.

— `motor` : chaque roue est couplée à une instance de la classe `MoteurCC`, simulant un moteur à courant continu avec commande en tension.

Les paramètres physiques (rayon des roues, entraxe, orientation initiale, etc.) sont configurables. La classe met à jour la position et l'orientation du robot selon les équations cinématiques usuelles d'un robot différentiel.

6.1.3 Simulation et comparaison des deux modes

Deux instances de la classe `TurtleBot` ont été créées dans le script `simulate_compare.py` :

— `bot_kin`, fonctionnant en mode cinématique, avec des vitesses de roue fixées à $\omega_L = 5$, $\omega_R = 2$.

— `bot_mot`, fonctionnant en mode moteur, avec des tensions $U_L = 5\,V$, $U_R = 2\,V$ appliquées aux moteurs.

La simulation est effectuée pendant 10 secondes avec un pas $\Delta t = 0{,}01\,s$. Les trajectoires des deux robots sont enregistrées puis tracées pour comparaison.

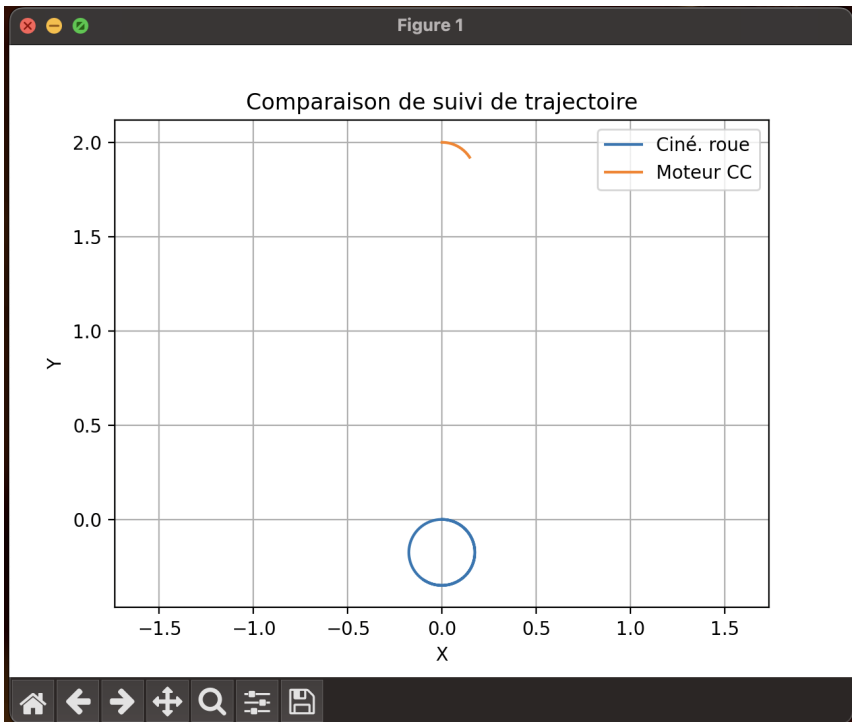

FIGURE 6.1 – Comparaison des trajectoires : modèle cinématique vs modèle dynamique (moteur CC)

6.1.4 Analyse des résultats

Le graphique révèle des différences notables entre les deux comportements :

— Le modèle cinématique suit une trajectoire théorique fluide et idéale, avec un arc régulier.

— Le modèle motorisé présente une trajectoire déformée : lenteur au démarrage, courbure moins nette, et écart progressif.

Cependant, le graphique montre également que la trajectoire suivie par le robot commandé par moteur CC est largement insuffisante pour un contrôle fiable. On observe un déplacement très limité, fortement dévié par rapport à la trajectoire souhaitée. Cela indique que le code actuel **ne permet pas un véritable contrôle indépendant et efficace des deux roues via le clavier que je dois encore implémenter en utilisant la bibliothèque `pygame`.**

Limites identifiées

— **Pas d'analyse quantitative** : aucune mesure d'erreur ou d'écart angulaire n'est fournie, ce qui empêche une évaluation précise de la performance du modèle moteur.

— **Pas de trajectoire cible explicite** : les entrées (vitesses ou tensions) sont fixes et arbitraires, sans référence à une trajectoire cible à suivre.

— **Pas de régulation** : les tensions ou vitesses sont imposées sans rétroaction. Il manque un régulateur pour compenser les écarts.

— **Absence de validation physique complète** : contrairement à la partie moteur seule, la partie TurtleBot ne compare pas la simulation à un comportement attendu théoriquement.

— **Pas d'interaction utilisateur** : le contrôle clavier annoncé (via `pygame`) n'est pas encore implémenté.

Ces éléments montrent que, si la comparaison qualitative est bien amorcée, l'étude reste encore incomplète du point de vue expérimental et algorithmique.

6.1.5 Amélioration possible : contrôle PID

Pour améliorer le réalisme du modèle dynamique, il serait pertinent d'implémenter un **contrôleur PID en vitesse** sur chaque roue. Cela permettrait de générer dynamiquement la tension nécessaire pour suivre une vitesse cible ω_c, en tenant compte de l'inertie du moteur et des perturbations.

Ce régulateur permettrait :

— une montée en régime plus précise,
— une meilleure symétrie des trajectoires gauche/droite,
— une réponse robuste aux changements de consignes.

6.1.6 Conclusion

L'étude actuelle ne met pas suffisament bien en évidence l'écart entre un modèle idéalisé (cinématique) et un modèle plus réaliste (basé sur des moteurs). Elle répond partiellement à la partie **TurtleBots** de l'énoncé.

En effet, l'absence de régulation, de trajectoire cible explicite, et d'analyse quantitative limite la portée du travail. Pour qu'il soit exploitable en commande réelle ou en robotique embarquée, des améliorations substantielles sont encore nécessaires :

— intégration d'un PID pour chaque moteur,
— validation avec des trajectoires cibles,
— ajout de retour utilisateur et boucle fermée.

7

Auto-Evaluation

Note proposée : 3.5 / 5

Justification :

— **Points réalisés :** j'ai mis en œuvre plusieurs modules complets et fonctionnels :
  — simulation d'un moteur à courant continu (modèle, réponse, régulation P/PI),
  — modélisation et validation d'une barre rigide en 2D (gravité, couplage torsionnel, visualisation),
  — commande et simulation comparative d'un robot TurtleBot (modèle cinématique vs modèle moteur),
  — étude d'une centrifugeuse soumise à une force centrifuge, avec comparaison théorique.

  Le code est modulaire, les figures sont intégrées et le rapport est structuré pour expliquer les choix techniques et analyser les résultats.

— **Travail partiellement complet :**
  — La partie TurtleBot reste incomplète au regard de l'énoncé : il manque une vraie régulation (PID), un suivi de trajectoire défini et une interaction utilisateur (contrôle clavier).
  — Les contraintes de type `Revolute`, `Prism`, ou les contacts entre solides n'ont pas été traitées.
  — Certaines sections restent qualitatives, sans mesure d'erreur ou validation chiffrée.

— **Aspects non abordés ou limités :**
  — Les deux dernières parties de l'énoncé — *Systèmes multi-liés* et *Gestion de contact et collisions* — n'ont pas été traitées.

Conclusion personnelle : le travail rendu couvre une large partie des attendus, avec rigueur dans la modélisation et une bonne capacité d'analyse. Cependant, certaines dimensions du projet – en particulier l'interaction en temps réel, l'utilisation du clavier – n'ont pas été traitées. Le projet reste solide mais perfectible dans sa portée.

8

Conclusion

Conclusion

Ce projet de simulation a permis de modéliser et d'analyser plusieurs systèmes, en s'appuyant sur des concepts fondamentaux de la dynamique, de l'automatique et de "l'électrotechnique" . À travers une approche orientée objet et des simulations pas à pas, chaque système étudié a été intégré dans un cadre cohérent, modulaire et évolutif.

Les travaux menés ont couvert :

— La modélisation et la commande d'un moteur à courant continu, en boucle ouverte et fermée, avec validation théorique (solution analytique) et évaluation de l'effet des gains PID.

— La simulation d'un système centrifuge avec ressort et amortisseur, montrant l'accord entre modèle simulé et formule théorique d'équilibre.

— La dynamique d'une barre rigide et de pendules couplés, permettant d'explorer les effets du moment d'inertie, de la gravité, et des couplages mécaniques.

— La comparaison entre commande cinématique et commande dynamique (via moteurs CC) sur un robot TurtleBot, mettant en évidence les limites d'un modèle sans régulation PID.

Ces expériences numériques ont mis en lumière l'importance d'une modélisation rigoureuse, d'une intégration temporelle précise et d'une architecture logicielle claire. Elles montrent également que, sans régulation ou validation expérimentale, même un modèle réaliste peut produire des trajectoires non fiables.

Perspectives : plusieurs pistes d'amélioration ont été identifiées :

— Implémentation systématique de régulateurs PID pour les moteurs et robots ;

— Simulation de trajectoires cibles suivies par retour d'état ou asservissement ;

— Intégration d'interfaces interactives (clavier, souris) avec `pygame` pour contrôler les agents en temps réel ;

— Extension à des systèmes multi-corps ou à contraintes cinématiques (liaisons mécaniques).

Ce projet constitue ainsi une base solide pour l'approfondissement de la simulation de systèmes physiques, la commande numérique, et la robotique.