Compte rendu

Projet:

Simulation robotique

Université: *Sorbonne Université*

Spécialité: *SAR*

Cour: *Simulation*

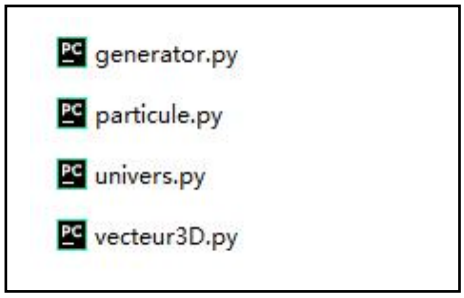
Enseignant référent: *Sinan Haliyo*

Etudiant: *LU Zhichen*

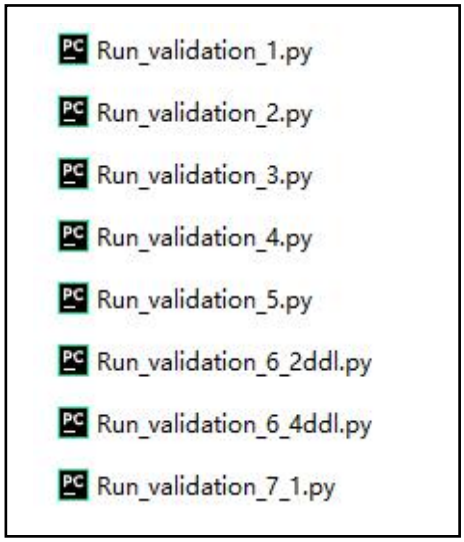
Numéro d'étudiant: *21117174*

1. Description du travail réalisé :

a) J'ai conçu les trois fichiers requis pour le projet. J'ai ajouté du code de test simple dans le fichier "Univers" respectivement, afin de vérifier leur fonctionnement. En revanche, les fichier "Generateor" et "Particule" ne contiennent pas de code de test, car tous les codes de ce fichier sont bien appliqués et démontrés dans les 7 questions de la partie "Validation".



b) J'ai rédigé chaque question de validation dans des fichiers individuels correspondants, nommés "Run_validation_x.py" où "x" représente le numéro de la question. Chaque fichier peut être exécuté directement sans nécessiter de débogage supplémentaire, et ils se sont tous exécutés avec succès.



2. Explication des choix techniques

a) Classe Particule :

i. __init__(name, color, masse, fix=False, pos=V3D(), vit= V3D(), acc=

V3D(), force= V3D())

ii. PFD()
iii. getPosition()
iv. getSpeed()
v. simule(step)
vi. setForce(force)
vii. setSpeed(speed= V3D())
viii. setPosition(position= V3D())
ix. plot2D() et plot3D(ax)
x. gameDraw(window)

b) Classe Univers :

i. __init__(name, dimension, scale, step=0.1, background=(255, 255, 255))
ii. addAgent(*particule, reset=True)
    # Voici cette méthode qui est séparée en deux situations :
    # Si reset est True, cela signifie l'initialisation de la population dans cet univers
    # Si reset est False, cela signifie que la population n'est pas initialiséedans cet univers."addSource(*generator)
iii. simule()
iv. simuleAll(time)
v. gameInit()
vi. gameUpdate(active=True, fps=60, instruction="*String*")
    # ***active*** est pour contrôle de l'exécution du projet, ***instrcution*** est pour afficher les instruction du clavier
vii. plot2D() et plot3D()

c) Des classes 'générateur' :

i. Class Gravity :

1. __init__(univers, *particule, g=V3D(0, 0, -9.81))
2. set_force()

ii. Class ForceConst :

1. __init__(force= V3D())
2. set_force(univers, *particule)

iii. Class ForceHarmonie :

1. __init__(pulsation, force= V3D())
2. set_force(univers, *particule)

iv. Class Viscosity :

1. __init__(coef, univers, *particule)
2. set_force()

v. Class ForceField :

1. __init__(amplitude, position= V3D())
2. set_force(univers, *particule)

vi. Class SpringDumper :

1. __init__(stiffness, dumping, length)
2. set_force(particule1, particule2)

vii. Class Rod:

1. __init__(particule1, particule2)
   # Ici, les deux particules sont passées en paramètre afin d'obtenir la différence scalaire des positions actuelles des deux particules, qui sera utilisée automatiquement comme la longueur de la tige
2. set_force(particule1, particule2)
   # Ici , set force sur les deux particules et longueur de la tige est fixée dans la méthode __init__()

viii. Class PrismJoint:

1. __init__(axe)
   # Par défaut, ***axe*** est utilisé pour permettre à la particule de glisser le long de l'axe x. Pour l'axe y : V3D(0,1,0,).
2. set_force(particule)
   # Passer la force correspondante projetée dans la direction spécifiée.
3. control_auto(particule1, particule2)
   # ***Particule1*** est définie comme la particule située en dessous, qui peut seulement glisser dans la direction correspondante. ***Particule2*** est la particule située au-dessus et peut se déplacer librement.

3. Implémentations différentes

a) Implémentation de la méthode "set_force()" :

Dans chaque classe du fichier "generator", j'ai ajouté une méthode de classe "set_force()" qui remplace la méthode de classe "addSource()" de la classe "Univers". À mon sens, l'application d'une force à un seul particule et l'application d'une force à toutes les particules de l'univers peuvent être réalisées par une seule méthode de classe "set_force()". En effet, l'opération réalisée par "addSource()" consiste fondamentalement à appliquer une force à chaque particule de l'univers.

b) Interaction avec le clavier :

J'ai pris en compte la possibilité d'interagir avec le clavier pendant l'exécution du code. J'ai donc ajouté autant d'interactions clavier que possible. Pour faciliter la compréhension de la relation entre les touches et les actions correspondantes, j'ai ajouté des affichages dans la méthode de classe "gameUpdate()" de la classe "Univers" (affichés dans le coin supérieur droit de la fenêtre Pygame après l'exécution de chaque fichier de code).

c) Orientation du modèle dans le plan XY :

Dans ce projet, j'ai choisi de placer le modèle dans le plan XY car je souhaitais utiliser Pygame pour présenter visuellement le processus de fonctionnement du modèle. Pygame est particulièrement adapté aux jeux en 2D. Par conséquent, les graphiques 3D générés par Python sont orientés parallèlement au plan XY.

d) Pause de l'exécution et contrôle de la fenêtre Pygame :

Pour pouvoir mettre en pause l'exécution du code à tout moment et observer l'état instantané, j'ai ajouté l'attribut "self.active" dans la méthode de classe "gameUpdate()" de la classe "Univers". Cet attribut permet de contrôler l'exécution de la fenêtre Pygame. Pour mettre en pause, il suffit d'appuyer sur la touche Espace, ce qui arrête la mise à jour de la fenêtre et met en pause le chronomètre. Appuyer à nouveau sur la touche Espace permet de reprendre l'exécution.

e) Attribut supplémentaire dans la classe "Particule" :

J'ai ajouté un attribut supplémentaire, "self.A_con", dans la classe "Particule". Cet attribut est utilisé pour stocker l'accélération constante appliquée à la particule (dans ce cas, seulement pour l'application de l'accélération gravitationnelle g). Ainsi, lors de l'exécution de la méthode de classe "PFD", il n'est pas nécessaire de prendre en compte la force causant l'accélération constante. Les autres forces résultantes appliquées à la particule sont stockées dans la liste "self.A", ce qui signifie que l'accélération totale de la particule est donnée par "self.A_con + self.A[-1]".

f) Implémentation de la fonction "game_draw()"

Dans certains fichiers .py spécifiques, j'ai ajouté la fonction 'game_draw()'. Son but est de représenter visuellement les liaisons entre les particules, telles que les tiges ou les ressorts, en utilisant des segments de ligne dans la fenêtre Pygame. Cela permet d'obtenir une représentation plus intuitive.

Ces choix techniques ont été faits en tenant compte des exigences du projet et de l'objectif de bien présenter le fonctionnement du modèle dans la fenêtre

Pygame.

4. Description et présentation des résultats à l'aide d'illustrations.

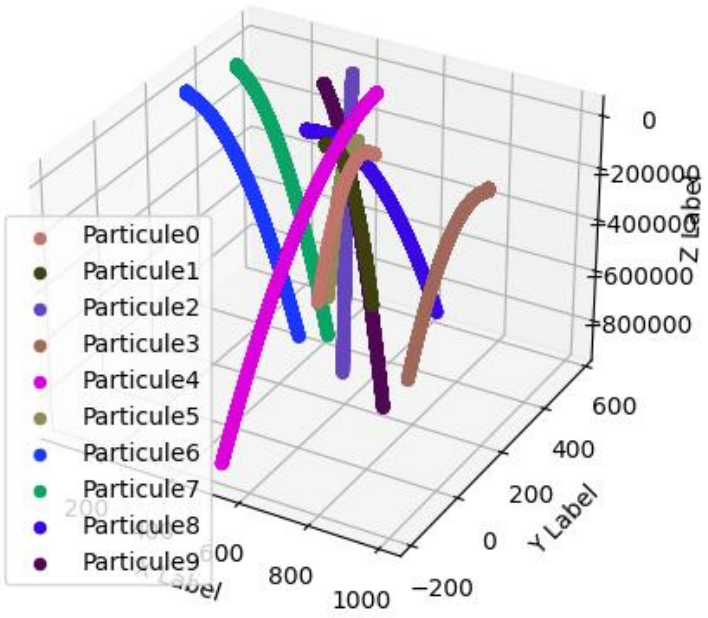
!!!Cette section doit être utilisée en conjonction avec le fichier Python, car Pygame offre une expérience plus intuitive et plus conviviale. Veuillez exécuter le fichier Python pour bénéficier d'une visualisation et d'une interaction optimales.!!!

a) Validation 1 :

Mode de fonctionnement du programme :

- Ouvrez le fichier Run_validation_1.py et l'exécuter.
- Laissez le programme s'exécuter pendant au moins 5 secondes et observez le mouvement des particules dans la fenêtre Pygame.
- Fermez la fenêtre Pygame pour terminer l'exécution du code et obtenir le graphique 3D généré par Python.

Voici l'image 3D obtenue après l'exécution du programme :



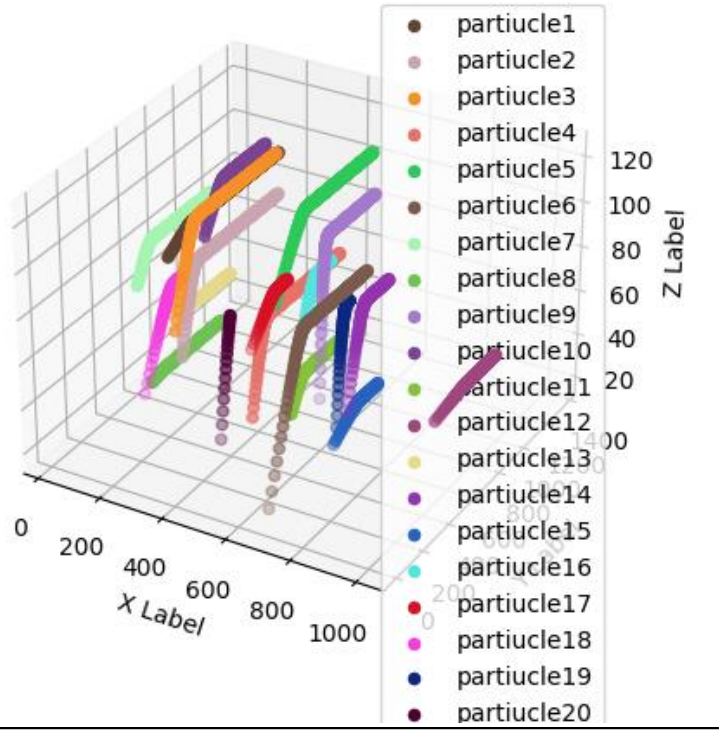
---

b) Validation 2 :

Mode de fonctionnement du programme :

- Ouvrez le fichier Run_validation_2.py et l'exécuter.
- Appuyez sur la touche Esc pour générer des particules aléatoires.
- Laissez le programme s'exécuter pendant au moins 5 secondes et observez le mouvement des particules dans la fenêtre Pygame.
- Fermez la fenêtre Pygame pour terminer l'exécution du code et obtenir le graphique 3D généré par Python.

Voici l'image 3D obtenue après l'exécution du programme :



c) Validation 3 :

Mode de fonctionnement du programme :

- Ouvrez le fichier Run_validation_3.py et l'exécuter.
- Clavier :
  - Appuyez sur la touche 1 pour appliquer une force constante dans la direction positive de l'axe y.
  -
  - Appuyez sur la touche 2 pour appliquer une force harmonique.
  - Appuyez sur la touche 3 pour annuler la force harmonique ou constante actuellement appliquée.
- Laissez le programme s'exécuter pendant au moins 5 secondes et

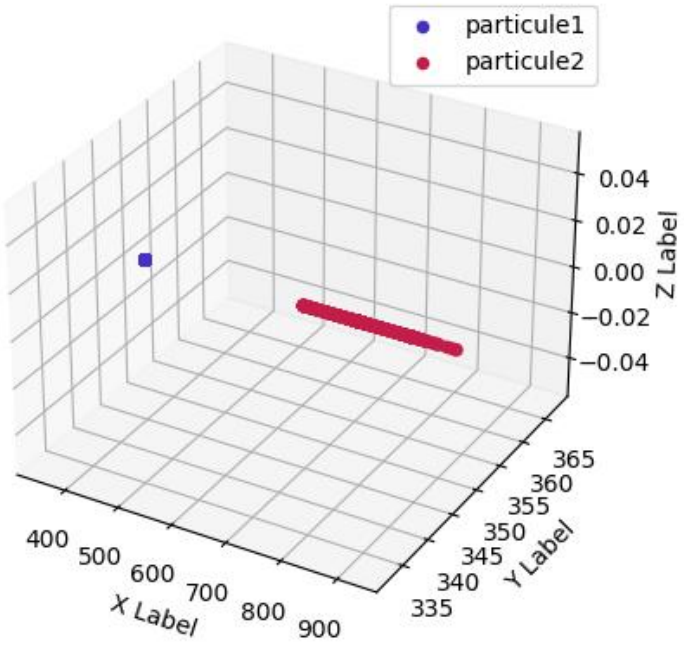
observez le mouvement des particules dans la fenêtre Pygame.

- Fermez la fenêtre Pygame pour terminer l'exécution du code et obtenir le graphique 3D généré par Python.

Veuillez noter que la logique de génération des images est la suivante : chaque fois que vous appuyez sur une touche, une image est générée représentant le mouvement des particules après l'application de la force précédente. Par exemple, après avoir lancé le code, la première pression de la touche 1 (application d'une force constante) générera une image 3D montrant la position fixe des particules (l'image de la position initiale). Ensuite, en appuyant à nouveau sur la touche 2 (application d'une force harmonique), une autre image 3D sera générée, représentant le mouvement des particules après l'application de la force constante. Enfin, en appuyant sur la touche 3 (annulation de la force supplémentaire), une autre image 3D sera générée, montrant le mouvement des particules après l'application de la force harmonique. Enfin, lors de la fermeture de la fenêtre Pygame, une dernière image 3D sera générée, montrant le mouvement des particules après l'annulation de la force supplémentaire.

Voici l'image 3D obtenue après l'exécution du programme :

Ici, seule une image est fournie car une fois une force harmonique ou constante appliquée, la trajectoire de mouvement des particules sera représentée par une ligne droite sur le graphique. Lors de l'exécution réelle, des graphiques spécifiques seront générés pour chaque cas.

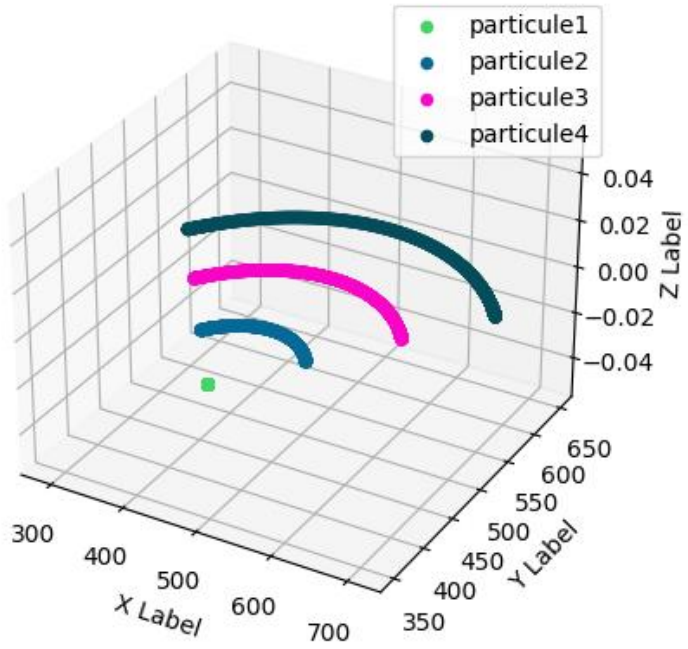


d) Validation 4 :

Mode de fonctionnement du programme :

- Ouvrez le fichier Run_validation_4.py et l'exécuter.
- Laissez le programme s'exécuter pendant au moins 5 secondes et observez le mouvement des particules dans la fenêtre Pygame.
- Fermez la fenêtre Pygame pour terminer l'exécution du code et obtenir le graphique 3D généré par Python.

Voici l'image 3D obtenue après l'exécution du programme :



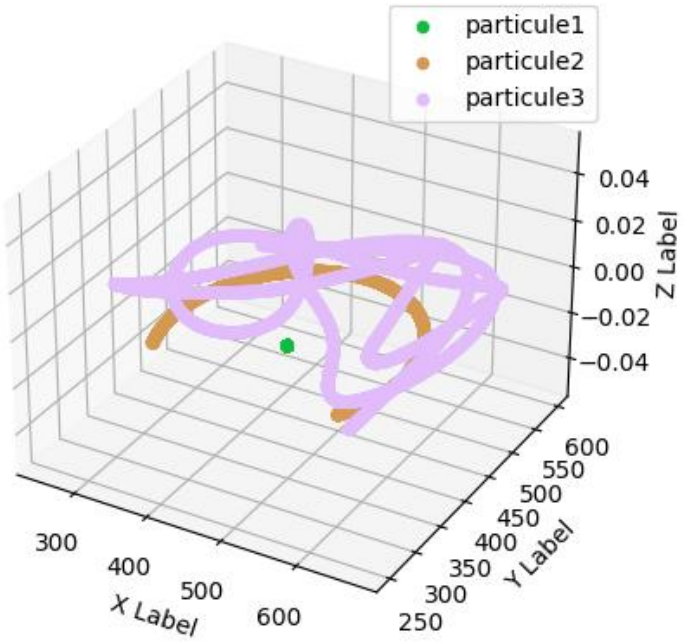
---

e) Validation 5 :

Mode de fonctionnement du programme :

- Ouvrez le fichier Run_validation_5.py et l'exécuter.
- Laissez le programme s'exécuter pendant au moins 5 secondes et observez le mouvement des particules dans la fenêtre Pygame.
- Fermez la fenêtre Pygame pour terminer l'exécution du code et obtenir le graphique 3D généré par Python.

Voici l'image 3D obtenue après l'exécution du programme :



---

f) Validation 6 :

Mode de fonctionnement du programme :

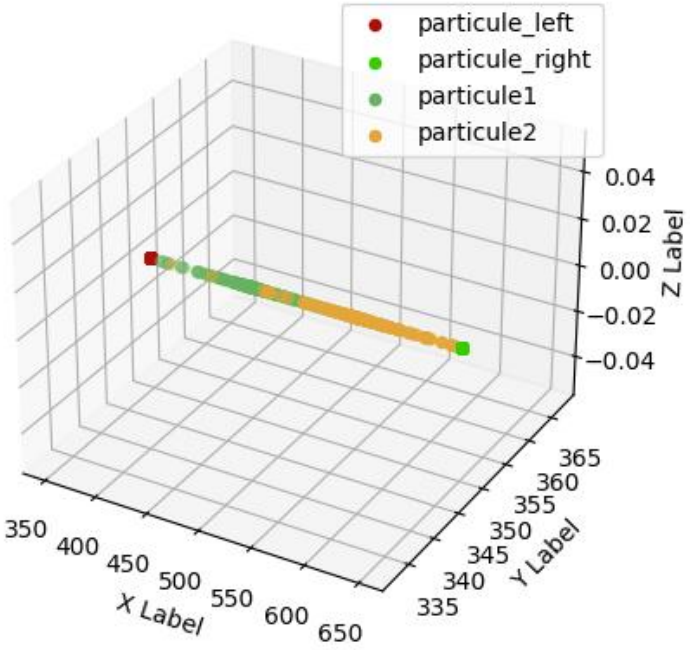
- Ouvrez le fichier Run_validation_6_2ddl.py et l'exécuter.**(Il y aussi .py pour 4ddl)**
- Clavier
  - Appuyez sur la touche 1 pour le premier modèle de vibration (même direction).
  - Appuyez sur la touche 2 pour le deuxième modèle de vibration (direction opposée).
  - Appuyez sur la touche 3 pour rétablir la position d'équilibre.
- Laissez le programme s'exécuter pendant au moins 5 secondes et observez le mouvement des particules dans la fenêtre Pygame.
- Fermez la fenêtre Pygame pour terminer l'exécution du code et obtenir le graphique 3D généré par Python.

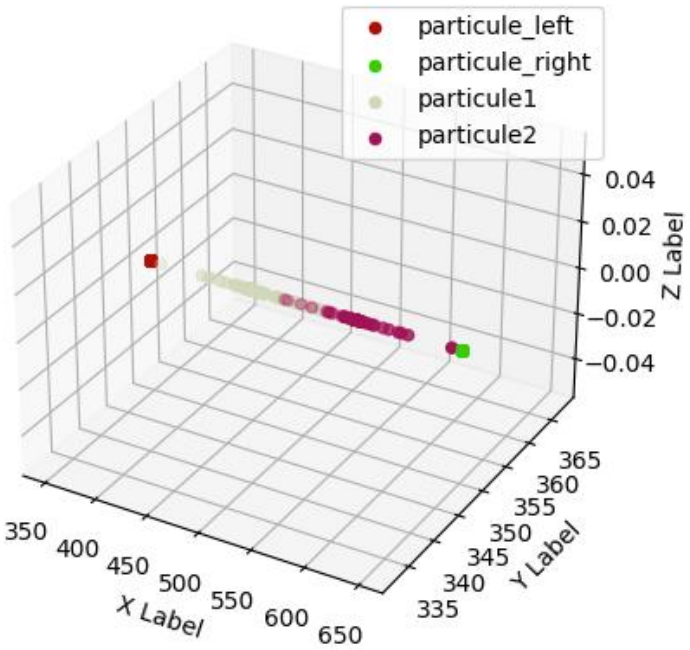
Voici l'image 3D obtenue après l'exécution du programme :

- La première image représente le modèle de vibration dans la même

direction.

- La deuxième image représente le modèle de vibration dans la direction opposée.





---

g) Validation 7 :

En raison de la sensibilité de ce modèle aux petites perturbations, il est recommandé d'exécuter d'abord le mode de contrôle automatique (auto control) avant de passer au mode de contrôle manuel. Cela évitera que le mode de contrôle automatique ne puisse pas bien contrôler les particules en raison d'une déviation importante causée par le mode de contrôle manuel.

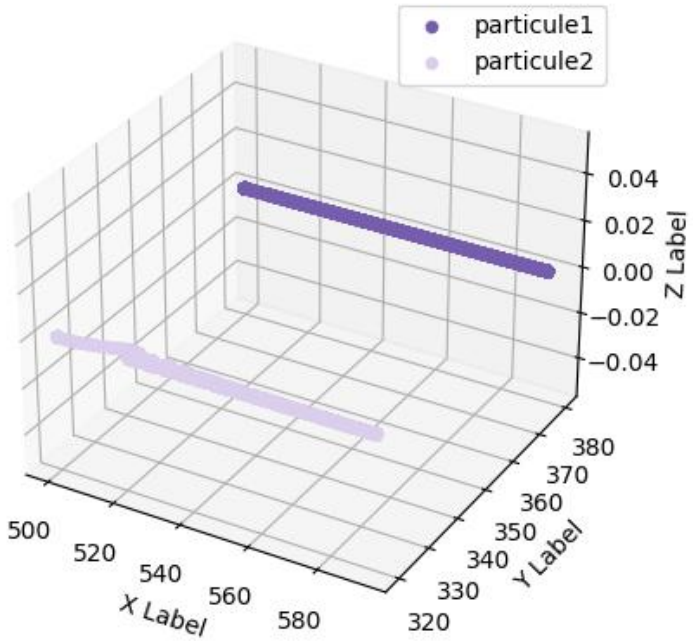
Mode de fonctionnement du programme :

- Ouvrez le fichier Run_validation_7.py et l'exécuter.
- Clavier
  - Appuyez sur la touche 1 pour activer le contrôle par les touches directionnelles de la particule inférieure. Les touches gauche et droite permettent respectivement d'appliquer une vitesse fixe à la particule inférieure. Appuyez sur la touche Esc pour quitter ce mode.

  - Appuyez sur la touche 2 pour activer le mode d'équilibrage automatique. Les touches gauche et droite permettent d'appliquer une force perturbatrice vers la gauche et vers la droite (dans la direction y) à la particule supérieure. Maintenez la touche enfoncée pour continuer à appliquer la force, mais veuillez noter que si la perturbation devient trop importante, les paramètres de contrôle actuels peuvent ne pas permettre une opération d'équilibrage satisfaisante (étant donné que l'exigence consiste à appliquer une perturbation légère, les paramètres de contrôle donnés sont relativement faibles afin de mieux observer le processus de contrôle automatique). Appuyez sur la touche Esc pour quitter ce mode.

- Laissez le programme s'exécuter pendant au moins 5 secondes et observez le mouvement des particules dans la fenêtre Pygame.
- Fermez la fenêtre Pygame pour terminer l'exécution du code et obtenir le graphique 3D généré par Python.

Voici l'image 3D obtenue après l'exécution du programme auto control :



---

5. Conclusion et perspectives :

a) ***Ce projet a été pour moi un défi très intéressant, en particulier pour la simulation des forces réelles du monde physique. Cela m'a permis de mieux comprendre la complexité et l'imprécision des forces et des mouvements des objets dans le monde réel. En particulier, lors de la simulation du contrôle automatique de Double Pendulum et des pendules inversés, j'ai initialement eu du mal à obtenir des résultats de simulation cohérents. Ce n'est que lorsque j'ai analysé la liste des forces appliquées aux particules que j'ai réalisé que les erreurs étaient dues à la valeur élevée du step de simulation. J'ai donc réduit cette valeur. De plus, pour une meilleure présentation et fluidité dans Pygame, j'ai également modifié les valeurs de fps dans certains fichiers .py spécifiques.***

Ensuite, j'ai également dû appliquer des contraintes spécifiques lors de

l'application des forces, par exemple, pour limiter le glissement de la particule inférieure sur l'axe x lors du contrôle du pendule inversé, j'ai fixé la direction de projection et j'ai défini les forces dans les directions y et z à zéro.

b) Personnellement, ce projet m'a permis de prendre conscience de l'importance d'avoir une compréhension approfondie du monde physique réel pour pouvoir simuler avec précision les différentes forces appliquées à différents objets et leurs mouvements correspondants. Je continuerai à améliorer mon code, à prendre en compte davantage de situations de forces différentes et à optimiser mon code pour le contrôle automatique du pendule inversé.

6. Auto-Évaluation et commentaires

Note attribuée: 4.5/5

Justification :

- Je m'attribue une note de 4.5/5 en tenant compte de mon assiduité, de ma participation active, des efforts fournis et du travail personnel investi dans ce cours. J'ai été présent à toutes les séances et j'ai participé activement à chaque leçon. J'ai également consacré du temps en dehors des cours pour revoir et pratiquer le contenu de chaque TP de manière approfondie.
- Concernant le projet, je suis globalement satisfait de la qualité de mes simulations, en particulier, j'ai développé une compréhension approfondie de la complexité du monde réel par rapport à la simulation. J'ai pu mettre en pratique mes connaissances et appliquer des concepts de modélisation dans mes projets. Mais je reconnais qu'il y a toujours place à amélioration. Je vais continuer à travailler sur le contrôle automatique du pendule inversé et à ajuster les paramètres de simulation pour obtenir de meilleurs résultats. De plus, je suis conscient que je peux optimiser davantage le code principal du programme pour le rendre plus concis et plus efficace.
- En conclusion, je suis motivé pour continuer à apprendre de nouvelles connaissances et à améliorer mes compétences. Je vais poursuivre mes efforts pour m'engager pleinement dans l'apprentissage et continuer à progresser dans ce domaine.