,

Rapport Bureau Etude — Automatique

Auteurs :

VIGNAUX Adrien **BLANCHARD Enzo**

Professeurs :

COTS Olivier **BEUZEVILLE Theo**

Projet de première année de SN année : 2024-2025

Table des matières

1. Introduction

1.1. Contexe

Lors des Travaux Pratiques effectués en Automatique, nous nous sommes concentrés sur la simulation et la mise en application d'un système contrôlé, plus particulièrement un pendule inversé.
Ce dernier peut se représenter de la manière suivante :

$$(S) : \left\{ \begin{array}{c} \overset{\circ}{x_1}(t) = x_2(t) \\ \overset{\circ}{x_2}(t) = \frac{g}{l}\sin(x_1(t)) - \frac{\cos(x_1(t))u(t)}{l} \\ x_1(0) = x_{0,1} = \alpha_0 \\ x_2(0) = x_{0,2} = \alpha_0 \end{array} \right\}$$

1.2. Objectif

L'objectif final de ces séances est de pouvoir contrôler un robot Lego (mentionné plus tard) à l'aide de ce système, afin qu'il soit dans un état d'équilibre : il doit tenir debout.

Robot Lego

2. Simulation d'un pendule inversé contrôlé (TP02)

2.1. Situation

En utilisant Simulink (un outil inclus dans le logiciel Matlab), nous allons modéliser le système du pendule inversé.

2.2. Contrôle par retour d'état

Pour réaliser ce système, nous devons écrire un retour d'état de la forme :

$$u(t) = \text{ue} + K * (x - \text{xe})$$

A l'aide des conditions énoncés au début, voici ce qu'il en résulte sur Simulink :

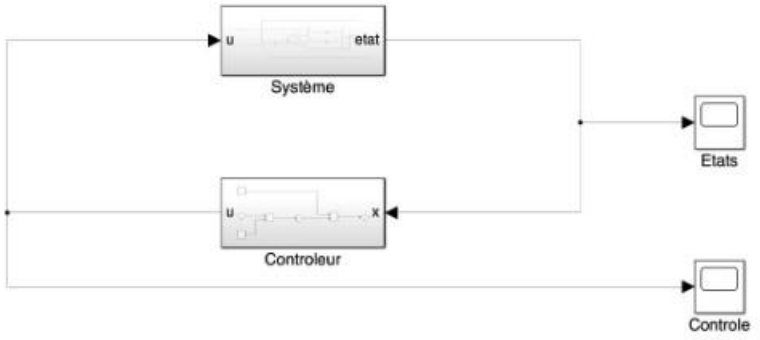

Fig. 1. – Premier schema de modelisation du pendule inversé

Nous allons tester différentes situations à l'aide de ce tableau de données :

Cas	x_0	t_f	K	**Intégrateur**
Cas 1.1	$(\pi/20, 0)$	10	$(30, 10)$	ode45
Cas 1.2	$(\pi/20, 0)$	100	$(10, 1)$	ode45
Cas 1.3	$(\pi/20, 0)$	100	$(10, 1)$	Euler, ode1
Cas 1.4	$(\pi/20, 0)$	1000	$(10, 1)$	Euler, ode1
Cas 1.5	$(\pi/10, 0)$	100	$(10, 1)$	ode45
Cas 1.6	$(\pi/10, 0)$	100	$(30, 10)$	ode45

Tableau de données TP02

Il faut noter que la courbe bleue représente la position de notre système (qui doit tendre vers 0 pour se stabiliser), tandis que la courbe jaune représente la tension nécessaire au contrôle du système afin de le stabiliser.

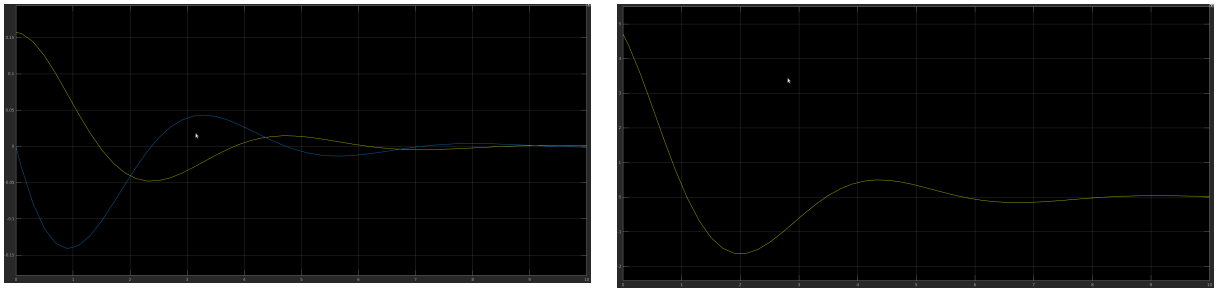

Fig. 2. – Courbe Etat, Contrôle pour le cas 1.1

Alors que la position n'est pas à un état d'équilibre, la tension de contrôle régule petit-à-petit cette dernière : les oscillations s'atténuent jusqu'à ce que la position soit stabilisée.

C'est une situation qui répond bien aux critères exigés.

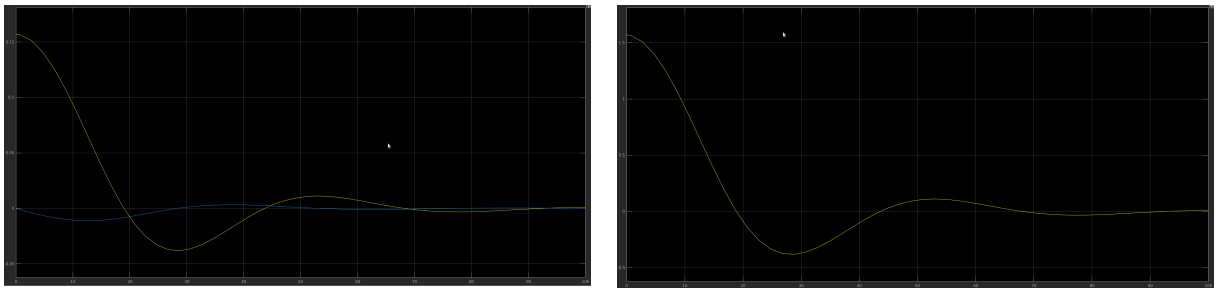

Fig. 3. – Courbe Etat, Contrôle pour le cas 1.2

Cette fois, le gain K étant différent, le temps de réponse pour atteindre l'équilibre est nettement plus important que précédemment.

Il faut donc bien choisir ce paramètre afin d'optimiser les performances du système et assurer au plus vite son équilibre.

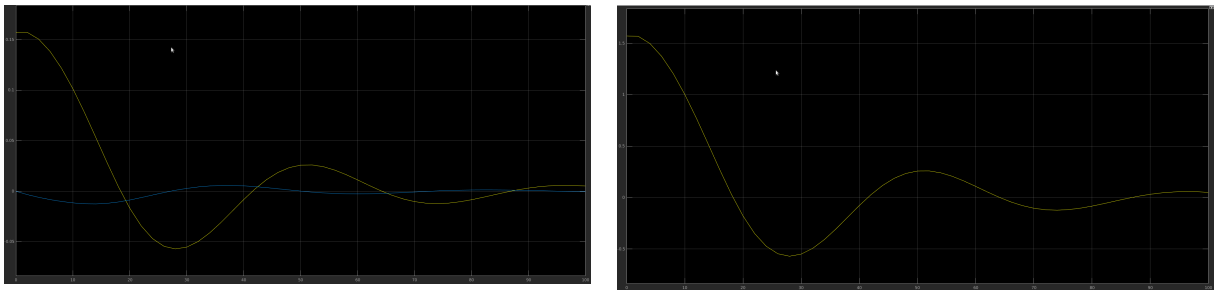

Fig. 4. – Courbe Etat, Contrôle pour le cas 1.3

Nous avons décidé de changer d'intégrateur : nous passons de ode45 à Euler. On constate des résultats encore moins intéressants, avec une plus grande amplitude de notre tension de contrôle (on utilise plus d'énergie) et un temps de réponse encore plus grand.

Il est donc important de bien identifier quel intégrateur est profitable à notre système.

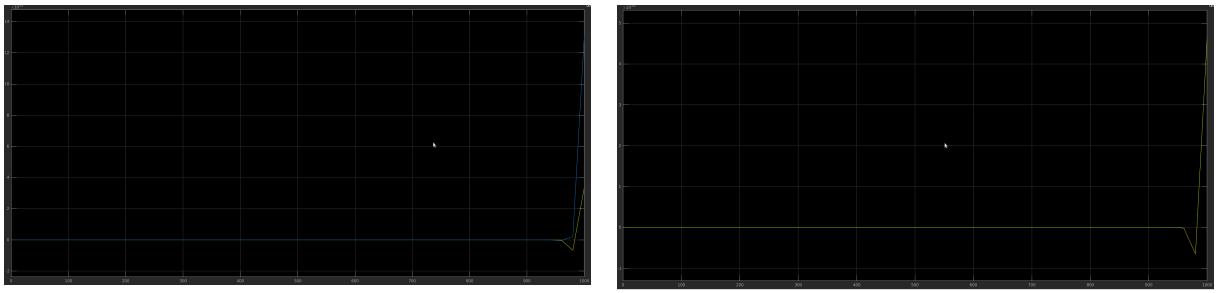

Fig. 5. – Courbe Etat, Contrôle pour le cas 1.4

Cette situation est similaire au cas précédent, cependant le temps de simulation est décuplé : cela nous permet d'apercevoir une divergence de la part du système.

Suivant la durée à laquelle il nous est nécessaire d'avoir notre système actif, cette situation rend l'utilisation de la méthode d'Euler couplé à cette constante K non pas inefficace mais inutilisable.

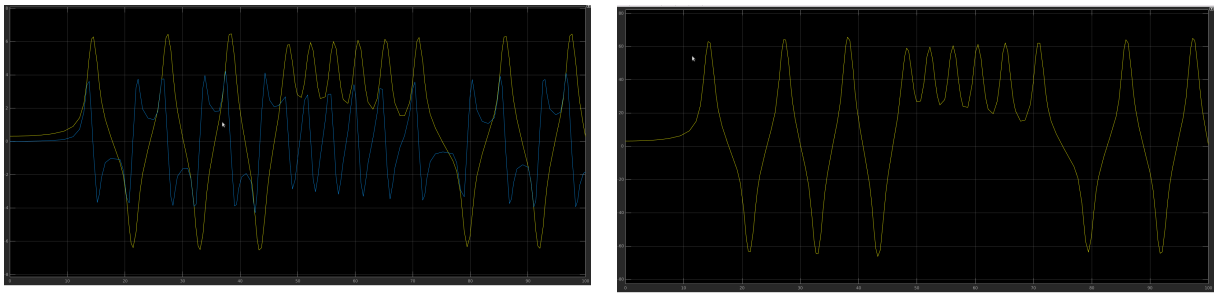

Fig. 6. – Courbe Etat, Contrôle pour le cas 1.5

Cette fois, on effectue encore un test avec le même gain K, mais on réemploie la méthode d'intégration ode45. Cependant, on remarque que le système n'arrive pas à atteindre l'équilibre : en effet, les efforts fournis par la tension de contrôle ne suffisent pas à contrôler efficacement le système, ainsi le système ne se comporte pas comme prévu, et les tentatives de contrôle sont vaines puisque ce ne sera jamais suffisant pour diminuer pas à pas les oscillations de la position afin d'atteindre l'équilibre.

Le gain doit être suffisamment important afin de répondre aux exigences de notre modèle.

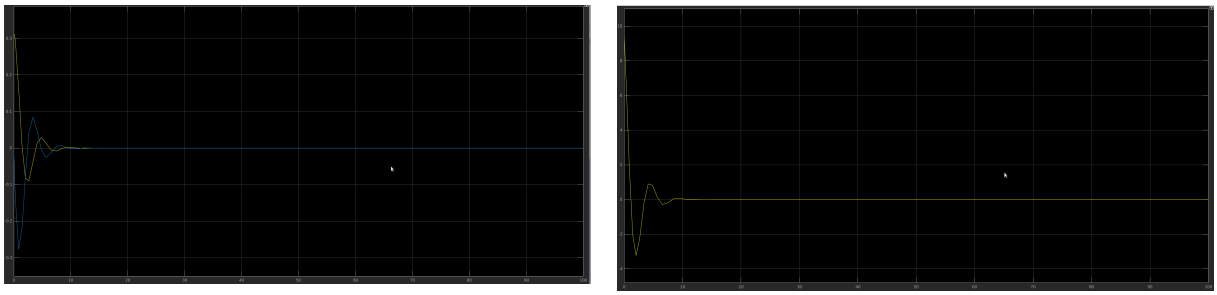

Fig. 7. – Courbe Etat, Contrôle pour le cas 1.6

Cet ultime test reprend les mêmes paramètres que le cas 1.1, à l'exception d'une position initiale différente et un temps de simulation décuplé. Les résultats sont satisfaisants : le temps de réponse tourne aux alentours de dix secondes, et le système ne diverge pas même après un long temps de simulation.

Finalement, toutes ces situations nous ont permis de comprendre les points clés afin de bien modéliser notre système du pendule inversé et réussir à atteindre l'équilibre rapidement et sans risque de rechute. Nous considérons les cas 1.1 et 1.6 comme valides et à même de répondre à nos besoins. Il est important de prendre une constante de contrôle K assez élevé pour réaliser notre modèle, et nous garderons le modèle d'intégration ode45 qui a prouvé des résultats plus satisfaisants que la méthode d'intégration d'Euler.

2.3. Implémentation de capteurs

A présent, la réalité ne nous permettra pas d'obtenir toutes les informations nécessaires au retour d'état de notre système. Les capteurs ne nous permettront de ne récupérer que la vitesse angulaire du système, ce qui va nous demander de prendre ce critère en compte.

Pour ce faire, nous allons modéliser ce capteur que nous allons implémenter dans notre schéma Simulink actuel, mais cela ne résout toujours pas le problème : il nous manque la position angulaire.

C'est alors que nous allons aussi modéliser une partie "Prédicteurs", qui va nous permettre de récupérer cette donnée manquante à l'aide de celle actuellement à notre disposition :

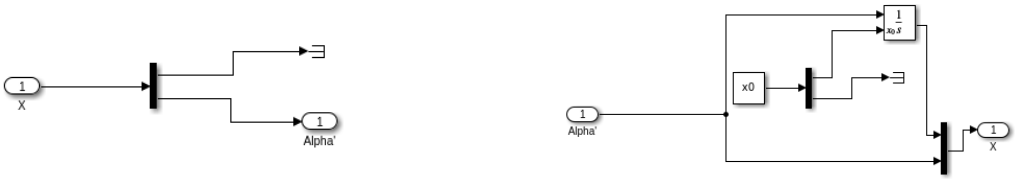

Fig. 8. – capteur et predicteur du module Simulink

A l'aide d'un intégrateur, nous arrivons à récupérer notre donnée manquante et ainsi mettre en marche notre retour d'état comme convenu.

Réalisons quelques tests pour confirmer le bon fonctionnement de notre modèle :

Cas	x_0	t_f	K	pas	Intégrateur
Cas 1	$(\pi/20, 0)$	100	$(10, 1)$	par défaut	ode45
Cas 2	$(\pi/20, 0)$	100	$(10, 1)$	0.001	Euler, ode1
Cas 3	$(\pi/20, 0)$	100	$(10, 1)$	5	Euler, ode1

Tableau de données TP02

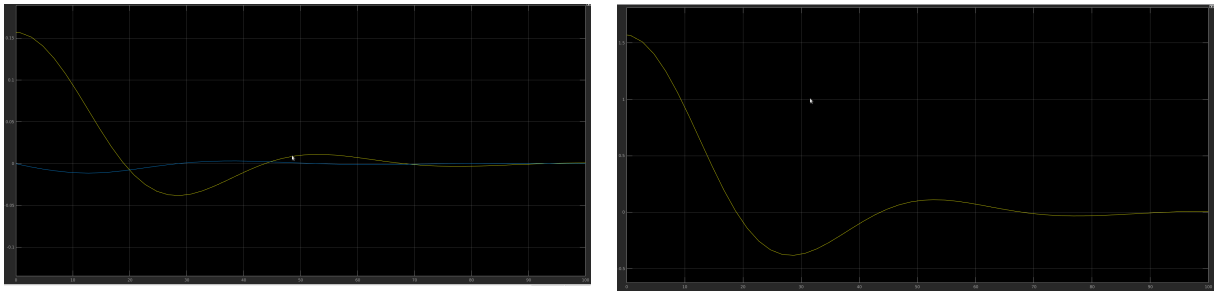

Fig. 9. – Courbe Etat, Contrôle pour le cas 1.1

Ce premier test montre que le système arrive à se stabiliser comme précédemment, bien que cela lui requiert un peu plus de temps. Cependant, c'est tout à fait normal : la récupération de la position angulaire n'arrange pas notre système, au contraire.

En revanche, ce n'est rien d'alarmant, puisque notre système respecte toujours nos exigences.

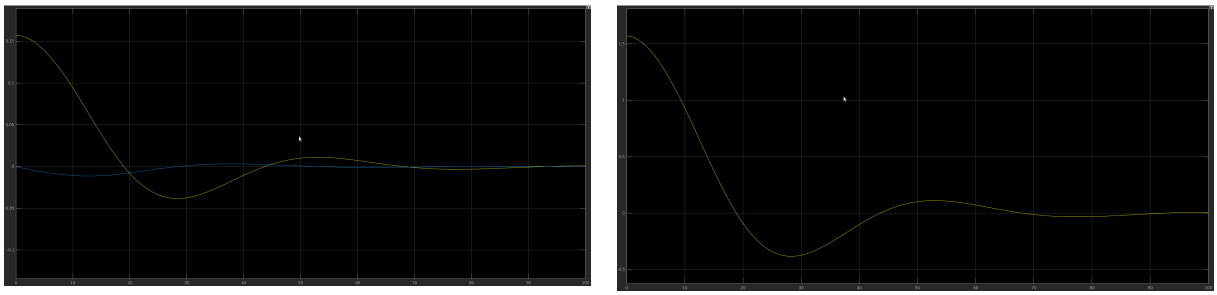

Fig. 10. – Courbe Etat, Contrôle pour le cas 1.2

La méthode d'intégration d'Euler fonctionne de manière presque identique au cas précédent : c'est lié au pas, qui étant extrêmement faible, se confond presque à un scénario continu.

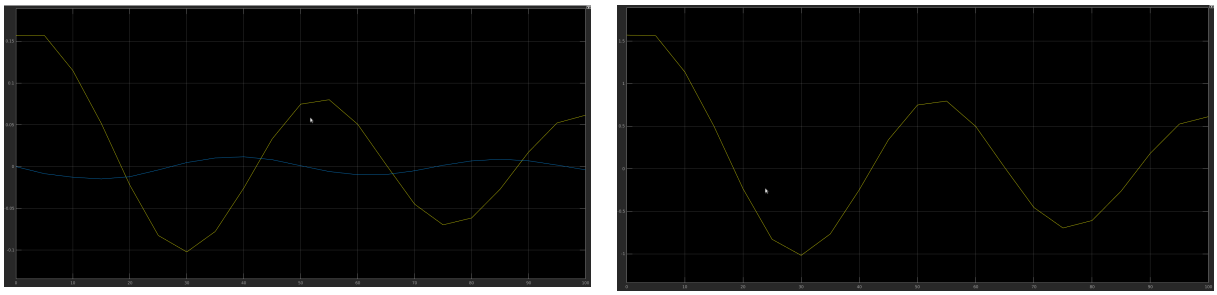

Fig. 11. – Courbe Etat, Contrôle pour le cas 1.2

Cette fois, le pas étant bien plus important, le système ne réagit pas comme escompté : il faut donc faire attention en manipulant la méthode d'Euler, le pas influe catégoriquement son efficacité.

3. Simulation du comportement d'un robot Lego (TP03)

3.1. Situation

A présent, nous allons modéliser et simuler le robot Lego pendule inversé puis constater les résultats obtenus. Nous débuterons par un modèle Simulink continu qui est le suivant :

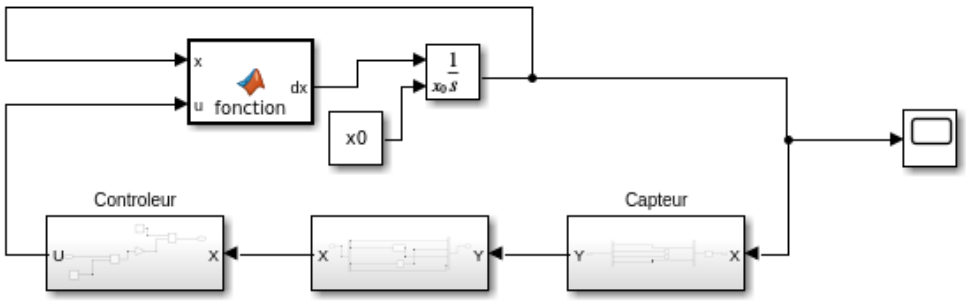

Fig. 12. – Schema de modelisation du robot

3.2. Contrôle par retour d'état

Suite à cette implantation, nous avons effectué différents tests avec les données suivantes :

Cas	x_0	t_f	K	Intégrateur
Cas 1.1	$(0,0,0,0)$	2	K	ode45
Cas 1.2	$(0,\pi/5,0,0)$	5	K	ode45
Cas 1.3	$(0,\pi/10,0,0)$	5	K	ode45

Tableau de données TP03

Tout comme au début du TP02, au départ, seul le point initial est modifié afin de constater les différents comportements du robot qui peuvent survenir.

Retrouvons dès à présent les différents résultats :

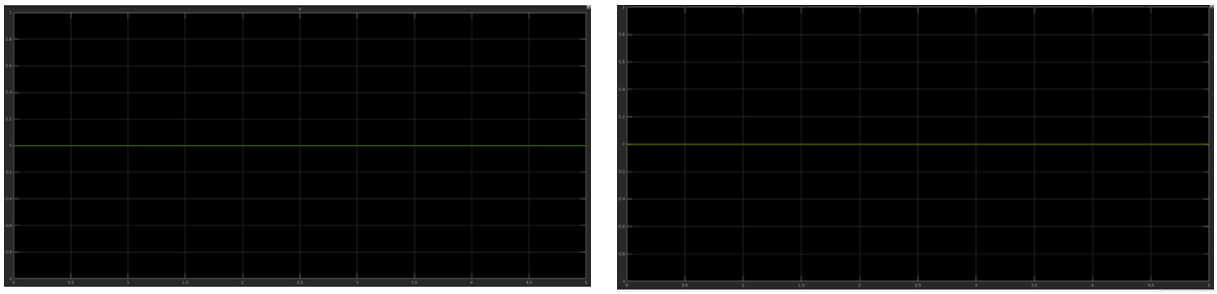

Fig. 13. – Courbe Etat, Contrôle pour le cas 1.1

Étant donné que nous sommes initialement à un point d'équilibre, il faut en effet que le système ne réagisse pas.

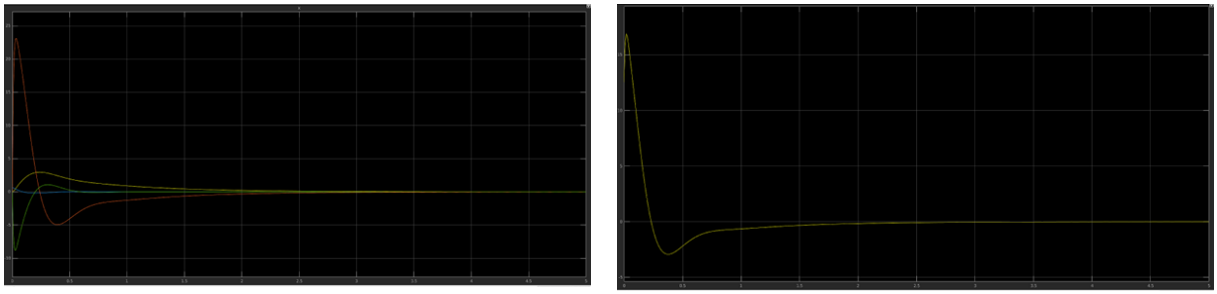

Fig. 14. – Courbe Etat, Contrôle pour le cas 1.2

Courbe bleue : angle d'inclinaison du robot

Courbe orange : vitesse moyenne des roues du robot

Courbe jaune : angle moyen des roues du robot

Courbe verte : vitesse de changement d'angle du robot.

On constate d'abord une forte augmentation des vitesses (des roues et de changement d'angle) au début de la simulation : le système effectue une régulation importante afin de stabiliser le robot. Par la suite, lorsque l'angle du robot passe d'un côté à l'autre de la valeur nulle, les vitesses tentent de réguler, de moins en moins fort, en sens inverse. Finalement, c'est au bout de deux secondes que le système atteint l'équilibre, puisque toutes les courbes convergent vers 0.

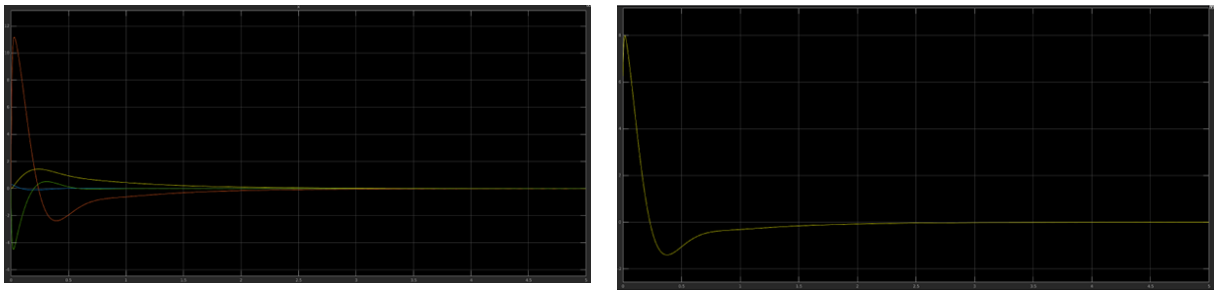

Fig. 15. – Courbe Etat, Contrôle pour le cas 1.3

Le comportement est similaire au cas 1.2, cependant le système met un petit peu plus de temps à converger vers 0, donc à se stabiliser : l'équilibre est atteint comme souhaité pour plusieurs départs différents.

3.3. Implémentation des capteurs

Nous ajoutons à présent un modèle de capteurs, qui simulent les informations que l'on récupérerait en réalité, à savoir la vitesse moyenne des roues du robot (Ψ'), ainsi que l'angle (Θ). Puisqu'il nous manque deux informations pour compléter le retour d'état, nous allons les retrouver à l'aide d'un intégrateur et d'un dérivateur, ces derniers étant implémentés dans un bloc "Prédicteur".

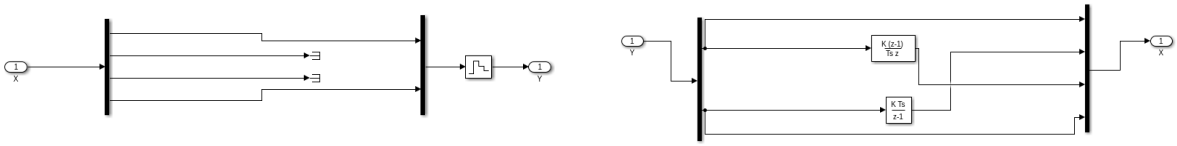

Fig. 16. – capteur et predicteur en mode discret du module Simulink

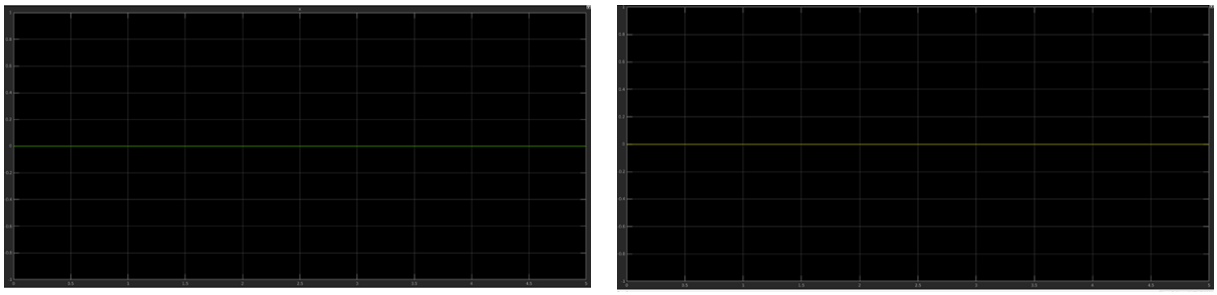

Fig. 17. – Courbe Etat, Contrôle pour le cas 1.1

Similaire au cas 1.1 de la partie précédente, un état initial à l'équilibre amène naturellement à une non-réaction de la part du système.

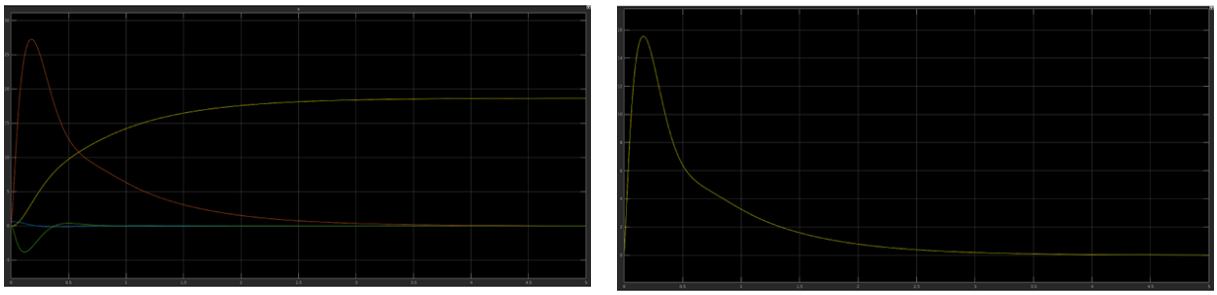

Fig. 18. – Courbe Etat, Contrôle pour le cas 1.2

Nous avons deux changements notables à la différence de précédemment : La courbe jaune, soit l'angle d'inclinaison du robot, ne converge plus vers 0 mais vers une autre valeur quelconque : nous pouvons en déduire que le robot s'est stabilisé après avoir quelque peu bougé de sa position initiale ; Le système est plus lent à se stabiliser : on se rapproche bien d'une situation réelle grâce aux capteurs modélisés.

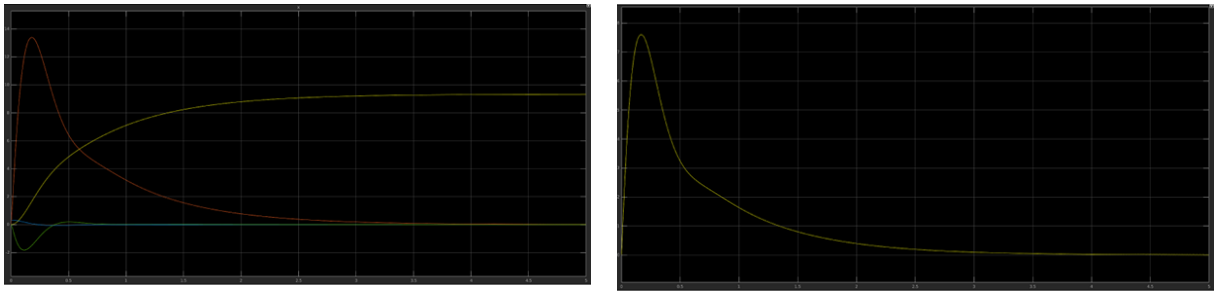

Fig. 19. – Courbe Etat, Contrôle pour le cas 1.3

On a le même attendu qu'entre les figures X et X de l'étape précédente.

3.4. Passage en mode discret

En réalité, l'implantation du modèle réalisé dans le robot se fera en modèle discret. Nous avons donc la nécessité de simuler cet état, en modifiant le capteur et le prédicteur pour satisfaire l'exigence souhaitée :

Suite à cela, nous obtenons les résultats suivant :

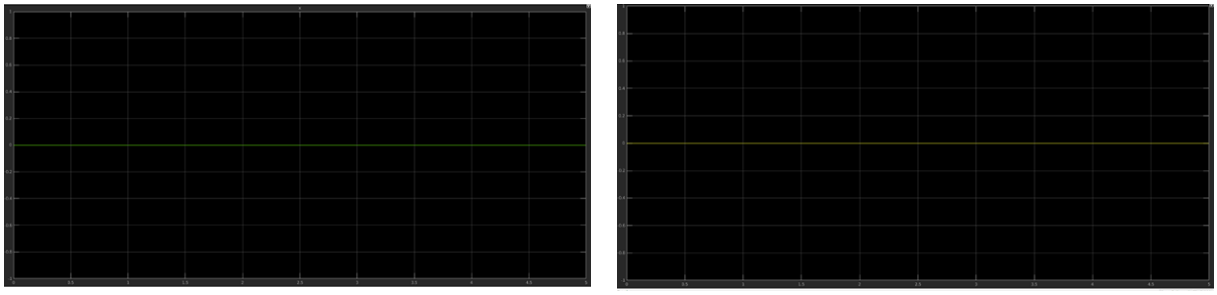

Fig. 20. – Courbe Etat, Contrôle pour le cas 1.1

L'état initial nul est et doit toujours être le même : c'est bien le cas une nouvelle fois.

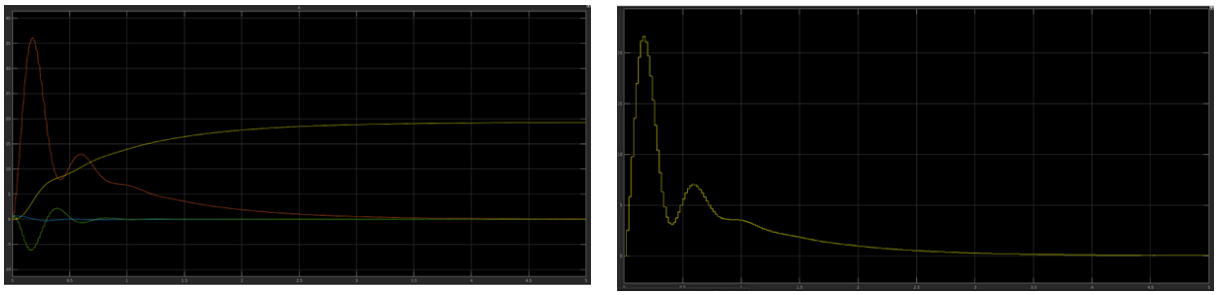

Fig. 21. – Courbe Etat, Contrôle pour le cas 1.2

On constate une déformation des courbes : elles sont à présent en escalier. Nous sommes bien sur une discrétisation de ces dernières comme désiré. Assurément, le système réagit tout aussi bien qu'avant.

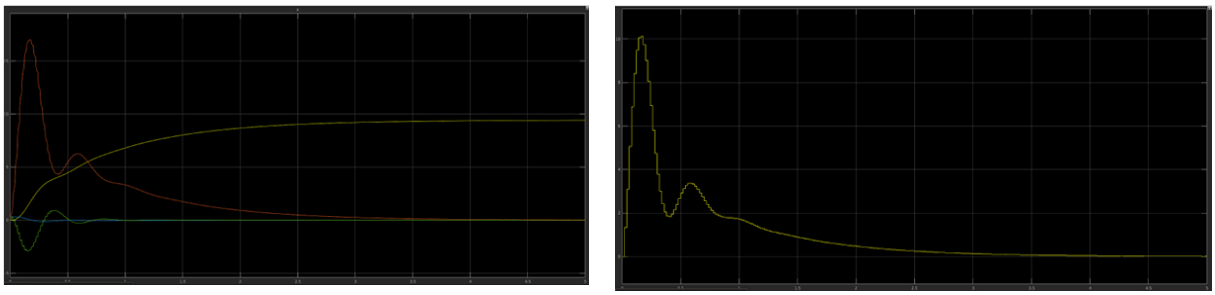

Fig. 22. – Courbe Etat, Contrôle pour le cas 1.3

Nous obtenons un résultat tout aussi convaincant, avec comme toujours les amplitudes des paramètres d'état qui sont diminuées par deux par rapport au cas 1.2.

4. Mise en pratique : Robot LegoNXT (TP04)

Après avoir validé le modèle du pendule inversé, et avoir simulé de multiples situations, nous sommes finalement rendu à l'implantation de ce modèle dans le robot LegoNXT :

Après nous avoir fourni un ensemble de fichiers écrits en langage C, nous avons exclusivement à remplir la partie de la modélisation du système, puis de transmettre ce fichier au robot afin qu'il exécute notre programme.

Nous avons pu observer le bon équilibre du robot au bout de quelques secondes, et les perturbations (pousser le robot par exemple) ne le font pas sortir de son état d'équilibre, dans la mesure du raisonnable évidemment.

5. Conclusion

C'est à l'aide de ce dernier TP que nous avons pu pleinement valider notre modèle du pendule inversé, puisque le robot se comportait comme attendu lors de nos simulations.

De plus, cette réactivité en temps réel à cause d'une quelconque perturbation montre la bonne prédiction des informations manquantes suite au passage par capteurs : le robot cherche continuellement à se retrouver en état d'équilibre.

Ainsi, nous pouvons en conclure que le modèle du pendule inversé permet de stabiliser un système à roues tel que le robot LegoNXT.