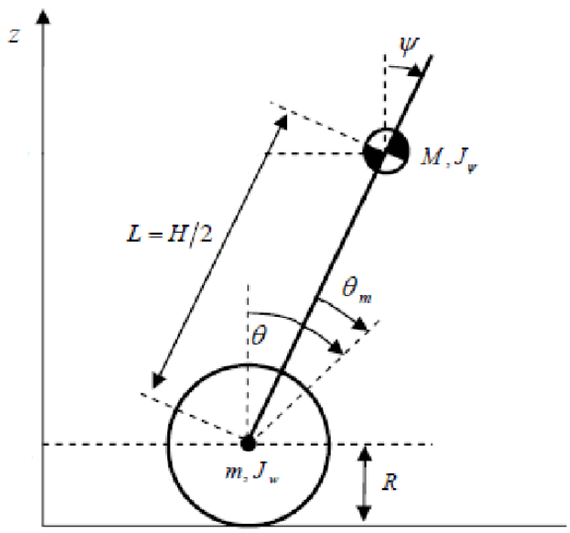
Rapport de bureau d'études
Automatique Systèmes Cyber Physique

RAGOT Cyrian
Groupe K



Département Sciences du Numérique - Première année
2023-2024

Table des matières

Introduction **3**

1 Modèle du pendule inversé **4**
1.1 Contrôle par retour d'état 4
1.1.1 Analyse du modèle théorique du pendule inverse 4
1.1.2 Simulation du modèle sur Simulink 5
1.2 Contrôle par retour de sortie avec capteur 8

2 Modèle du robot Lego **10**
2.1 Contrôle par retour d'état 10
2.1.1 Analyse du modèle théorique du robot 10
2.1.2 Simulation du modèle sur Simulink 10
2.2 Contrôle par retour de sortie avec capteur 11
2.3 Modèle hybride par discrétisation 13

3 Robot Lego NXT **15**

Conclusion **16**

Introduction

Dans le cadre du cours d'automatique, nous nous sommes familiarisés avec les notions de système et des méthodes pour étudier la stabilité de ces systèmes selon certaines conditions. Nous avons aussi introduit des méthodes de contrôlabilité de ces systèmes.

Nous avons pu alors appliquer ces connaissances dans un projet d'automatique qui s'est étendu sur plusieurs séances de TP.

Objectif : Stabiliser un robot Lego NXT verticalement.

Pour remplir cet objectif, plusieurs modèles seront étudiés en passant de la théorie mathématique à des simulations de modèles créés sur Simulink et enfin de l'implémentation dans le robot.

1 Modèle du pendule inversé

1.1 Contrôle par retour d'état

Dans cette partie, nous étudierons un modèle simple d'un pendule inversé contrôlé par retour d'état (figure 1) pour lequel on a accès aux variables de sortie. Le système contrôlé issu des équations physiques de la dynamique est

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = \frac{g}{l} sin(x_1(t)) - \frac{cos(x_1(t))u(t)}{l} \\ \dot{x}_1(0) = x_{0,1} = \alpha_0 \\ \dot{x}_2(0) = x_{0,2} = \dot{\alpha}_0, \end{cases} \tag{1}$$

avec

- $\mathbf{g = 9.81 m/s^2}$ constante de gravité
- $\mathbf{l = 10 m}$ longueur du pendule
- $\mathbf{t_0 = 0 s}$ instant initial
- $\boldsymbol{x(t) = (x_1(t), x_2(t))^\intercal = (\alpha(t), \dot{\alpha}(t))^\intercal}$ variable d'état
- $\boldsymbol{(x_e, u_e)^\intercal = (0, 0, 0)^\intercal}$ point de fonctionnement
- $\boldsymbol{u(t)}$ contrôle par retour d'état

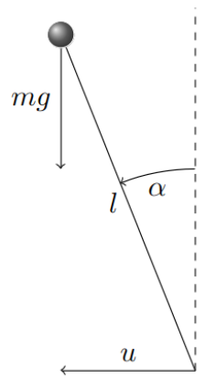


Figure 1: *Schéma du pendule inversé*

1.1.1 Analyse du modèle théorique du pendule inverse

L'équation d'état est alors

$$\dot{x}(t) = f(x(t), u(t)), \tag{2}$$

où

$$f(x, u) = \begin{pmatrix} x_2 \\ \frac{g}{l} sin(x_1) - \frac{cos(x_1)u}{l} \end{pmatrix} \tag{3}$$

On souhaite stabiliser le système à l'origine (la position verticale du pendule inversé), cependant le système non contrôlé n'est pas stable à l'origine. En effet, lorsque $u = 0$, en passant par la jacobienne de g au point de fonctionnement puis son polynôme

caractéristique, on trouve la valeur propre $\sqrt{\frac{g}{l}}$ qui est à partie réelle strictement positive.

On choisit alors un contrôle en boucle fermée par retour d'état linéaire de la forme $u(t) = u_e + K(x(t) - x_e)$ avec $K = (k_1, k_2)$. Cherchons alors K de manière à avoir un contrôle qui stabilise le système asymptotiquement en x_e.

On boucle le système :

$$\dot{x}(t) = f(x(t), u_e + K(x(t) - x_e)) := g(x(t)) \tag{4}$$

Alors $g(x_e) = f(x_e, u_e) = 0$ donc x_e est un point d'équilibre de $\dot{x} = g(x)$.

La matrice jacobienne associée est alors

$$J_g(x) = \begin{pmatrix} 0 & 1 \\ \frac{cos(x_1)}{l}(g - k_1) + \frac{u_e}{l} sin(x_1) & -\frac{cos(x_1)}{l} k_2 \end{pmatrix} \tag{5}$$

Au point de fonctionnement $(x_e, u_e) = (0, 0, 0)$ on a

$$J_g(x_e) = \begin{pmatrix} 0 & 1 \\ \frac{g-k_1}{l} & -\frac{k_2}{l} \end{pmatrix} \tag{6}$$

De plus, le contrôle stabilise asymptotiquement le système en (x_e, u_e) si et seulement si les valeurs propres sont à parties réelles strictement négatives donc si et seulement si

$$\begin{cases} tr(J_g(x_e)) < 0 \\ det(J_g(x_e)) > 0 \end{cases} \tag{7}$$

Finalement,

$$\begin{cases} k_1 > g \\ k_2 > 0 \end{cases} \tag{8}$$

1.1.2 Simulation du modèle sur Simulink

Maintenant que l'on a compris comment le système contrôlé devrait réagir, nous allons effectuer des simulations sur Simulink où nous pourrons comparer le comportement du système pour différents cas d'étude.

Les schémas blocs construits sur Simulink lors des séances de TP sont représentés sur les figures 2, 3 et 4. Nous étudierons les simulations avec les différents cas du tableau 1. On rappelle que t_f est le temps de simulation en secondes.

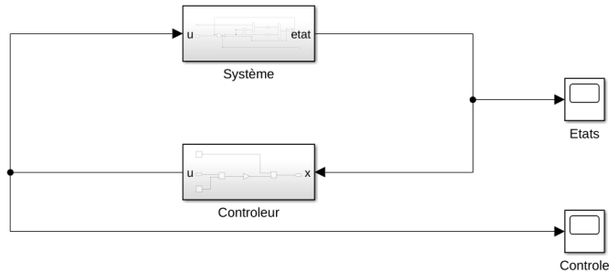


Figure 2: *Schéma bloc du système contrôlé par retour d'état*

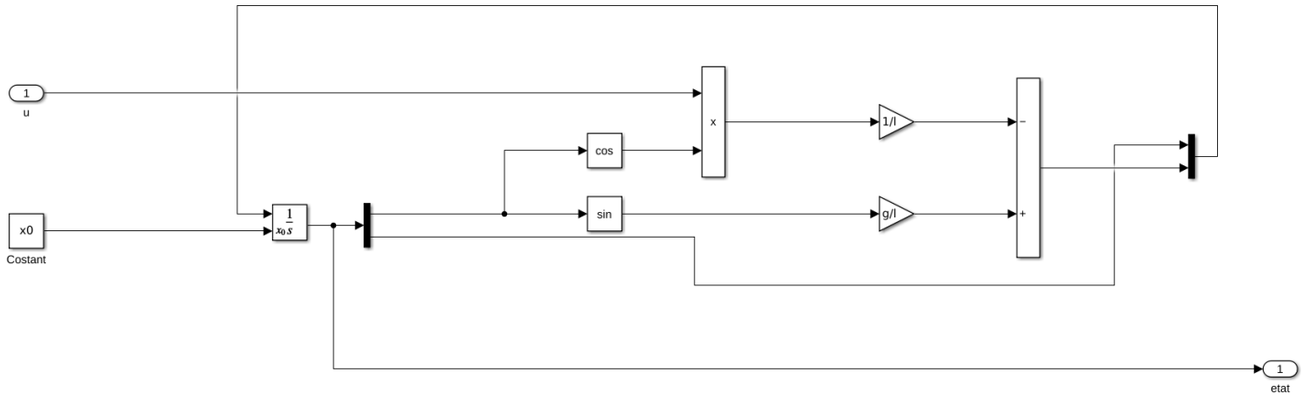


Figure 3: *Schéma bloc du système*

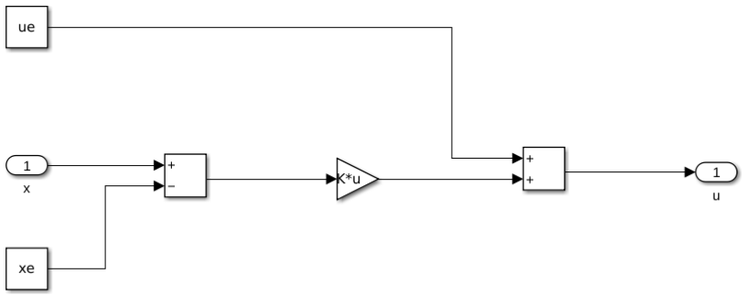


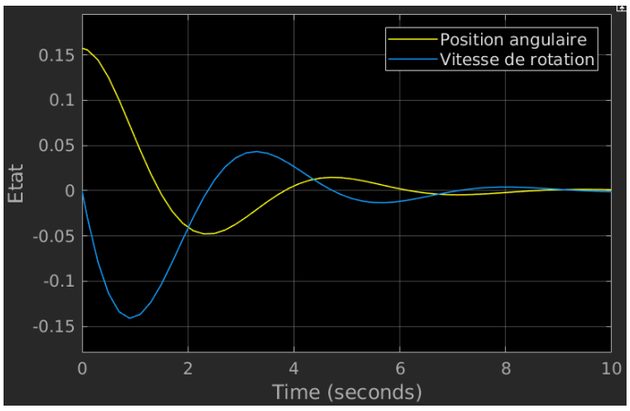
Figure 4: *Schéma bloc du contrôleur*

Cas	x_0	t_f	K	Intégrateur
Cas 1	$(\pi/20, 0)$	10	(30,10)	*ode*45
Cas 2	$(\pi/20, 0)$	100	(10,1)	*ode*45
Cas 3	$(\pi/20, 0)$	100	(10,1)	Euler, *ode*1
Cas 4	$(\pi/20, 0)$	1000	(10,1)	Euler, *ode*1
Cas 5	$(\pi/10, 0)$	100	(10,1)	*ode*45
Cas 6	$(\pi/10, 0)$	100	(30,10)	*ode*45

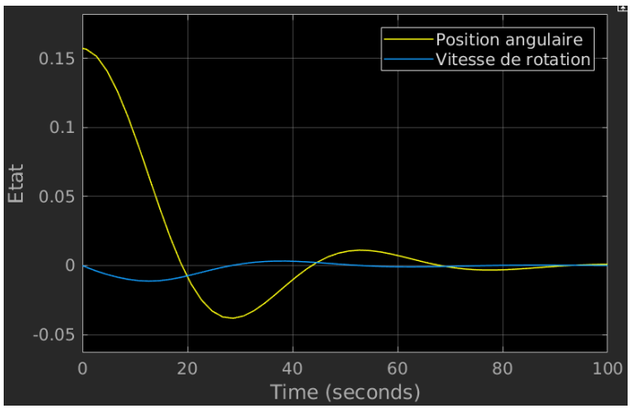
Table 1: *Données pour différents cas d'étude du contrôle par retour d'état*

Commentons les résultats des simulations obtenues pour ces cas d'étude.

Prenons les deux premiers cas dont les courbes sont représentées en figure 5. Les conditions pour la stabilisation sont vraies aux alentours du point de fonctionnement, or la valeur de x_0 est bien proche de $x_e = (0, 0)$. De plus, les conditions de l'équation (8) sont vérifiées dans ces deux cas. On a bien un système qui est stabilisé. Pour le cas 2, on diminue le gain K du contrôle, ceci prolonge le temps de réponse du système puisque le contrôle a moins d'impact.



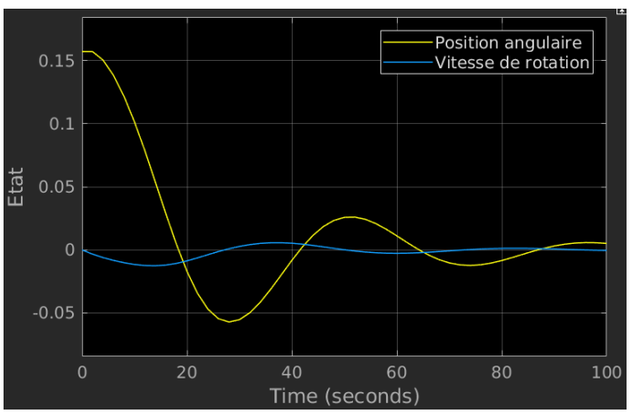
(a) Cas 1



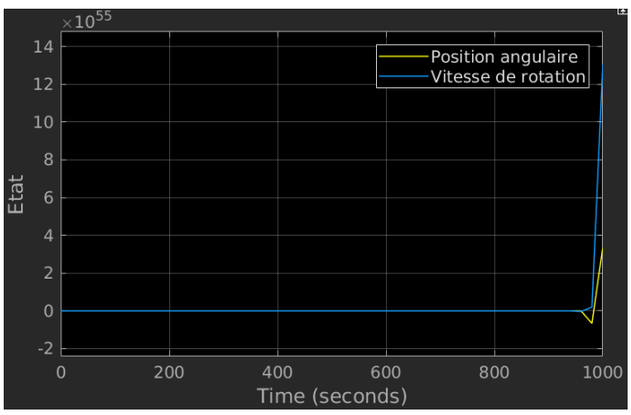
(b) Cas 2

Figure 5: *Comparaison du changement de gain*

Ensuite, pour les cas 3 et 4 (figure 6), on change d'algorithme d'intégration. Le cas 3 montre une perte de précision, les oscillations sont plus larges qu'au cas 2, mais restent stabilisées. En revanche, pour le cas 4, on augmente seulement le temps de simulation et la courbe de l'état devient absurde. En effet, l'intégrateur de l'algorithme d'Euler *ode*1 utilise un pas qui dépend de la durée de simulation. Ici, le pas est trop grand et provoque une simulation incorrecte.



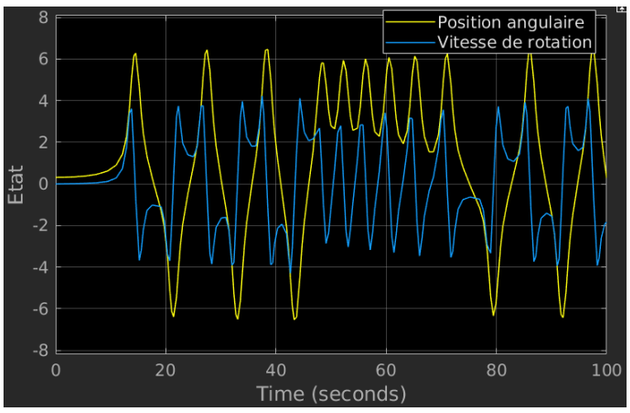
(a) Cas 3



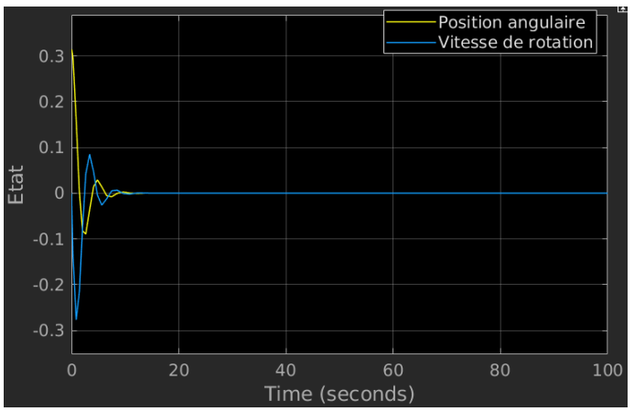
(b) Cas 4

Figure 6: *Comparaison du changement d'intégrateur*

Enfin, on repasse sur le premier intégrateur. Les cas 5 et 6 (figure 7) ont été obtenus en modifiant l'état initial x_0 que l'on a pris plus loin du point de fonctionnement, on a choisi un angle initial plus grand $\pi/10$ au lieu de $\pi/20$. En gardant un gain faible K, on ne stabilise plus le système, l'état diverge au cas 5. Cependant, si on augmente à nouveau K à la valeur du cas 1, on peut re-stabiliser le système (cas 6) puisqu'on donne plus d'importance au contrôle.



(a) Cas 5



(b) Cas 6

Figure 7: *Comparaison du changement d'état initial*

1.2 Contrôle par retour de sortie avec capteur

Nous avons pu contrôler correctement le système à l'aide des variables de sorties de position angulaire et de vitesse, cependant, sur un système réel, on a pas toujours accès à toutes les variables de sorties. On va donc maintenant supposer que l'on a accès qu'à la valeur $\dot{\alpha}$ à l'aide d'un capteur sur le système. Il faut alors reconstruire α à l'aide d'un prédicteur. Pour cela, on utilise un intégrateur continu.

Le schéma bloc réalisé sur Simulink en séance de TP reste le même pour le bloc du système simple (figure 3) et le bloc du contrôleur (figure 4), on ajoute simplement le capteur et le prédicteur sur la figure 8. Les blocs du prédicteur et du capteur sont représentés respectivement sur les figures 9 et 10. Nous étudierons les simulations avec les différents cas du tableau 2.

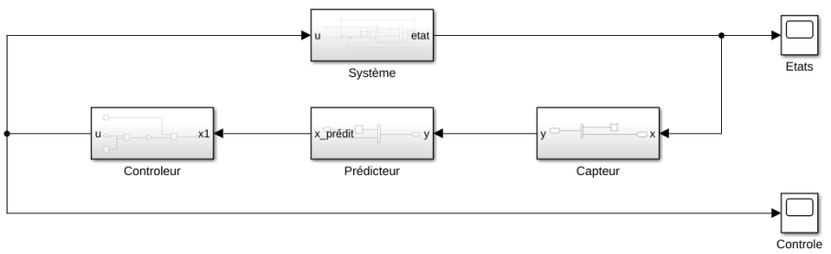


Figure 8: *Schéma bloc du système contrôlé par retour de sortie*

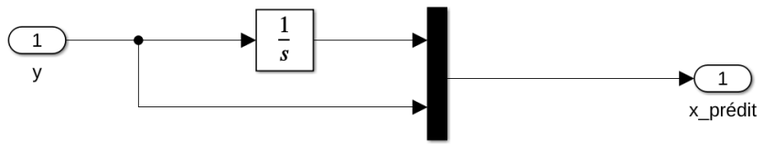


Figure 9: *Schéma bloc du prédicteur*

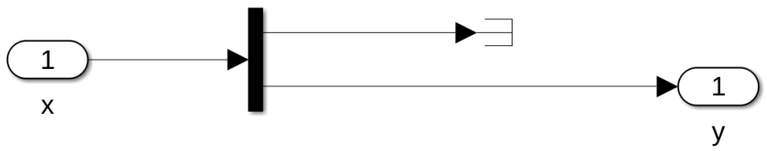


Figure 10: *Schéma bloc du capteur*

Cas	x_0	t_f	K	Pas	Intégrateur
Cas 1	$(\pi/20, 0)$	100	(10,1)	par défaut	*ode*45
Cas 2	$(\pi/20, 0)$	100	(10,1)	0.001	Euler, *ode*1
Cas 3	$(\pi/20, 0)$	100	(10,1)	5	Euler, *ode*1

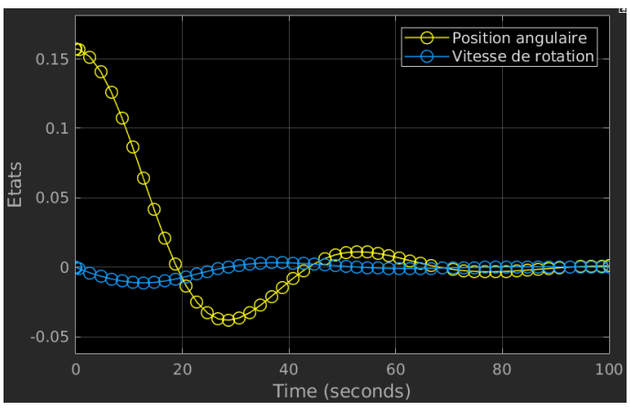
Table 2: *Données pour différents cas d'étude du contrôle par retour de sortie*

Nous pouvons observer sur les trois courbes (figure 11) des points indiquant le pas de l'intégrateur choisi.

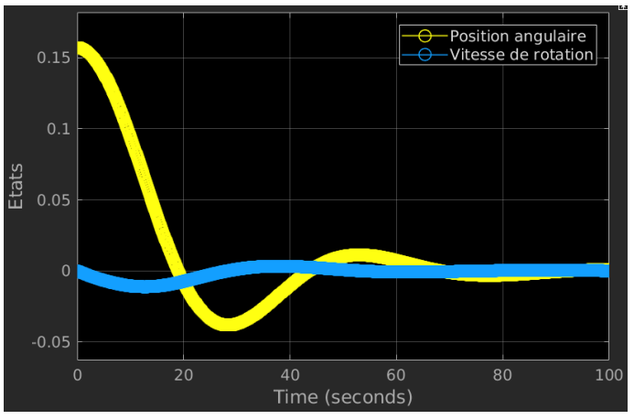
Pour le premier cas, nous avons utilisé l'intégrateur plus performant *ode*45 qui ne nécessite pas de préciser le pas qui est fixé par défaut. De plus, la courbe est similaire à celle du cas 2 de la figure 5, notre contrôle donc fonctionne aussi bien lorsque le signal est reconstitué au travers d'un capteur pour cet intégrateur.

Le second cas utilise l'intégrateur d'Euler, *ode*1 avec un pas fixé à 0.001 contrairement à la figure 6 où le pas était automatique. Ici, le pas choisi est suffisamment petit pour avoir un modèle correct qui donne un résultat similaire au cas 1. Cependant, le premier intégrateur est très certainement moins lourd et plus rapide que celui-ci vu le nombre de points calculé au cas 2.

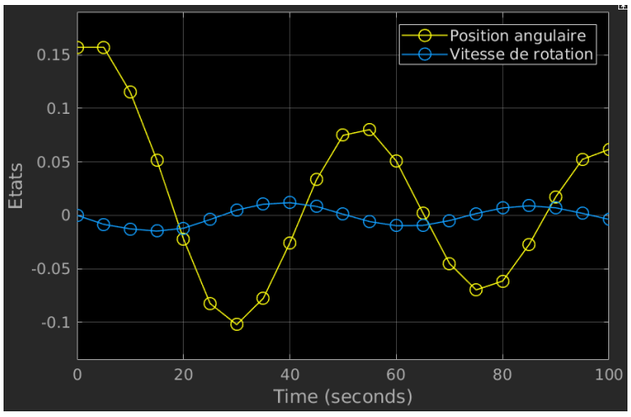
Quant au dernier cas, on a pris un pas de 5 bien trop grand pour l'intégrateur d'Euler *ode*1 ce qui entraîne une mauvaise simulation avec un temps de réponse plus grand mais on a toujours une stabilisation.



(a) Cas 1



(b) Cas 2



(c) Cas 3

Figure 11: *Comparaison de l'algorithme d'intégration utilisé*

2 Modèle du robot Lego

Après avoir étudié un modèle simple de pendule inverse, tentons de modéliser le système plus compelxe du robot Lego NXT qui s'appuie sur celui du pendule inverse.

2.1 Contrôle par retour d'état

Nous allons à nouveau étudier le modèle avec contrôle par retour d'état. Cette fois les équations de la dynamique sont plus complexes car elles font intervenir l'inertie du robot et deux autres variables d'état supplémentaires : la position angulaire des roues θ et leur vitesse angulaire $\dot{\theta}$. On a donc $x(t) = (x_1(t), x_2(t), x_3(t), x_4(t))^\intercal = (\theta(t), \alpha(t), \dot{\theta}(t), \dot{\alpha}(t))^\intercal$ et $x_e = (0, 0, 0, 0)^\intercal$.

2.1.1 Analyse du modèle théorique du robot

L'équation d'état reste la même que dans 1.1.1 (voir (2)) mais f est définie différemment. Nous n'irons par dans les détails pour rester dans le cadre de notre étude.
Comme pour la partie 1.1.1, on boucle le système avec le contrôle $u(t)$ (voir (4)) et $K \in \mathbb{R}$. Puis, on trouve la matrice jacobienne au point de fonctionnement sous la forme

$$J_g(x_e) = A + BK \tag{9}$$

où $A \in \mathbb{M}_4$ et $B \in \mathbb{R}^4$.

Finalement, la stabilisation du système va dépendre des valeurs propres de $J_g(x_e)$, on aimerait alors trouver K tel que ces valeurs propres soient strictement négatives. On prendra les valeurs propres contenues dans $V = (-136.5905, -2.6555, -3.5026, -5.9946)$.

2.1.2 Simulation du modèle sur Simulink

Nous pouvons maintenant effectuer des simulations du robot Lego sur Simulink. La figure 12 représente le schéma bloc du système contrôlé par retour d'état, le bloc de contrôle étant le même que celui de la figure 4. La table 3 référencie les différents cas étudiés lors des simulations.

Les équations de la dynamique sont ici décrites par une fonction matlab fournie pour simuler le système. De plus, on trouve K en calculant les matrices A et B puis en utilisant la fonction $place(A, B, V)$ qui retourne $-K$ dans un fichier matlab à part. L'algorithme d'intégration utilisé est $ode45$.

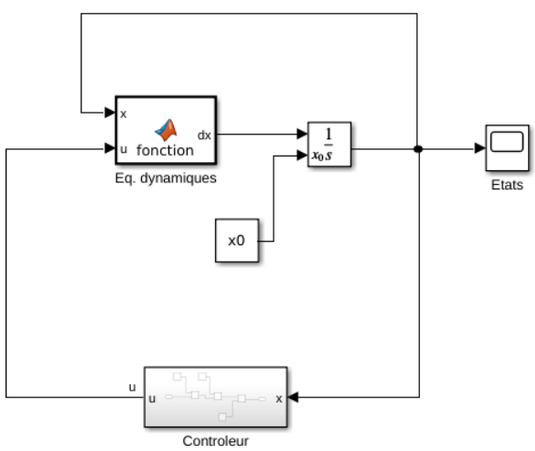
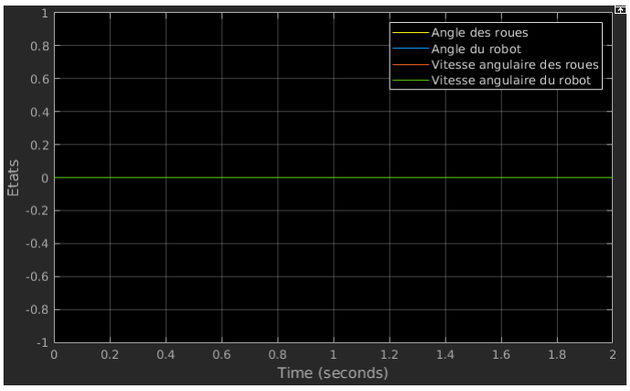


Figure 12: *Schéma bloc du système contrôlé du robot Lego*

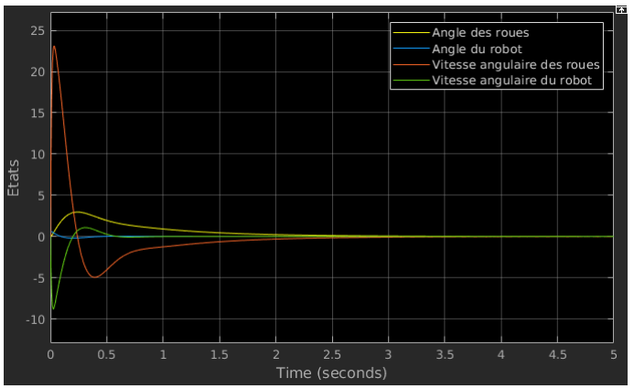
Cas	x_0	t_f
Cas 1	$(0, 0, 0, 0)$	2
Cas 2	$(0, \pi/5, 0, 0)$	5
Cas 3	$(0, \pi/10, 0, 0)$	5

Table 3: *Données pour différents cas d'étude du contrôle par retour d'état*

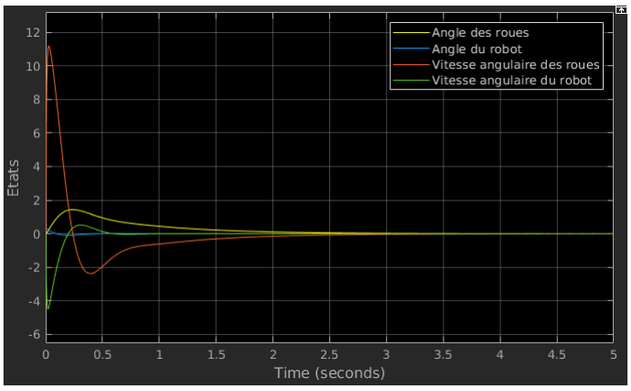
Les résultats sont représentés sur la figure 13. On remarque pour le cas 1 que l'état reste constant nul pour un état initial nul, le système ne dégénère donc pas au point de fonctionnement. Pour des états initiaux non nuls (cas 2 et 3), on visualise bien une stabilisation du système contrôlé. Enfin, on remarque que pour une position angulaire initiale plus faible (cas 3), les oscillations sont moins fortes, ce qui est normal puisque le système démarre plus proche du point de fonctionnement.



(a) Cas 1



(b) Cas 2



(c) Cas 3

Figure 13: *Résultats du contrôle par retour d'état du robot Lego pour différents états initiaux*

2.2 Contrôle par retour de sortie avec capteur

Dans cette partie nous allons encore une fois effectuer un contrôle par retour de sortie en considérant que l'on a accès qu'aux valeurs de la vitesse angulaire $\dot{\alpha} = x_4$ du robot et de la position angulaire des roues $\theta = x_1$ à l'aide de capteurs. Encore une fois nous avons besoin d'un prédicteur pour reconstruire l'information des deux autres composantes manquantes.

Le schéma bloc réalisé en figure 14 est similaire à celui de la figure 12 mais on y ajoute un bloc pour le capteur (figure 15) et pour le prédicteur (figure 16).

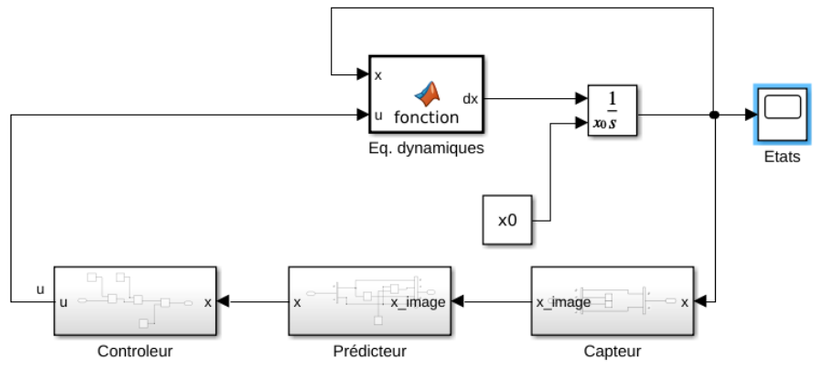


Figure 14: *Schéma bloc du système contrôlé par retour de sortie du robot Lego*

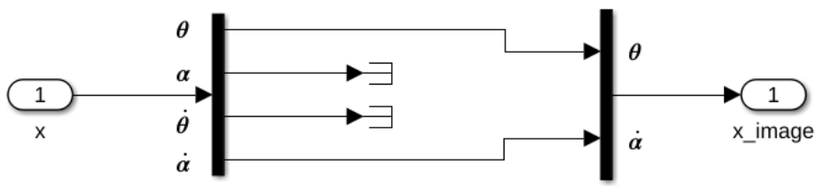


Figure 15: *Schéma bloc du capteur*

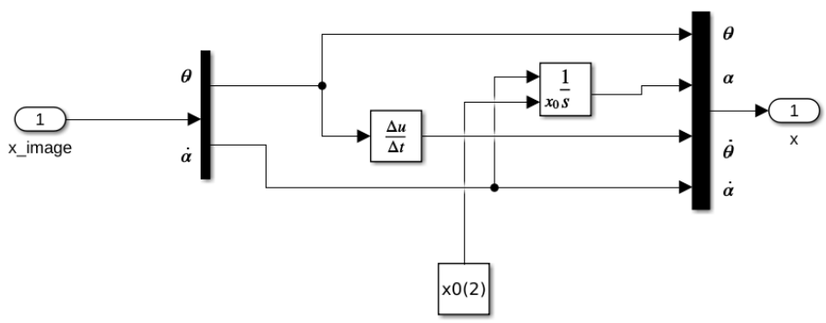
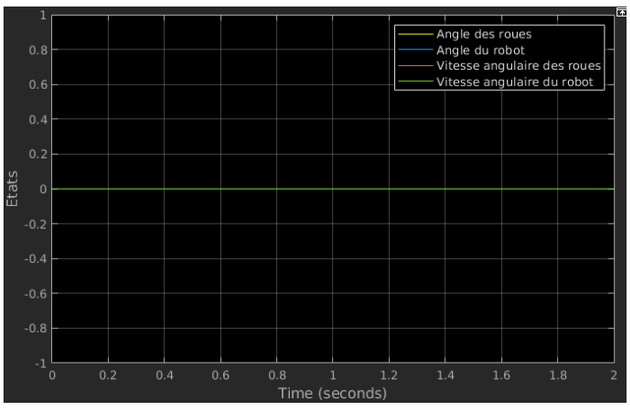
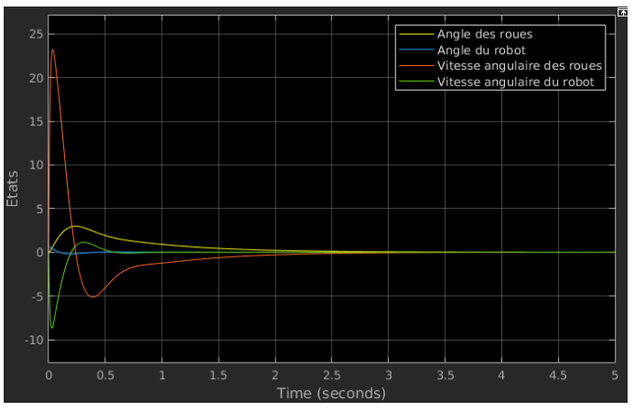


Figure 16: *Schéma bloc du prédicteur*

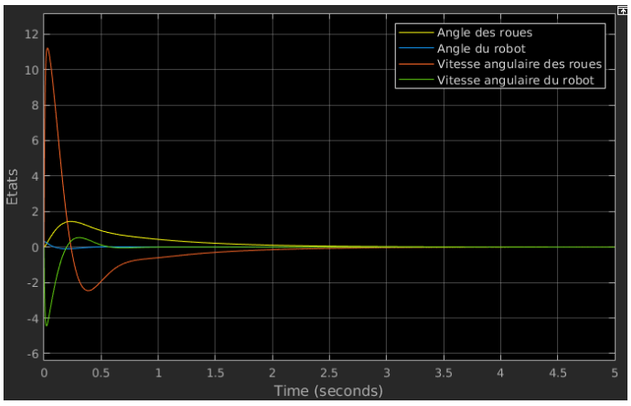
Pour la simulation, on étudie les mêmes cas que dans la partie 2.1.2 qui sont donnés en table 3. Les résultats sont donnés en figure 17. Les courbes obtenues semblent être les mêmes que celles de la figure 13 pour le contrôle par retour d'état, ce qui est normal en utilisant l'intégrateur performant *ode*45. Les conclusions sont donc les mêmes. Avec un bon intégrateur, on peut toujours stabiliser aussi bien le système avec ou sans capteur.



(a) Cas 1



(b) Cas 2



(c) Cas 3

Figure 17: *Résultats du contrôle par retour de sortie du robot Lego pour différents états initiaux*

2.3 Modèle hybride par discrétisation

Enfin, il reste un aspect que l'on n'a pas traité pour le pendule inverse et qui est important pour un système réel, c'est la prise en compte de signaux discrets. En effet, le logiciel implanté dans le robot, qui calcul le contrôle et la prédiction, fonctionne avec un signal discret tandis que le système représentant la physique est continu. On change donc les blocs du capteur et du prédicteur sur Simulink pour simuler un signal discret (voir figures 18 et 19). Le reste du système reste le même que précédemment (figure 14).

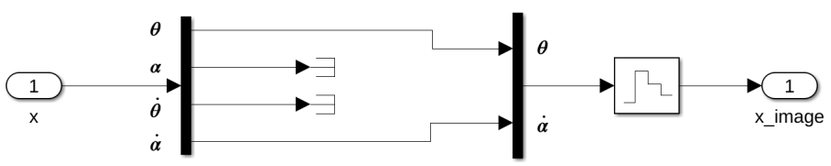


Figure 18: *Schéma bloc du capteur avec signal discret*

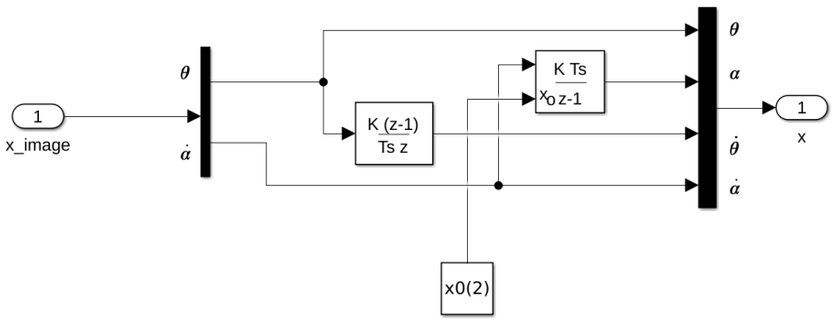
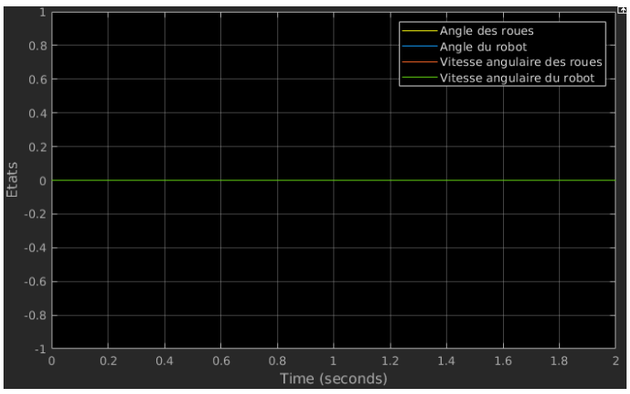
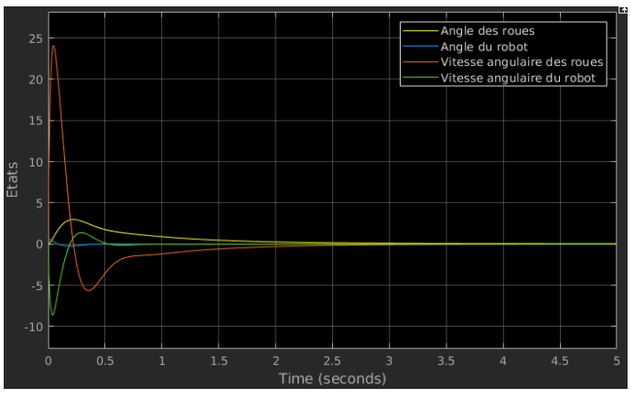


Figure 19: *Schéma bloc du prédicteur avec signal discret*

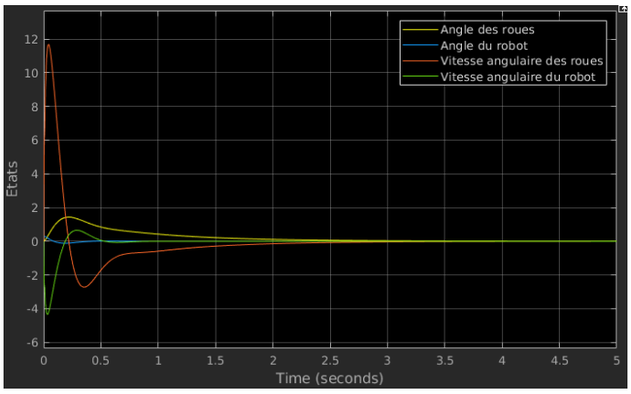
Les cas d'études de la simulation sont encore les mêmes (table 3) mais on pense bien à fixer le pas de la simulation à 0.005, sans quoi la discrétisation trop forte produit une simulation peu concluante. Les résultats sont représentés figure 20.
Finalement, les courbes sont encore similaires aux deux parties précédentes 2.2 et 2.1, les oscillations sont seulement un peu plus amples. Donc, avec un bon intégrateur et une discrétisation qui a un pas suffisamment faible, on peut avoir un modèle très correct.



(a) Cas 1



(b) Cas 2



(c) Cas 3

Figure 20: *Résultats du contrôle par retour de sortie du robot Lego pour différents états initiaux avec discrétisation*

3 Robot Lego NXT

Au final, on a pu modéliser correctement le modèle du robot Lego NXT sur Simulink et vérifier qu'il fonctionne déjà dans ce cadre là. Maintenant, il n'y a plus qu'à l'implanter dans le logiciel qui sera utilisé dans le robot. On écrit ce logiciel en C.

Plusieurs programmes sont déjà mis à notre disposition. Le logiciel se sépare en deux tâches, l'une va afficher les valeurs de α, θ et de l'état x. La deuxième, celle qui nous intéresse, va permettre de maintenir la position verticale du robot. Elle se décompose en plusieurs modes. Le premier mode INIT_MODE, initialise les variables du système. Le second mode CAL_MODE, appelé après $4ms$, calibre le gyroscope à la verticale pendant $6s$ (il faut le tenir à la main). Enfin, le mode CONTROLE_MODE calcul le contrôle et l'état du système toutes les $4ms$, c'est le coeur du fonctionnement du système. On remarque alors l'intérêt d'avoir simulé notre modèle avec discrétisation dans la partie 2.3 en prenant un pas de simulation de $5ms$.

En séance de TP, nous avons écrit ce dernier mode. Il faut déjà écrire la fonction *estimateur* qui calcul l'état du système $x = (\theta, \alpha, \dot{\theta}, \dot{\alpha})$ à partir de l'état observé $y = (\theta_m, \dot{\alpha})$ et du pas temporel (voir Listing 1).

```
void estimateur(float dt){
  float x_before = x[0];
  x[1] = x[1] + dt*y[1];
  x[3] = y[1];
  x[0] = y[0] + x[1];
  x[2] = (x[0]-x_before)/dt;
}
```

Listing 1: Fonction estimateur

Ensuite, nous avons écrit la fonction *controleur* qui calcul la commande u à appliquer au système en fonction de l'état courant du système (voir Listing 2).

```
void controleur(){
  float somme = ue;
  float K[4] = {0.6700, 19.9055, 1.0747, 1.9614};
  for (int i=0; i<=3; i++){
    somme += K[i]*(x[i]-xe[i]);
  }
  u = somme;
}
```

Listing 2: Fonction controleur

Puis, nous avons complété le mode CONTROLE_MODE à l'aide de ces fonctions (voir Listing 3)

```
case(CONTROL_MODE):
  dt = delta_t();
  y[0] = getMotorAngle();
  y[1] = getGyro(gyro_offset);
  estimateur(dt);
  controleur();
  nxt_motors_set_command(u);
  break;
```

Listing 3: CONTROLE_MODE

Le logiciel est alors fini, il ne reste qu'à le compiler et le télécharger dans le robot Lego NXT.

Conclusion

Pour conclure ce projet, nous avons d'abord pu construire un modèle mathématique du système et le valider par simulation de plusieurs cas de figures avec différents paramètres.

En premier, nous sommes passés par un système simple de pendule inversé qui modélise globalement le fonctionnement du robot. Cependant, pour se rapprocher de la réalité, nous avons introduit les vraies équations de la physique liées à la dynamique du robot en partie 2. Nous avons aussi choisi d'étudier des contrôles par retour d'état pour la théorie puis par retour de sortie avec des capteurs puisque le système réel fonctionne avec des capteurs. En effet, on a pas toujours accès à toutes les variables de sorties sur un système réel. Nous avons aussi choisi de simuler la discrétisation du système car l'implémentation du programme final utilise des variables discrètes.

Finalement, au fur et à mesure on a pu valider nos modèles et obtenir des contrôles performants pour aboutir à un logiciel fonctionnel.