Systèmes Multivariables: Etude de cas

Laleh Makarem, Alain Bock, Damien Perritaz, MER Denis Gillet

10 novembre et 22 décembre 2010
Salle ME A0 392

Points importants

Une étude de cas vous est proposée pour illustrer le cours de Systèmes Multivariables. Cette étude de cas se déroule en deux parties pratiques et vous permet d'obtenir un bonus maximum de 0.5 pt rajouté à la note de l'examen.

Un rapport (fichier pdf) et le code (un zip contenant tous les fichiers Matlab nécessaires à l'exécution des deux parties) doivent être rendus au plus tard pour le vendredi **7 janvier 2010**, par email à *alain.bock@epfl.ch*. Vous pouvez travailler seul ou par deux. Si vous travaillez par deux, un seul rapport contenant explicitement les deux noms sera rendu. Le rapport doit au minimum répondre aux questions 2.4, 3.8, 4.6, 5.5, 5.8, 6.1, 6.2, 6.3 et 6.4 ci-dessous.

Introduction

Au printemps 2007, le Laboratoire d'Automatique a commencé à étudier la comande d'un unicycle entièrement autonome. Plusieurs projets de semestre et de Master ont ainsi vu le jour. Après avoir modélisé le système et étudié ses caractéristiques (gouvernabilité, observabilité, etc.), nous en sommes maintenant à la construction d'un premier prototype. De futurs projets d'étudiants permettront d'avoir un démonstrateur illustrant les possibilités de l'automatique.

Dans le cadre de l'étude de cas relative au cours de Systèmes Multivariables, nous vous proposons d'étudier une problématique similaire. L'accent de ce cours n'étant pas la modélisation, le modèle complet vous est donné. De plus, nous allons commander un système dont les équations sont sensiblement plus simples : un Segway (Fig. 1) qui se déplace en ligne droite. Celui-ci ne peut en effet pas tomber de côté.

Objectifs

L'objectif de l'ensemble de l'étude de cas constitué de deux parties est de commander le Segway autour de l'origine et le stabiliser dans sa position verticale. Dans ce contexte, la première partie se concentre sur la **linéarisation** du modèle, puis sa **discrétisation** et enfin la **validation** en simulation. Dans la deuxième partie, vous allez faire de la commande en boucle fermée à l'aide d'une contre-réaction d'état discrète déterminée au moyen de la formule d'**Ackermann** ainsi que d'une **approche optimale LQR**.

FIG. 1 – Le système pris en considération est un Segway.

Fichiers à disposition

Pour la première partie, les fichiers Matlab/Simulink suivants sont à modifier :
- *donneePartie1.m* est le corps de la première partie de l'étude de cas. Vous y trouverez la structure des questions et exercices ;
- *conditionsInitiales.m* pour modifier les conditions initiales pour les simulations ;
- *simulationLinearisation.mdl* pour simuler les modèles non linéaire et linéarisé ;
- *simulationDiscretisation.mdl* pour simuler les modèles continu et discrétisé.

Pour la deuxième partie, les fichiers Matlab/Simulink suivants sont à modifier :
- *donneePartie2.m* est le corps de la deuxième partie de l'étude de cas ;
- *conditionsInitiales.m* pour modifier les conditions initiales pour les simulations ;
- *simulationAckermann.mdl* pour simuler la commande par Ackermann ;
- *simulationLqr.mdl* pour simuler la commande LQR.

D'autres fichiers sont nécessaires au bon fonctionnement du programme :
- *modeleNL.m* vous donne directement le modèle non linéaire du système ;
- *parametresPhysiques.m* contient les valeurs des paramètres physiques du système ;
- *systemeNL.m* pour simuler le modèle non linéaire sous Simulink ;
- *affichage.m* pour visualiser le comportement du système en 2D ;
- *modeles.mat* (pour la deuxième partie) pour charger les valeurs des modèles.

Pour le bon fonctionnement de ces fichiers il est nécessaire de disposer de Matlab, Simulink et de la Control System Toolbox.

Pour les fichiers Matlab (*.m), les lignes qui se terminent par `%-!-%` doivent simplement être décommentées. Les lignes qui se terminent par `%-?-%` doivent être décommentées et complétées. Les suites de symboles `%-!!!-%` et `%-???-%` indiquent respectivement une section entière à décommenter et une section entière à décommenter et compléter.

Les fichiers Simulink (*.mdl) vous permettent de simuler le comportement des différents modèles en boucle ouverte, ainsi que le comportement du système réel auquel on applique différents régulateurs. La plupart des composants seront déjà disposés dans l'interface graphique et n'auront plus qu'à être connectés, alors que d'autres devront être ajoutés et configurés.

Remarque Nous considérons que l'utilisation d'un logiciel tel que Matlab est quelque chose d'acquis de la part de tous les étudiants. L'objectif ici n'est pas d'apprendre à utiliser un logiciel mais de s'en servir pour résoudre des problèmes typiques. L'étudiant qui ne serait pas encore à l'aise avec Matlab est vivement encouragé à se familiariser au préalable. Sur la page Web du cours se trouve un document tutoriel. La page internet suivante peut également vous être très utile : www.mathworks.com/products/matlab/demos.html (en anglais), en particulier ce qui concerne l'exécution du programme par cellules.

Modèle

Le point de départ de cette étude de cas est le modèle d'état non linéaire du Segway, donné sous la forme $\dot{x} = f(x, u)$, où x correspond au vecteur d'état $\begin{pmatrix} \xi & \theta & \dot{\xi} & \dot{\theta} \end{pmatrix}^T$, et u à l'entrée appliquée au moteur (Fig. 2). Le modèle explicite où tous les états sont mesurés est :

$$
\begin{aligned}
\dot{x} &= \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} x_3 \\ x_4 \\ \frac{2\big(\big(m_p d^2+J_p\big)\big(dR^2\sin(x_2)m_p x_4^2+R\eta x_4+Ru-\eta x_3\big)-dR\cos(x_2)m_p(dgR\sin(x_2)m_p+\eta(x_3-Rx_4))\big)}{-R^2\cos(2x_2)m_p^2d^2+m_p((m_p+4m_r)R^2+4J_r)d^2+2J_p((m_p+2m_r)R^2+2J_r)} \\ \frac{2\big((m_p+2m_r)R^2+2J_r\big)(dgR\sin(x_2)m_p+\eta(x_3-Rx_4))-2dR\cos(x_2)m_p\big(dR^2\sin(x_2)m_p x_4^2+R\eta x_4+Ru-\eta x_3\big)}{R\big(-R^2\cos(2x_2)m_p^2d^2+m_p((m_p+4m_r)R^2+4J_r)d^2+2J_p((m_p+2m_r)R^2+2J_r)\big)} \end{pmatrix} \\
y &= x
\end{aligned}
$$

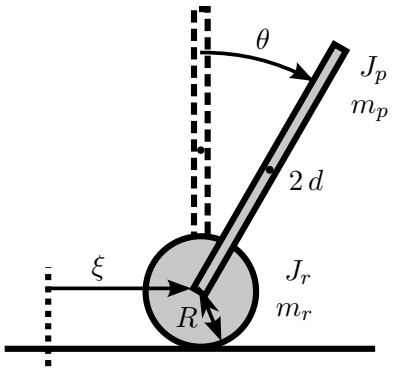

FIG. 2 – Le Segway à stabiliser en position verticale et à ramener à l'origine.

Première partie

1) Linéarisation du modèle (~ 25 *min*)

But: Trouver un modèle linéarisé autour d'un point de fonctionnement pour pouvoir appliquer les techniques de commande vues au cours.

La première étape, pour pouvoir utiliser les outils développés dans le cours, consiste à linéariser la dynamique $\dot{x} = f(x,u)$ pour retomber sur un schéma connu du type : $\dot{\tilde{x}} = A\,\tilde{x} + B\,\tilde{u}$. Nous allons linéariser autour du point d'équilibre instable $\overline{x} = 0$. Rappel : $\tilde{x} = x - \overline{x}$; idem pour y et u.

1. Utilisez la fonction `jacobian` de Matlab pour trouver $A(x,u)$ et $B(x,u)$;
2. Déterminez intuitivement (sans calcul) l'entrée $\overline{u}$ au point d'équilibre ;
3. Evaluez numériquement les matrices A et B au point de fonctionnement $(\overline{x},\overline{u})$. Pour ce faire, utilisez la fonction `eval` de Matlab. Déterminez les matrices C et D ayant les bonnes dimensions dans le cas où tous les états sont mesurés.

2) Simulation de la réponse homogène (~ 25 *min*)

But: Simuler le modèle non linéaire et le modèle linéarisé puis comparer les résultats.

Vous avez maintenant une représentation d'état continue non linéaire du modèle du Segway d'une part, ainsi que son approximation linéaire autour d'un point de fonctionnement d'autre part. Comparez les deux modèles en les intégrant numériquement avec Simulink.

1. Fixez les conditions initiales pour les 4 états dans le fichier *conditionsInitiales.m* ;
2. Simulez le comportement du système réel (modèle non linéaire) pour une entrée nulle dans le fichier *simulationLinearisation.mdl* de Simulink. Ajoutez les éléments nécessaires (View : Library Browser) pour simuler le modèle linéarisé en parallèle ;

3. Observez l'évolution de la représentation en 2D des deux modèles ;
4. Discutez des différences entre les deux modèles pour valider la linéarisation, pour différentes conditions initiales et différentes entrées u.

Remarques Les paramètres de simulation, comme la durée de simulation, peuvent être modifiés (Simulation : Configuration Parameters). L'intégration numérique se fait implicitement avec la fonction `ode23` qui est basée sur Runge-Kutta.

3) Discrétisation du système linéarisé ($\sim 40\ min$)

But: Obtenir un modèle linéaire discrétisé pour appliquer les méthodes de commande d'état.

Nous voulons à présent obtenir les matrices Φ (`Phi`) et Γ (`Gam`).

1. Prenez une période d'échantillonnage de $1\,ms$;
2. En vous basant sur le polycopié, calculez la matrice Ψ (`Psi`) ;
3. Calculez les matrices Φ et Γ à l'aide de Ψ ;
4. Déterminez une représentation d'état continue (`Mc`) avec la fonction `ss` (State Space) de Matlab ;
5. Discrétisez le modèle d'état continu avec la fonction `c2d` (Continuous to Discrete) de Matlab pour obtenir un nouveau modèle (`Md`) et comparez les deux résultats ;
6. Simulez le comportement des modèles continu et discret pour une entrée nulle dans le fichier *simulationDiscretisation.mdl*. Pour simplifier, travaillez en variables écart en omettant $\overline{x}$ et $\overline{u}$. N'oubliez pas de donner la période d'échantillonnage au modèle discret pour qu'il ne soit pas intégré de manière continue ;
7. Utilisez ensuite la partie du code qui vous est fournie pour sauvegarder différentes valeurs dans le fichier *modeles.mat* pour la suite ;
8. Discutez les différences entre les deux modèles pour valider la discrétisation.

Deuxième partie

4) Commande d'état par Ackermann (~ 40 *min*)

But: Concevoir une commande d'état discrète par placement des valeurs propres.

1. Calculez la matrice de gouvernabilité G puis comparez le résultat obtenu avec ce que donne la fonction `ctrb` de Matlab. Verifiez si le système est gouvernable ;
2. En choisissant des valeurs propres en $(0.95, 0.999, 0.99, 0.99)$ déterminez la matrice de gain K à l'aide des relations vues au cours (p. 92 du polycopié). Utilisez `poly` et `polyvalm` ;
3. Comparez ensuite votre K avec le résultat obtenu par la fonction `acker` de Matlab ;
4. Considérez la réponse du système aux conditions initiales $x_0 = \begin{pmatrix} \xi & \theta & \dot{\xi} & \dot{\theta} \end{pmatrix}_0^T = \begin{pmatrix} 0.2 & 0.4 & 0 & 0 \end{pmatrix}^T$ en modifiant le fichier *conditionInitiales.m*. Simulez le comportement du modèle non linéaire du Segway en boucle fermée (`KAcker`) dans le fichier *simulationAckermann.mdl* de Simulink. N'oubliez pas de donner la période d'échantillonnage au régulateur discret pour qu'il ne soit pas intégré de manière continue ;
5. Observez l'évolution de la représentation en 2D du système bouclé ;
6. Essayer d'autres valeurs propres et discutez leurs effets.

5) Commande d'état optimale (~ 40 *min*)

But: Synthétiser une commande LQR discrète.

On cherche à trouver une commande optimale (LQR) avec les matrices de pondérations Q_1 et Q_2. Comme l'équation de Riccati qu'il faut résoudre est stationnaire pour un horizon d'optimisation infini, une implantation récurrente va nous faire converger vers la solution réelle. On va donc chercher à obtenir la solution optimale à partir d'une solution prise au hasard.

1. Définissez les matrices de pondération Q_1 et Q_2. Pour commencer, choisissez des matrices identité de dimensions appropriées ;
2. Pour assurer la convergence de l'algorithme, initialisez $S = I$, où I est une matrice identité de dimension appropriée ;
3. Implémentez l'équation de Riccati sous forme récurrente. Définissez un nombre d'itérations suffisamment grand pour assurer une "bonne" convergence de l'algorithme. Pour vous convaincre de cette convergence, vous pouvez représenter graphiquement l'évolution des éléments de S comme expliqué au cours. Autre moyen : représenter l'évolution des valeurs singulières de S (cf. Algèbre Linéaire ; commande Matlab `svd(S)`) ;
4. Après avoir obtenu une approximation de S_∞, calculez la matrice de gain K_∞ ;

5. Comparez les matrices S_∞ et K_∞ obtenues avec celles qui sont obtenues avec la fonction `dlqr` de Matlab ;
6. Pour les mêmes conditions initiales que précédemment, simulez le comportement du modèle non linéaire du Segway en boucle fermée (`KDlqr`) dans le fichier *simulationLqr.mdl*. N'oubliez pas de donner la période d'échantillonnage au régulateur discret pour qu'il ne soit pas intégré de manière continue ;
7. Observez l'évolution de la représentation en 2D du système bouclé ;
8. Essayer d'autres matrices de pondérations et discutez leurs effets.

6) Validation ($\sim$ 10 *min*)

But: Valider les régulateurs pour différentes conditions.

Deux manières de synthétiser un régulateur pour notre système Segway ont été effectuées. Nous allons maintenant observer différents comportement de ces commandes afin de les comparer. A chaque question, il faudra simuler à nouveau le système en boucle fermée, avec les gains trouvés précédemment par Ackerman ou LQR.

1. Changez les conditions initiales dans le fichier *conditionsInitiales.m* et évaluez le comportement en boucle fermée ;
2. Dans les fichiers Simulink, ajoutez un élément de saturation de la commande à $[-10, +10]$ et discutez du résultat ;
3. Donnez les avantages et inconvénients des deux commandes, notamment concernant les performances et la simplicité de modifier le comportement dynamique ;
4. Discutez les modifications à apporter si seulement les états ξ et θ sont mesurés.

Quelques fonctions Matlab utiles

`help`	**la fonction d'aide** (taper : `help leNomDeLaFonction`)
`lookfor`	**pour trouver les fonctions relatives à un terme particulier**
`plot`	pour créer une figure à partir d'un ou deux vecteurs pour l'affichage
`factorial`	pour obtenir la fonction factorielle
`'`	transposée
`poly`	pour obtenir les coefficients d'un polynôme à partir de ses racines
`polyvalm`	pour calculer un polynôme matriciel à partir des coefficients
`length`	donne la longueur d'un vecteur
`size`	donne la dimension d'une matrice
`inv`	pour calculer l'inverse d'une matrice
`jacobian`	pour dériver symboliquement des expressions
`svd`	décomposition en valeurs singulières
`ss`	pour créer un objet correspondant au modèle d'état de Matlab
`c2d`	transforme un objet continu en objet discret
`ctrb`	calcule la matrice de gouvernabilité d'un système d'état
`obsv`	calcule la matrice d'observabilité d'un système d'état
`acker`	implante la formule d'Ackermann
`ode23`	pour intégrer numériquement des équations dynamiques continues
`syms`	pour déclarer des variables symboliques