FRTN05 Nonlinear Control and Servo Systems
Laboration 2 – Nonlinear Control
The Furuta Pendulum

Bo Lincoln, Dept. of Automatic Control

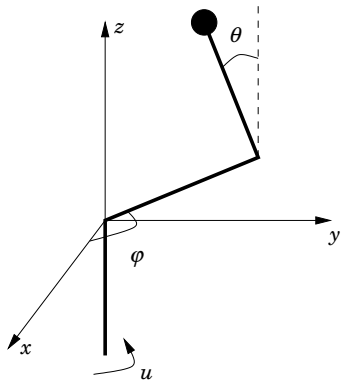

Figure 1 A schematic picture of the Furuta pendulum

1. Introduction

In this laboratory session we will design a nonlinear controller for an inverted pendulum on a rotating base (known as a "Furuta" pendulum). This is done by **Lyapunov design**, i.e. by inventing a Lyapunov function and making sure its derivative is negative by control. In the end of the lab there is also a short section on **friction compensation**, which makes a huge difference for our pendulum.

Important! There are 5 assignments in the lab. Number 2 and 3 have **home assignment** parts, which you will have to do before the lab.

The lab will be performed in Matlab Simulink, which you are now familiar with. We will both run pure simulations with a model of the pendulum in Simulink, and we will also run it with "hardware-in-the-loop". The latter means that we will replace the pendulum model with a special Simulink block which communicates with the real pendulum in real time, and actually run our algorithms on the real pendulum.

The Matlab files you need to copy are `pendlib.mdl`, `pendbasic.mdl`, `estimatefriction.mdl`, and `pendinit.m`. Start Matlab and run `pendinit`.

2. The Furuta Pendulum

The Furuta pendulum is a free pendulum on a rotating arm, see Figure 1. We can control the torque for rotating the arm, and if it is done right, we can balance the pendulum in the upright position. To be able to do this, we need a correct mathematical model of the nonlinear dynamics of the pendulum.

2.1 Nonlinear Model

The dynamics of the Furuta pendulum are rather complicated due to the rotation of the pendulum. Some Lagrange theory "simplifies" matters, and

after some algebra we end up with:

$$\begin{aligned}(J_p + Ml^2)(\ddot{\theta} - \dot{\varphi}^2 \sin\theta\cos\theta) + Mrl\ddot{\varphi}\cos\theta - gl(M + m/2)\sin\theta = 0 \\ Mrl\ddot{\theta}\cos\theta - Mrl\dot{\theta}^2\sin\theta + 2(J_p + ml^2)\dot{\theta}\dot{\varphi}\sin\theta\cos\theta \\ +(J + mr^2 + Mr^2 + (J_p + ml^2)\sin^2\theta)\ddot{\varphi} = u.\end{aligned} \tag{1}$$

To make it more readable, we introduce

$$\begin{aligned} a &= J_p + Ml^2 & b &= J + Mr^2 + mr^2 \\ c &= Mrl & d &= lg(M + m/2) \end{aligned} \tag{2}$$

and the equations of motion can be rewritten:

$$\begin{aligned} a\ddot{\theta} - a\dot{\varphi}^2\sin\theta\cos\theta + c\ddot{\varphi}\cos\theta - d\sin\theta = 0 \\ c\ddot{\theta}\cos\theta - c\dot{\theta}^2\sin\theta + 2a\dot{\theta}\dot{\varphi}\sin\theta\cos\theta + (b + a\sin^2\theta)\ddot{\varphi} = u, \end{aligned} \tag{3}$$

where θ is the angle of the pendulum, and φ is the angle of the arm. The control signal, u, is the motor torque on the arm.

Approximate coefficients for the pendulum used in the laboration are:

$$\begin{aligned} l &= 0.413\ m & r &= 0.235\ m \\ M &= 0.01\ kg & J &= 0.05\ kgm^2 \\ J_p &= 0.0009\ kgm^2 & m &= 0.02\ kg \\ g &= 9.8 \end{aligned}$$

2.2 Linearized Model

To be able to stabilize the pendulum in the upright position, we need a controller. The most commonly used method to design a controller is to linearize the nonlinear model and form a linear controller for this model.

To do this, we introduce the state

$$x = \begin{pmatrix} \theta & \dot{\theta} & \varphi & \dot{\varphi} \end{pmatrix}$$

and linearization of the system (3) around

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix},$$

which is the upright position of the pendulum with zero velocity, gives

$$\dot{x} = Ax + Bu = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{bd}{ab-c^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-cd}{ab-c^2} & 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ \frac{-cg}{ab-c^2} \\ 0 \\ \frac{ag}{ab-c^2} \end{pmatrix} u. \tag{4}$$

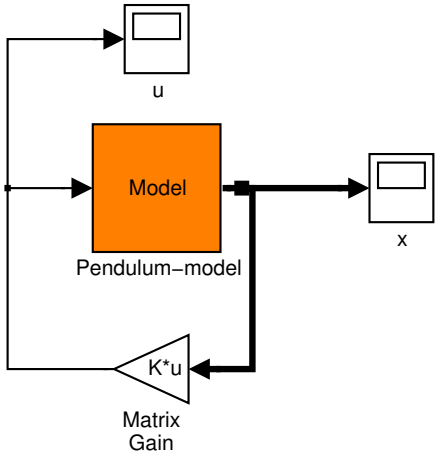

Figure 2 The set-up to simulate your linear controller.

ASSIGNMENT 1

a) Examine the linearized model (A,B) i.e. which eigenvalue corresponds to which state and where does the control action effect the system?

b) There are many ways to determine a linear control law for the system (pole placement , LQ-design, loopshaping etc). Show that the linear feedback

$$u = -Lx$$

with $L = \begin{pmatrix} -7.5343 & -1.3465 & 0 & -0.2216 \end{pmatrix}$ almost stabilizes the linear model. Which state is not stabilized? If there is time, try your own design.

c) Open Simulink and get the "Pendulum-model" block from `pendlib.mdl`. Insert this controller and simulate it (see Figure 2). Does it work? Find a region in the θ-$\dot{\theta}$-plane for which the linear controller is able to keep the pendulum in an upright position. (Initial states are found in the variable `x0`.) Why does it not work for all initial states?

□

3. Swinging Up the Pendulum

Now that we have a stabilizing controller for the upright position, we would like to design a controller which gets the pendulum close enough for our linear controller to work (a "swing-up"-controller). To be able to solve the problem, we will approximate the Furuta pendulum model by a planar pendulum (see Figure 3).

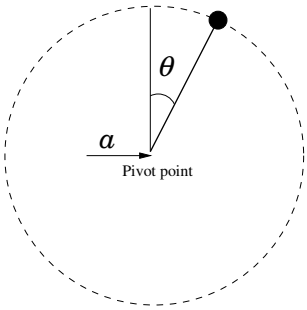

Figure 3 The planar pendulum, with only two states. The dashed circle is the possible "orbit" for the pendulum mass.

The equations of motion now simplify to

$$\ddot{\theta} = \frac{Mgl}{J_p}\sin\theta - \frac{Ml}{J_p}a\cos\theta, \tag{5}$$

where a is our control signal. It is the acceleration sideways of the pivot point (**Do not** mix a up with the constant in (2)). We can create this acceleration with our usual control signal u, i.e. the torque to the pendulum arm.
All parameters of the pendulum can actually be described by one parameter, which we call ω_0:

$$\omega_0 = \sqrt{\frac{Mgl}{J_p}}$$

and the equation simplifies to

$$\ddot{\theta} = \omega_0^2\sin\theta - \omega_0^2\frac{a}{g}\cos\theta$$

ASSIGNMENT 2
Home assignment: The name ω_0 suggests that it denotes a frequency. Show why, and suggest a way to estimate this parameter from experiments. *Hint:* Keep $a = 0$ and linearize around $\theta = \pi$ (down position).
At the lab: Do the experiment on the Pendulum-model in Simulink or on the real pendulum. Which value of ω_0 did you get?
(For the model to work be sure to use the Fixed-step ode5 solver with step size $h = 0.01$. The solver settings can be found in the Simulation Parameters dialog. Note that the θ-measurements are discontinuous at the downward position.) □

3.1 Energy Control

A very useful method to get the pendulum to the upright position is to control its *energy* to be the energy of the top position. The total energy

(potential energy + kinetic energy) of the pendulum is:

$$E = Mgl(\cos\theta - 1) + \frac{J_p}{2}\dot{\theta}^2$$

or normalized by ω_0^2

$$E_n = \cos\theta - 1 + \frac{1}{2\omega_0^2}\dot{\theta}^2$$

If the pendulum has energy $E_n = 0$, it will either be moving along the orbit in Figure 3 or be at the top position, with zero velocity (check this by inserting $\theta = 0$, $\dot{\theta} = 0$). This is due to the fact that no energy leaves the pendulum in our model. When the pendulum is close to the "up-position" it will have low speed, and it is very easy for the linear controller to "catch" it.

ASSIGNMENT 3
Home assignment: We want the energy $E_n \to 0$ both if the pendulum has too much or too little energy. Let $V(x,a) = \big(E_n(x,a)\big)^2$ be a Lyapunov function candidate. Find a simple controller $a = F(x)$ such that $\frac{d}{dt}V \leq 0$ for all x.
Hints:

- Differentiate and simplify V first, and see how you can choose a to make the derivative negative.
- Don't use control signals larger than 1.
- Make the controller non-linear!

At the lab: Simulate the controller with the Pendulum-model in Simulink. If it does not work properly, modify it until it does. □

4. The Full Monty

Now it is time to put it all together. We want the controller to have two modes:

1. Swinging up the pendulum (your controller).
2. Catching and balancing it in the top position.

In the Simulink model `pendbasic.mdl` a working top-position-controller is already given.

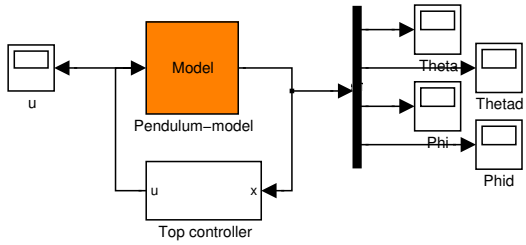

Figure 4 The Simulink setup with pendulum model and a top-position controller.

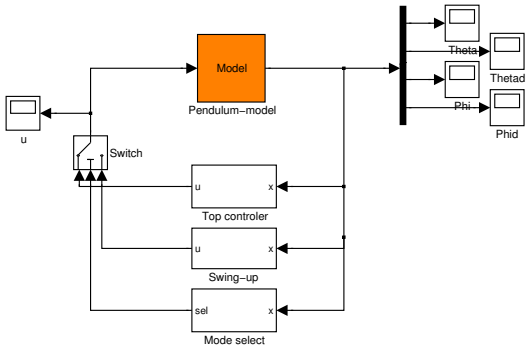

Figure 5 The full pendulum controller.

ASSIGNMENT 4
Construct (in Simulink) a switching method, which switches between your swing-up controller and the given top-position controller. See Figure 5 for a useful structure. *Hint:* For example, switch to top controller when θ and $\dot{\theta}$ are both "small", i.e. close to the top position with low velocity.
When it works with the Pendulum-model, go to the real pendulum and run your controller there. Modify it until it works. □

5. Friction Compensation

When the pendulum is in its upright position, it oscillates in φ (the arm position). This is due to friction which makes the arm stick although $u \neq 0$. A simple way of modeling the friction is shown in Figure 6.

A simple but efficient way of compensating for this is to estimate k off-line ($\hat{k}$) and give the control signal a extra "push" in the right direction depending on $\dot{\varphi}$ – see Figure 7.

In the Simulink model `estimatefriction.mdl` (see Figure 8) a simple PI-controller has been implemented to keep the arm velocity $\dot{\varphi}$ constant (with the pendulum hanging down).

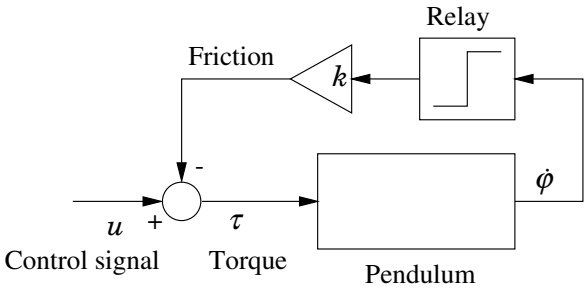

Figure 6 A model of the φ friction. The friction will work in the opposite direction of the velocity $\dot{\varphi}$. The constant k is the friction torque.

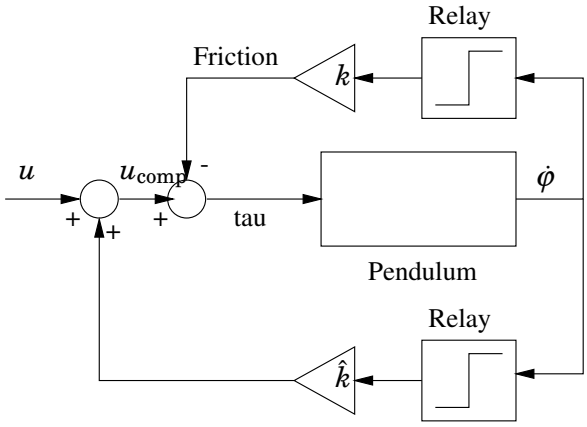

Figure 7 A simple friction compensation where an extra torque $\hat{k}$ is added in the right direction.

ASSIGNMENT 5
How can you use this to get an estimate $\hat{k}$ of the friction torque? Estimate it for the pendulum-model, and if there is time, on the real pendulum.
Insert the "Friction-compensation" block from `pendlib.mdl` into your controller. Insert the estimate $\hat{k}$ and simulate. Does it perform better in the top position?
If there is time, run it on the real pendulum too. □

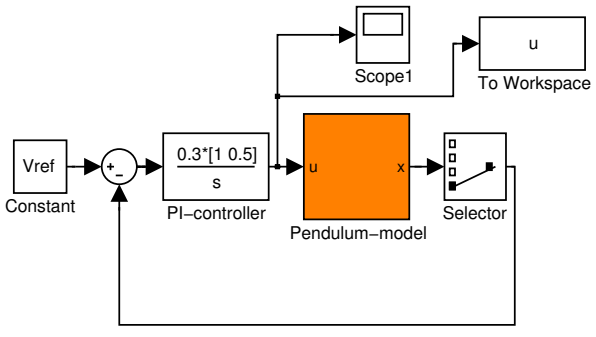

Figure 8 The Simulink setup to estimate friction.