The embedded system project

Midterm review

2003-04-10

Abstract

As part of a project course in automatic control, we have been given the task of controlling an inverted pendulum with an inertia wheel as actuator. The task is to make the pendulum swing up from the hanging down position to balancing in the inverted position. This is a non-linear system that had to be linearised. We made a model and designed both LQG and PID controllers that stabilised the pendulum. We also made a controller that can swing the pendulum up, but so far the system does not perform sufficiently.

Table of contents

1 Introduction 4
2 The project 4
2.1 Resources 4
2.2 Plan 4
2.3 Future work 5
3 Hardware 5
3.1 Description of the hardware 5
3.2 Getting to know the system 5
4 The model 6
4.1 Physical effects 6
4.1.1 Gravity 6
4.1.2 Inertia wheel 6
4.1.3 Mass moment of inertia 7
4.1.4 Electric characteristics of the motor 7
4.1.5 Friction 7
4.2 A non-linear state-space model 8
4.3 Linearising the model 8
4.4 Verifying the model 8
4.4.1 Verifying the motor dynamics 9
4.4.2 Verifying coupling between motor torque and pendulum swinging 10
5 Controller design 10
5.1 Balance controller 10
5.1.1 LQG controller 11
5.1.2 PID controller 12
5.2 Swing controller 13
5.3 Stop controller 13
5.4 Integrating controllers 13
5.5 Implementation of controllers in the DSP 14
5.5.1 Continuous vs. discrete representation 14
5.5.2 Differentiation 14
5.5.3 Implementing an LQG controller 14
5.5.4 Implementing a PID controller 14
6 Conclusions 14
7 References 15

1 Introduction

The project is part of the automatic control project course 2E1242, at KTH, and we are assigned to develop a control system for an inverted pendulum. This is a non-linear problem that has to be linearised. The control law is implemented in a DSP, so that the system can function autonomously without being connected to a computer. Our task is to design a controller that can swing the pendulum up from the hanging down position to the inverted position, and then balance it there. This report is a documentation of our progress, half way into the project.

2 The project

2.1 Resources

The project group consists of four people:

- Mikael Ek, project leader and responsible for documentation and web pages
- Michael Redmond, responsible for implementation in the DSP
- Magnus Lindhé, responsible for modeling
- Niklas Mattsson, responsible for automatic control design

The project also has an external resource, Alberto Speranzon, a consultant available one hour per week in five weeks.

2.2 Plan

The project started on the 17th of March 2003 and the final report has to be handed in before the 23rd of May 2003. The work is planned for seven weeks plus a three week break for Easter. So far we have been slightly ahead of the plan, the way we work has been a little different than planned. When we planned the project we thought we should design one thing and deliver it to the implementation group, then at the same time design the next feature and when that one is ready, deliver it. This resulted in the plan as seen in the table below.

Milestone	**Date**
System modelled	2003-03-21
Balancing controller designed	2003-03-28
Swing controller designed	2003-04-02
Balancing controller in DSP	2003-04-08
Controller design ready	2003-04-08
Swing controller in DSP	2003-04-11
Safety features designed	2003-05-06
Swing and balance integrated in DSP	2003-05-06
Final controller design ready	2003-05-14
Everything implemented	2003-05-16

Table 1. Milestones for the project

This plan was based on the assumption that it would be difficult to get to know the DSP and to implement the controller in it. Since this was easier than we thought, we are much ahead with the implementation, and we now work more in parallel. As soon as the design group comes up with something, we try it in the DSP for real. At the same time we continuously try to improve the control law in the DSP.

2.3 Future work

We will make improvements on the controllers so they work better together. We shall also implement some safety features. As a final task we will investigate what happens to the controller performance when we simplify the controllers, and lower the sampling frequency.

3 Hardware

3.1 Description of the hardware

The embedded system we use is a control kit from Mechatronic Systems Inc. It is based on a DSP development system from Texas Instruments and consists of the following parts:

- TMS320C6711 DSK board (a DSP board with parallel interface port)
- PWM/Optical Encoder data Acquisition board (a typical digital I/O board)
- PWM amplifier board
- 5V and 25V power supplies.
- 25V DC motor with 1000 counts/rev optical encoder
- Pendulum arm attached to a 1000 counts/rev optical encoder
- Inertia wheel
- A Matrix Digital LK202-25 LCD screen

There are two ways of controlling the system and those are either programming the DSP board flash memory or using the parallel port and a computer IDE, in our case Code Composer Studio. The controlling program in the DSP then uses predefined functions for the digital I/O daughter board to send a control signal to the PWM and receive data from the two optical encoders. Both the input and output are handled and translated on the daughter board. The PWM control signal goes from the daughter board to the PWM amplifier board and it is a translator between the 25V power supply and the 24V motor. The output voltage to the motor is based on the control signal from the daughter board.

3.2 Getting to know the system

Our first goal was to be able to get data out of the system to create and verify our models of the system. However we had neither of the recommended software for this venture and had to come up with our own method. We decided to use the serial port of the daughter board, used by the LCD screen, and connect it directly to the computer's serial port. For this we created our own serial cable and conveniently used the daughter board's predefined function for writing to the LCD screen. The data was received and saved by a program named CodeVisionAVR, by HP InfoTech. The data was then imported to Matlab. The second step was to collect the required data for model verification and we started to examine the daughter board's encoder and PWM functions closer. We tested the inertia torque reaction when starting the motor when standing still, and the step response for the angular velocity of the motor.

4 The model

We have decided on using the following notation for the mechanics of the pendulum:

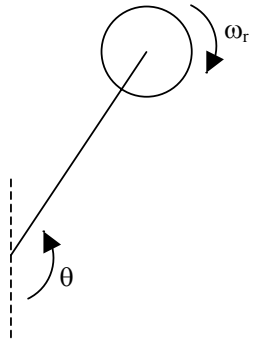

Figure 1. The notation for the pendulum

4.1 Physical effects

There are a number of physical effects that affect the movement of the pendulum. We have tried to identify the most important of these and quantify them:

4.1.1 Gravity

The mass of the pendulum arm, the motor and the inertia wheel combine to one single centre of mass, situated somewhere on the pendulum arm. It gives a gravitational torque (defined as positive in the positive θ-direction):

$$M_g = -mgl\sin\theta, \qquad (1)$$

where m is the combined mass, g is the gravitational acceleration and l is the distance from the pivot.

4.1.2 Inertia wheel

When the motor attempts to change the rotational velocity of the inertia wheel, this requires a torque which also affects the motor itself, in the opposite direction. The wheel has a mass moment of inertia, J_r:

$$M_{motor} = -M_r = -J_r\dot{\omega}_r \qquad (2)$$

The torque that acts on the motor can then be translated to a torque acting on the whole pendulum (defined as positive in the positive θ-direction) [1]:

$$\overline{M}_{pend} = \overline{M}_{motor} + \bar{r}\times\underbrace{\sum\overline{F}}_{=0} = \overline{M}_{motor} \qquad (3)$$

4.1.3 Mass moment of inertia

When torques are applied to the pendulum or inertia wheel, they change their angle and rotational velocity according to the torque equation:

$$M = J\ddot{\theta} = J\dot{\omega} \qquad (4)$$

The inertia wheel and rotating parts of the motor have one mass moment of inertia, J_r, and the whole pendulum has another, J_p.

4.1.4 Electric characteristics of the motor

The motor can be modelled as this equivalent circuit:

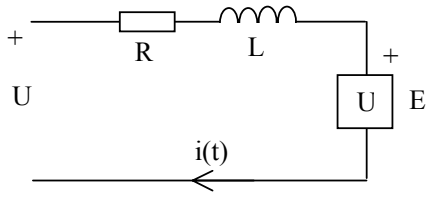

Figure 2. Equivalent circuit for the motor

The voltage E is the electromotorical force (EMF) that is proportional to the angular velocity of the motor. L is the inductance in the motor windings and R causes the active power that is converted into heat and mechanical power. According to the data sheet of the motor, the inductance is 6.27 mH [2]. Since we have a voltage of 24 V, it would take only 0.5 ms to change the current from 0 to the maximum rated current of 2 A. This time constant is so small in comparison to the rest of the system that we can omit the inductance. There is a controller in the amplifier that controls the voltage to make the current follow the control signal *u(t)*. The torque produced by the motor can be modelled as proportional to the motor current (the reversed sign is caused by the sign convention of the ready-made software function to control the motor):

$$M_{electric} = k_i i(t) = -k_i u(t) \qquad (5)$$

4.1.5 Friction

The pendulum is affected by both friction in the bearings and air resistance when it swings. We made a simple experiment, just letting the pendulum swing back and forth with the motor turned off. The amplitude of the swinging decreased slowly over more than a minute, so we decided that the friction there could be omitted.

The motor shaft also has some friction that has a stronger influence on the system. We modelled this friction as a reverse torque proportional to the angular velocity of the motor and inertia wheel:

$$M_{friction} = -k_f \omega_r \qquad (6)$$

This looks very much like the effect of the EMF, and in the modelling we decided to treat it as one single breaking torque, proportional to the angular velocity.

4.2 A non-linear state-space model

We could then write the sum of all the torques acting on the inertia wheel and the pendulum:

$$M_r = J_r\dot{\omega}_r = -k_i u(t) - k_f \omega_r(t) \tag{7}$$

$$M = J_p\ddot{\theta} = -mgl\sin\theta - M_r \tag{8}$$

Using three states, this can be converted to a state-space model:

$$\bar{x} = \begin{pmatrix} \dot{\theta} \\ \theta \\ \omega_r \end{pmatrix}, \dot{\bar{x}} = \begin{pmatrix} -a\sin x_2 + b_p c x_3 \\ x_1 \\ -b_r c x_3 \end{pmatrix} + \begin{pmatrix} b_p \\ 0 \\ -b_r \end{pmatrix} u \tag{9}$$

Here we have $a = \frac{mgl}{J_p}, b_p = \frac{k_i}{J_p}, b_r = \frac{k_i}{J_r}, c = \frac{k_f}{k_i}$. The values for these parameters could be found in a document [3]. They are a=78.4, b_p=1.08, b_r=198 and c=0.012.

4.3 Linearising the model

To be able to use linear control design methods, we linearised the model around the instable equilibrium (pendulum pointing upwards). This is done by replacing the sinus term with its linear Taylor expansion at θ=π:

$$\sin\theta = 0 + \cos\pi(\theta - \pi) + O(\theta^2) \tag{10}$$

The linear model then becomes

$$\Delta\bar{x} = \begin{pmatrix} \dot{\theta} \\ \theta - \pi \\ \omega_r \end{pmatrix}, \Delta\dot{\bar{x}} = \begin{pmatrix} 0 & a & b_p c \\ 1 & 0 & 0 \\ 0 & 0 & -b_r c \end{pmatrix} \Delta\bar{x} + \begin{pmatrix} b_p \\ 0 \\ -b_r \end{pmatrix} u \tag{11}$$

4.4 Verifying the model

As described in the section about the DSP, we managed to get data from both encoders in the Mechkit. We tried two different experiments.

4.4.1 Verifying the motor dynamics

For three seconds the motor is accelerated to full forward speed and then instantaneously switched to full reverse speed, for three more seconds. We held the motor still all the time to minimise vibrations.

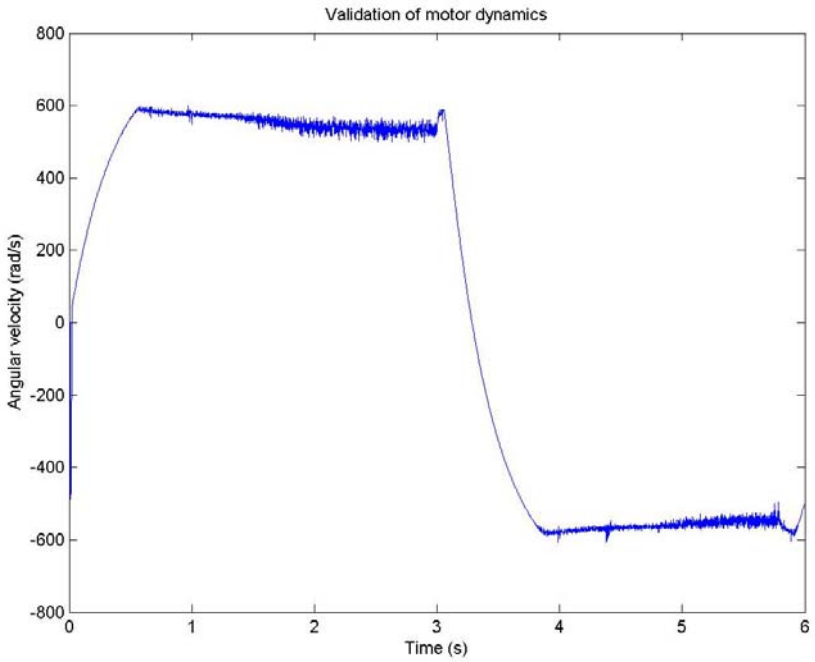

Figure 3. Validation of motor dynamics.

The motor dynamics agree very well with the model, except at high velocities when the velocity saturates earlier than predicted. It even decreases some, at constant current. Judging from the sound of the motor we currently believe that this is caused by high-frequency vibrations, possibly taking energy from the rotational movement. So far we have decided not to incorporate this in the model, as this would make it unnecessarily complex.

4.4.2 Verifying coupling between motor torque and pendulum swinging

This test showed the pendulum response to a step in the control signal. We started with the pendulum hanging down and the motor turned off, then at t=0.1 s we made a step to u=10. At t=5.8 s we stopped the motor again:

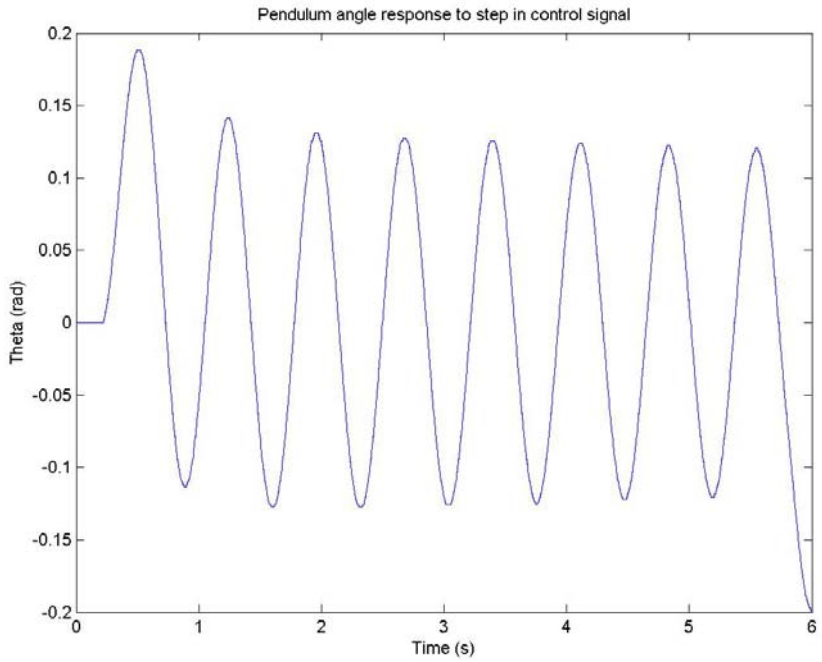

Figure 4. The pendulum angle response to a step in the control signal

This too agrees very well with simulations. The step response also shows that the friction in the bearings of the pendulum are negligible as the amplitude decays very slowly.

5 Controller design

To make the control design simpler we decided to divide the controller into three different parts, balance, swing and stop. The balance controller is responsible for balancing the pendulum at the inverted equilibrium, the swing controller should swing the pendulum until it reaches the inverted position and the stop controller should bring the pendulum to the down position when shutting the systemn down.

5.1 Balance controller

We had two requirements on our controllers; we wanted them to stabilise the pendulum at the inverted equilibrium while keeping the motor velocity as low as possible. A low motor velocity gives the controller a maximal ability to react to sudden disturbances, from any direction. If the motor has a high velocity in any direction, it will not be able to accelerate much more in that direction if needed to produce torque.

5.1.1 LQG controller

Our first approach was to use an LQG controller. This approach felt appropriate to use for this kind of problem, as we wanted to minimise all the states and make the system non-sensitive and robust. It also guarantees a phase margin of at least 60° and an infinite amplitude margin. The LQG-method minimises the integral:

$$\int_0^\infty \left(z^T \mathbf{Q} z + \mathbf{R} u^2 \right) dt \qquad (12)$$

Here z are the states of the system, u the input to the system and **Q** and **R** are penalty matrices. We chose **R**=100 and **Q**=**I**, meaning that all states are equally important to minimise. When simulating the system with the state feedback from the LQG it worked fine, the system stabilised at the inverted equilibrium even if the pendulum was perturbed at the start. The control law is

$$u = -\begin{pmatrix} 53.32 & 472.2 & 0.1127 \end{pmatrix} \bar{x} \qquad (13)$$

The first problem we ran into, during the control design, was when trying to calculate the sensitivity S and complementary sensitivity function T. The sensitivity function showed that the system would be extremely sensitive to disturbance and the complementary sensitivity function looked very strange, probably due to numerical calculation errors.

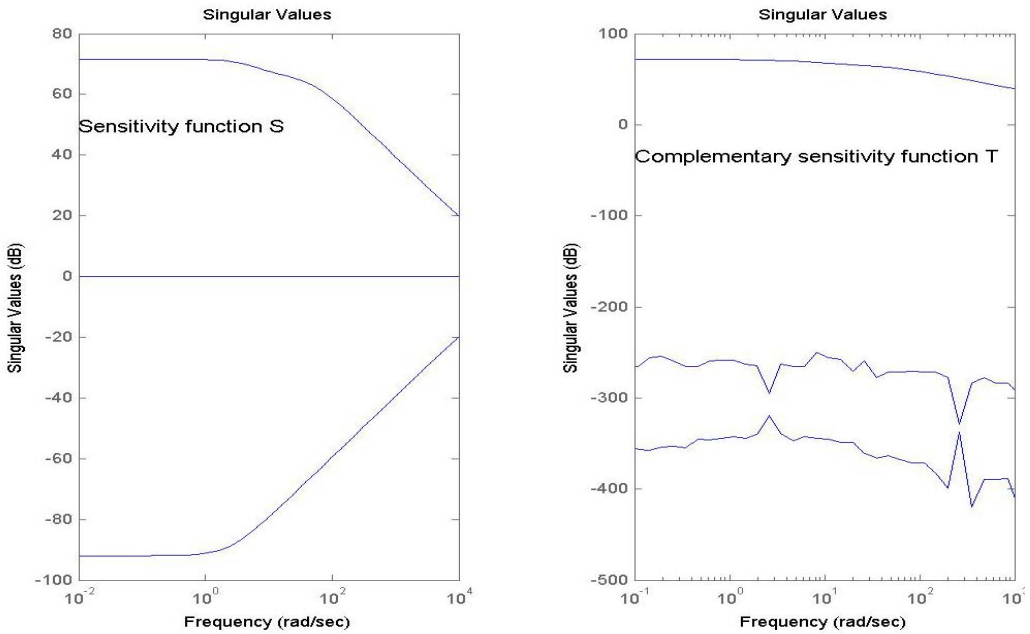

Figure 5. The sensitivity and complementary sensitivity of the closed-loop system with an LQG controller

Despite the bad sensitivity and the strange complementary sensitivity function we decided to make a test implementation of the controller. Just like in the simulation the implemented controller was able to stabilise the pendulum at the inverted position. The controller was even able to swing the pendulum to the inverted position and stabilise it. This was an unexpected consequence of the fact that we used a modulo operator to keep the angle (θ-π) in the interval [-π,π]. The angle thus made an abrupt change at the down position, resulting in the controller

giving the pendulum impulses that made it swing. However, the controller was not able to stabilise the pendulum when the velocity was too high.

5.1.2 PID controller

After our first consultant meeting we decided to try a simpler approach than the LQG, a PID controller. We decided to use the angle θ as the input to the controller. To make sure that the motor velocity did not become too large, we also added a small proportional term, keeping it down. The first approach was to use lead-lag design directly on the instable system, but this gave strange results and a system that was theoretically stable but did not work when implemented in practice. When simulated, this controller behaved well, even when we inserted noise at the output of the process. We currently do not know why it did not reveal the true instability of the system.

Our second approach was to first design a PI-controller, using pole placement, that theoretically stabilised the system but gave very poor performance. We then treated that closed-loop system as a new process and used the lead-lag algorithm on that. To implement this, we then integrated both the inner and outer loop into one control law. This gave reasonable results, both theoretically and practically, but still a little worse than the LQG controller.

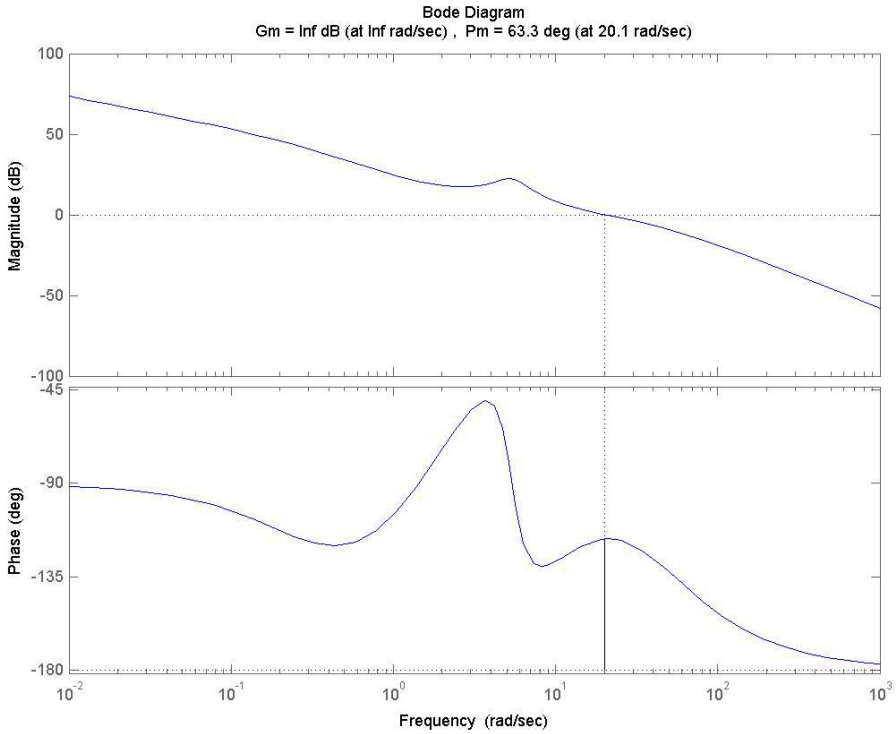

Figure 6. Bode diagram for the open-loop system designed with an inner PI-controller and an outer controller designed with lead-lag.

The transfer functions of the inner and outer controller were

$$F_i = \frac{100\,\mathrm{s} + 180}{\mathrm{s}},\ F_o = \frac{11.93\,\mathrm{s}^3 + 280.9\,\mathrm{s}^2 + 1891\,\mathrm{s} + 2755}{\mathrm{s}^3 + 74.64\,\mathrm{s}^2 + 1386\,\mathrm{s}} \tag{14}$$

When implementing the PID controller, we had some problems with integral windup. We first countered this by just resetting the integral state whenever the control signal saturated, but then we refined the approach some and just did not update the integral state when the control signal saturated. Both approaches gave good results, but the controller had some problems taking control of the system again once it had fallen out of the balancing state. We hope to avoid this problem by using a more advanced approach, with an integrator that tracks the difference between the control signal and its saturated version, subtracting this from the integral state. So far the simulations have not been satisfactory, so we have not implemented that yet.

5.2 Swing controller

The swinging mode is much simpler than the balancing mode. Right now we use a P-controller, taking $(\theta \bmod 2\pi)-\pi$ as input argument. This gives a step from -π to π in the input signal when the pendulum passes the lower position. The controller then rapidly changes the control signal to the motor, creating a torque. There are some difficulties in the swing mode design:

The balance controller can run into problems if the velocity of the pendulum is to high when entering balance control mode. Therefore we should try to keep the velocity as low as possible when swinging, without interfering with the performance, i.e. the time to swing the pendulum up to the inverted position.

To be able to minimise the velocity we analysed the system and realised that the swinging mode can be entered in three different cases. The first case is when we start the controller and the pendulum is in the down position. Here we can use as much power as possible to make the pendulum swing and then reduce the power when we get close to the inverted position. The second case is when the balance controller fails to catch the pendulum. In this case the pendulum already has enough velocity to get to the inverted position. The last case is when the pendulum falls out of the balance mode, and then it will just need a small push when swinging back up.

At the moment we have some difficulties in controlling the impulse to the pendulum, so we are planning to change the swing controller into a P-controller using the angle θ as input. We can then change the P parameter depending on the velocity at some small angle.

5.3 Stop controller

The stop controller is to be used if the velocity of the pendulum is getting to high to recover from. It will also be used when the user would like to end the balancing. This controller is similar to the balance controller with the only difference that we're now stabilising around a stable equilibrium. We have tried to stabilise it using only a P-controller and it works but it will need improvements to get the pendulum to a complete stop.

5.4 Integrating controllers

The goal of the overall design is that the swing mode controller should give the pendulum a velocity low enough for the balancing mode controller to stabilise it in the inverted position. There should be no need for special case routines slowing the pendulum down when it is about

to enter balancing mode. Right now we have a zone of ±25° around the inverted position where the balancing mode controller takes control of the pendulum.

The stop mode controller of course takes control if the user gives the command to stop the balancing. In the future we intent to implement a safety function, starting the stop mode controller if the pendulum velocity exceeds a preset value.

5.5 Implementation of controllers in the DSP

5.5.1 Continuous vs. discrete representation

To make our calculations simpler, we decided to do the controller design for continuous systems and also use a sampling frequency high enough to be able to approximate the controller with a time-continuous version. We chose a sampling time of 1 ms. In the future we expect to change to a discrete model and design a corresponding discrete controller.

5.5.2 Differentiation

When implementing the controllers, we needed to be able to measure the derivatives of both θ and θ_r. (The encoder at the motor actually gives us values of the angle θ_r, that we then have to differentiate to get ω_r). We started by just taking the differences between two consecutive readings, but this gave a very strongly quantized derivative, due to the limited resolution of the encoders. After some tests we instead decided to use a fixed lag of five samples and calculate the derivative as

$$\frac{d\theta}{dt} \approx \frac{\theta_t - \theta_{t-5}}{5T_s} \tag{15}$$

5.5.3 Implementing an LQG controller

When implementing the LQG controller we chose not to have an observer, but to differentiate the angles and treat all three states as measurable. The control law then became a simple vector scalar product. We used two arrays to store old values of θ and θ_r, for the differentiation.

5.5.4 Implementing a PID controller

At first we calculated the P-, I- and D-parts respectively and then added them to get the control signal. This gave bad results, probably due to the imperfections of our derivative and integral approximations. We then converted the transfer function of the controller to state-space form, which seems more appropriate for computer implementation. This, combined with some kind of anti-windup, gave reasonable results.

6 Conclusions

The work has so far progressed faster than expected. We have confidence in the model, which agrees well with validation data. Implementing controllers has proven to be easier if they are expressed in state-space form, possibly since this better matches the way a computer works. There are problems when implementing both integration (integral windup) and differentiation (high noise due to limited resolution) that need to be addressed. So far the combination of controllers for swinging the pendulum up and balancing it are not sufficiently good, because the pendulum often reaches the inverted position with a too high velocity that the balancing controller cannot stabilise.

7 References

1. A J Thor, A Höglund. *Partikeldynamik och statik.* Studentlitteratur, Lund, 2001.
2. Pittman motors (1999). *Datasheet: LO-COG DC servo motors*. Mechatronics Control Kit, [cd-rom], Mechatronics Systems, Inc. 2001.
3. K J Åström, D J Block, M W Spong. *The reaction wheel pendulum.* Mechatronics Control Kit, [cd-rom], Mechatronics Systems, Inc. 2001.