Final Test - Solutions

Course: SCE4206 Systems and Control Laboratory
Teacher: Hans-Petter Halvorsen

<table>
<thead>
<tr>
<th>Class: SCE2</th>
<th>Date: 06.12.2010</th>
<th>Time: 0900-1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>The assignment contains:</td>
<td>Pages: 4 (incl. this page)</td>
<td>Tasks: 3</td>
</tr>
<tr>
<td>Aids:</td>
<td>Default aids are allowed</td>
<td></td>
</tr>
</tbody>
</table>

The test counts 20% of the final grade in the course.

If you miss assumptions for solving some of the problems, you must define proper assumptions yourself.

There are 20 equally weighed questions, which mean you should use approximately 6 minutes on each question. Short answers are required.
Task 1 (35%) State Estimation

1.1 (5%) What is a Kalman Filter? Describe briefly how it works.

Solutions:
Kalman Filter is a commonly used method to estimate the values of state variables of a dynamic system that is excited by stochastic (random) disturbances and stochastic (random) measurement noise. The Kalman Filter is a state estimator which produces an optimal estimate in the sense that the mean value of the sum of the estimation errors gets a minimal value.

Below we see a sketch of how a Kalman Filter is working:

![Kalman Filter Diagram]

The estimator (model of the system) runs in parallel with the system (real system or model). The measurement(s) is used to update the estimator.

1.2 (5%) Describe briefly the different steps in the Kalman Filter algorithm.

Solutions:
Pre Step: Find the steady state Kalman Gain $K$
$K$ is time-varying, but you normally implement the steady state version of Kalman Gain $K$.

Init Step: Set the initial Apriori (Predicted) state estimate
$$\hat{x}_0 = x_0$$

Step 1: Find Measurement model update
For Linear State-space model:
$$\tilde{y}_k = g(\hat{x}_k, u_k)$$
$$\hat{y}_k = C\hat{x}_k + Du_k$$

Step 2: Find the Estimator Error
$$e_k = y_k - \tilde{y}_k$$

Step 3: Find the Aposteriori (Corrected) state estimate
$$\hat{x}_k = \hat{x}_k + Ke_k$$
Where $K$ is the Kalman Filter Gain. Use the steady state Kalman Gain or calculate the time-varying Kalman Gain.

Step 4: Find the Apriori (Predicted) state estimate update
For Linear State-space model:
$$\hat{x}_{k+1} = f(\hat{x}_k, u_k)$$
$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k$$
Step 1-4 goes inside a loop in your program.
1.3 (5%) What kind of information is needed to find the steady-state Kalman Gain? Describe how you would do this in MathScript/MATLAB. You don’t need to know the exact names of the different functions in MathScript/MATLAB.

**Solutions:**
The following information is necessary in order to find the steady-state Kalman Gain:

MathScript/MATLAB Example:

```matlab
ssmodel = ss(A, B, C, D);
% Discrete model:
Ts=0.1; % Sampling Time
discretemodel = c_to_d(ssmodel, Ts);
% Check for Observability:
O = obsvmx (discretemodel);
r = rank(O);
% Find Kalman Gain
Q=[0.01 0; 0 200];
R=[0.0001];
[K] = kalman(discretemodel, Q, R);
```

1.4 (5%) Tuning the Kalman Filter: Describe how you would tune a Kalman Filter.

**Solutions:**
Usually it is necessary to fine-tune the Kalman Filter when it is connected to the real system. The process disturbance (noise) auto-covariance \( Q \) and/or the measurement noise auto-covariance \( R \) are commonly used for the tuning.

\( R \) is relatively easy to calculate from a time series of measurements (using some variance function in for example LabVIEW or MATLAB).

How do you tune \( Q \) to avoid too noisy estimates? The larger \( Q \) the stronger measurement-based updating of the state estimates because a large \( Q \) tells the Kalman Filter that the variations in the real state variables are assumed to be large.

So, the main tuning rule is as follows: Select as large \( Q \) as possible without the state estimates becoming too noisy.

1.5 (5%) What is an Observer? Describe briefly how it works.

**Solutions:**
Observers are an alternative to the Kalman Filter.
An Observer is an algorithm for estimating the state variables in a system based on a model of the system. Observers have the same structure as a Kalman Filter.

In Observers you specify how fast and stable you want the estimates to converge to the real values, i.e., you specify the eigenvalues of the system. Based on the eigenvalues you will find the Observer gain $K$ that is used to update the estimates.

One simple way to find the eigenvalues is to use the Butterworth eigenvalues from the Butterworth polynomial. When we have found the eigenvalues we can then use the Ackerman in order to find the Observer gain.

1.6 (5%) Describe advantages and disadvantages with an Observer versus a Kalman Filter.

**Solutions:**
The theory and implementation of observers are simpler than with Kalman Filters.

One particular drawback about observers is that they are not straightforward to design for systems having more than one measurement, while this is straightforward for Kalman Filters.

1.7 (5%) Given the following differential equations:

$$\begin{align*}
\dot{x}_1 &= -\frac{1}{A} x_2 + \frac{1}{A} K u \\
\dot{x}_2 &= 0
\end{align*}$$

**Use the Euler Forward discretization method and find the discrete state-space model on the following form:**

$$\begin{align*}
x_{k+1} &= A x_k + B u_k \\
y_k &= C x_k + D u_k
\end{align*}$$

**Solutions:**
Applying Euler Forward discretization method gives:

$$\begin{align*}
fraction{1}{T_s}\left(x_1(k+1) - x_1(k)\right) &= -\frac{1}{A} x_2(k) + \frac{1}{A} K u(k)
\end{align*}$$
This gives:

\[ x_1(k + 1) = x_1(k) - \frac{T_s}{A} x_2(k) + \frac{T_sK}{A} u(k) \]
\[ x_2(k + 1) = x_2(k) \]

The discrete state-space model then becomes:

\[
\begin{bmatrix}
  x_1(k+1) \\
  x_2(k+1)
\end{bmatrix} =
\begin{bmatrix}
  1 & -\frac{T_s}{A} \\
  0 & 1
\end{bmatrix}
\begin{bmatrix}
  x_1(k) \\
  x_2(k)
\end{bmatrix} +
\begin{bmatrix}
  \frac{T_sK}{A} \\
  0
\end{bmatrix} u(k)
\]

\[ y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\
  x_2(k) \end{bmatrix} \]

Assuming \( y = x_1 \)
**Task 2 (30%) System Identification**

2.1 (5%) What is System Identification?

**Solutions:**

System Identification methods may be used to build mathematical models of dynamic systems based on observed and measured input and output data from the system.

System Identification is defined as: Identification is the determination, on the basis of input and output, of a system within a specified class of systems, to which the system under test is equivalent.

2.2 (5%) What is Model Validation? – Describe briefly how you would do it.

**Solutions:**

Model validating is needed to check if the model is good. One method is to compare simulated signal with measured signal: You can simulate using the estimated model and compare the simulated output signal with the logged or measured output signal.

1. Excite the real system, and log input and output:

   ![Diagram of system identification process]

2. Split data, for estimation and for validation:

   ![Diagram of data splitting]

3. Estimate model:

   ![Diagram of model estimation]

4. Check (validate) model using e.g. simulation:

   ![Diagram of model validation]

2.3 (5%) Describe how you generate model data from the physical process.

**Solutions:**

You may use, e.g., LabVIEW for exciting the process and logging signals. Use open-loop experiments (no feedback control system).

You need to use a DAQ device in order to write data to the process (u) and read the measured value (y) back into the computer.
You can use the Write to Measurement File function on the File I/O palette in LabVIEW for writing data to text files (use the LVM data file format, not the TDMS file format which give binary files).

**Excite the real system, and log input and output:**

![Diagram of system](image)

2.4 (5%) List and describe briefly different excitation signals.

**Solutions:**
We have different excitation signals, such as, e.g.:

- A **PRBS** signal (Pseudo Random Binary Signal)
- A **Chirp** Signal
- A **Up-down** signal

**PRBS Signal**
A PRBS signal looks like this:

![PRBS Signal](image)

**Chirp Signal**
A Chirp signal looks like this:
Up-down Signal:
This signal is simple to generate manually during the experiment. This signal gives in many cases enough excitation for the estimator to calculate accurate parameter estimates, but the period of the signal shift must be large enough to give the system output a chance to approximately stabilize between the steps.

2.5 (5%) List and describe briefly different System Identification methods.

Solutions:
When you have a mathematical model and want to find unknown parameters:
Parameter Estimation:

- Least Square Method (LS)

Parameter Estimation using the Least Square Method (LS) is used to find a model with unknown physical parameters in a mathematical model.

When you don’t have a mathematical model, just input/output data:
System Identification
Sub-space methods/Black-Box methods
Polynomial Model Estimation: ARX/ARMAX model Estimation

Sub-space methods/Black-Box methods is used to find a model with non-physical parameters. A sub-space methods/Black-Box method estimates a linear discrete State-space model on the form:

\[ x_{k+1} = Ax_k + Bu_k \]
\[ y_k = Cx_k + Du_k \]

2.6 (5%) Given the following model:

\[ y(u) = au + b \]

The following values are found from experiments:

\[ y(1) = 0.8 \]
\[ y(2) = 3.0 \]
\[ y(3) = 4.0 \]

Describe how you would find the unknowns \( a \) and \( b \) using the Least Square (LS) method in MathScript/MATLAB.

**Solutions:**
We have that:

\[ Y = \Phi \theta \]

Where
\( \theta \) is the unknown parameter vector
\( Y \) is the known measurement vector
\( \Phi \) is the known regression matrix

The solution for \( \theta \) may be found as:

\[ \theta = \Phi^{-1}Y \]

It can be found that the least square solution for \( Y = \Phi Y \) is:

\[ \theta_{LS} = (\Phi^T\Phi)^{-1}\Phi^TY \]

We get:

\[ 0.8 = a \cdot 1 + b \]
\[ 3.0 = a \cdot 2 + b \]
\[ 4.0 = a \cdot 3 + b \]

This becomes:

\[
\begin{bmatrix}
0.8 \\
3.0 \\
4.0 \\
y
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1 \\
2 & 1 \\
3 & 1 \\
\Phi
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
\theta
\end{bmatrix}
\]

- Sub-space methods/Black-Box methods
- Polynomial Model Estimation: ARX/ARMAX model Estimation
We define $Y$ and $\Phi$ in MathScript/MATLAB and find $\theta$ by:

```matlab
phi = [1 1; 2 1; 3 1];
Y = [0.8 3.0 4.0]';
theta = inv(phi'*phi)* phi'*Y
% or simply by
theta=phi\Y
```

The answer becomes:

```
theta =

1.6
-0.6
```

(but you dont need to find the solution)
Task 3 (35%) SCADA System

3.1 (5%) What is a SCADA system?

**Solutions:**
Supervisory Control and Data Acquisition (SCADA) is an industrial control system monitoring and controlling a process. A SCADA system is only a software application.

A SCADA system consists of different modules/systems that together make the SCADA system, e.g.:

- Database System
- Management System
- Control System
- Datalogging System
- Monitoring System
- Alarm System
- Report System

A simple sketch of a SCADA system:

![SCADA System Diagram](image)

3.2 (5%) Create a database diagram (ER diagram) for your SCADA System.

**Solutions:**
Example:
3.3 (5%) Describe what a Primary Key and a Foreign Key is.

**Solutions:**

**Primary Key:**

The primary key of a relational table uniquely identifies each record in the table. It can either be a normal attribute that is guaranteed to be unique (such as Social Security Number in a table with no more than one record per person) or it can be generated by the DBMS (such as a globally unique identifier, or GUID, in Microsoft SQL Server). Primary keys may consist of a single attribute or multiple attributes in combination.

**Foreign Key:**

A foreign key in a database table is a key from another table that refers to (or targets) a specific key, usually the primary key, in the table being used. A primary key can be targeted by multiple foreign keys from other tables. But a primary key does not necessarily have to be the target of any foreign keys.

Example:

```
SCHOOL
PK SchoolId
  SchoolName
  Description
  Address
  Phone
  PostCode
  PostAddress

CLASS
PK ClassId
  ClassId
  ClassName
  Description
```

3.4 (5%) What is SQL? Describe what SQL is and give some examples.

**Solutions:**

SQL Structured Query Language is a database computer language designed for managing data in relational database management systems (RDBMS).
The most common operation in SQL is the query, which is performed with the declarative SELECT statement. SELECT retrieves data from one or more tables, or expressions.

Examples:
```sql
select * from Tag where TagId=5
```

The Data Manipulation Language (DML) is the subset of SQL used to add, update and delete data.

Examples:
```sql
INSERT INTO My_table
(field1, field2, field3)
VALUES
('test', 'N', NULL);
```
```sql
UPDATE My_table
SET field1 = 'updated value'
WHERE field2 = 'N';
```
```sql
DELETE FROM My_table
WHERE field2 = 'N';
```

These 4 operations is referred to a CRUD (Create, Read, Update and Delete)

The acronym CRUD refers to all of the major functions that need to be implemented in a relational database application to consider it complete. Each letter in the acronym can be mapped to a standard SQL statement:

<table>
<thead>
<tr>
<th>Operation</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>INSERT</td>
</tr>
<tr>
<td>Read (Retrieve)</td>
<td>SELECT</td>
</tr>
<tr>
<td>Update</td>
<td>UPDATE</td>
</tr>
<tr>
<td>Delete (Destroy)</td>
<td>DELETE</td>
</tr>
</tbody>
</table>

3.5 (5%) What is a Stored Procedure? Describe how you implement and use a Stored Procedure in Microsoft SQL Server.

**Solutions:**
A stored procedure is a subroutine available to applications accessing a relational database system. Typical uses for stored procedures include data validation (integrated into the database) or access control mechanisms. Furthermore, stored procedures are used to consolidate and centralize logic that was originally implemented in applications. Large or complex processing that might require the execution of several SQL statements is moved into stored procedures, and all applications call the procedures only.

A stored procedure is a precompiled collection of SQL statements and optional control-of-flow statements, similar to a macro. Each database and data provider supports stored procedures differently. Stored procedures offer the following benefits to your database applications:

- **Performance**—Stored Procedures are usually more efficient and faster than regular SQL queries because SQL statements are parsed for syntactical accuracy and precompiled by the DBMS when the stored procedure is created. Also, combining a large number of SQL statements with conditional logic and parameters into a stored procedure allows the procedures to perform queries, make decisions, and return results without extra trips to the database server.

- **Maintainability**—Stored Procedures isolate the lower-level database structure from the application. As long as the table names, column names, parameter names, and types do not change from what is stated in the stored
procedure, you do not need to modify the procedure when changes are made to the database schema. Stored procedures are also a way to support modular SQL programming because after you create a procedure, you and other users can reuse that procedure without knowing the details of the tables involved.

Security—When creating tables in a database, the Database Administrator can set EXECUTE permissions on stored procedures without granting SELECT, INSERT, UPDATE, and DELETE permissions to users. Therefore, the data in these tables is protected from users who are not using the stored procedures. Stored procedures are similar to user-defined functions. The major difference is that functions can be used like any other expression within SQL statements, whereas stored procedures must be invoked using the CALL statement.

The syntax for creating a Stored Procedure is as follows:

```
CREATE PROCEDURE <ProcedureName>
@<Parameter1> <datatype>
```

Example: Create a Stored Procedure:

This Procedure gets Customer Data based on a specific Order Number.

```
CREATE PROCEDURE sp_CustomerOrders
@OrderNumber varchar(50)
AS

declare @CustomerId int

select @CustomerId = CustomerId from [ORDER] where OrderNumber = @OrderNumber

select CustomerId, FirstName, LastName, [Address], Phone from CUSTOMER where CustomerId = @CustomerId

```

Example: Using a Stored Procedure:

```
exec sp_CustomerOrders '10002'
```

This gives the following result:

```
CustomerId      FirstName      LastName     Address      Phone
--------         -----------     -----------     ---------     -------
1              Tor           Hansen       Veveien 15     77775678
```

3.6 (5%) What is a Database Trigger?

Solutions:
A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for keeping the integrity of the information on the database. For example, when a new record (representing a new worker) added to the employees table, new records should be created also in the tables of the taxes, vacations, and salaries.
Triggers are commonly used to:

- prevent changes (e.g., prevent an invoice from being changed after it’s been mailed out)
- log changes (e.g., keep a copy of the old data)
- audit changes (e.g., keep a log of the users and roles involved in changes)
- enhance changes (e.g., ensure that every change to a record is time-stamped by the server’s clock, not the client’s)
- enforce business rules (e.g., require that every invoice have at least one line item)
- execute business rules (e.g., notify a manager every time an employee’s bank account number changes)
- replicate data (e.g., store a record of every change, to be shipped to another database later)
- enhance performance (e.g., update the account balance after every detail transaction, for faster queries)

The major features of database triggers, and their effects, are:

- do not accept parameters or arguments (but may store affected data in temporary tables)
- cannot perform commit or rollback operations because they are part of the triggering SQL statement
- can cancel a requested operation
- can cause mutating table errors, if they are poorly written.

Example:

```sql
CREATE TRIGGER tr_CheckGrades ON GRADE
FOR UPDATE, INSERT
AS
DECLARE @Grade varchar(10), @GradeId int
select @Grade=Grade, @GradeId=GradeId from INSERTED

IF @Grade = '1'
    update GRADE set Grade='A' where GradeId=@GradeId
IF @Grade = '2'
    update GRADE set Grade='B' where GradeId=@GradeId
IF @Grade = '3'
    update GRADE set Grade='C' where GradeId=@GradeId
IF @Grade = '4'
    update GRADE set Grade='D' where GradeId=@GradeId
IF @Grade = '5'
    update GRADE set Grade='E' where GradeId=@GradeId
IF @Grade = '6'
    update GRADE set Grade='F' where GradeId=@GradeId
GO
```

3.7 (5%) What is OPC?

Solutions:

OPC (“OLE for process control”) is a standard interface between numerous data sources, such as programmable logic controllers (PLCs), remote terminal units (RTUs), and sensors on a factory floor to HMI/SCADA applications, application tools, and databases. With OPC, your device-side server and application software can communicate without your duplicating device driver development and providing support for
hardware feature changes. The OPC Foundation defines the standards that allow any client to access any OPC-compatible device.

While OPC originally stood for “OLE for Process Control”, the official stance of the OPC Foundation is that OPC is no longer an acronym and the technology is simply known as “OPC”. One of the reasons behind this is while OPC is heavily used within the process industries.