Chapter 8

State estimation with Kalman Filter

8.1 Introduction

This chapter describes how to *estimate the values of state variables* of a dynamic system. Why can such state estimators be useful?

- **Supervision**: State estimates can provide valuable information about important variables in a physical process, for example feed composition to a reactor, flow variations to a separator, etc.
- **Control**: In general, the more information the controller has about the process it controls, the better (mor accurate) it can control it. State estimators can be a practical or economical alternative to real measurements. For example, the environmental forces acting on a vessel can be used by the controller to better control the position of the vessel, as in Dynamic Positioning systems.

Because the state estimators are kind of sensors implemented in software (computer programs) they are some times denoted *soft sensors*.

This chapter describes the *Kalman Filter* which is the most important algorithm for state estimation. The Kalman Filter was developed by Rudolf E. Kalman around 1960 [7]. There is a continuous-time version of the Kalman Filter and several discrete-time versions. (The discrete-time versions are immediately ready for implementation in a computer program.) Here the *predictor-corrector* version of the discrete-time Kalman

Filter will be described. This version seems to be the most commonly used version.

8.2 Observability

A necessary condition for the Kalman Filter to work correctly is that the system for which the states are to be estimated, is *observable*. Therefore, you should check for observability before applying the Kalman Filter. (There may still be other problems that prevent the Kalman Filter from producing accurate state estimates, as a faulty or inaccurate mathematical model.)

Observability for the discrete-time systems can be defined as follows [13]: The discrete-time system

$$x(k+1) = Ax(k) + Bu(k) \tag{8.1}$$

$$y(k) = Cx(k) + Du(k) \tag{8.2}$$

is observable if there is a finite number of time steps k so that knowledge about the input sequence $u(0), \ldots, u(k-1)$ and the output sequence $y(0), \ldots, y(k-1)$ is sufficient to determine the initial state state of the system, $x(0)$.

Let us derive a criterion for the system to be observable. Since the influende of input u on state x is known from the model, let us for simplicity assume that $u(k) = 0$. From the model (8.1) – (8.2) we get

$$\begin{aligned} y(0) &= Cx(0) && (8.3)\\ y(1) &= Cx(1) = CAx(0) && (8.4)\\ &\vdots \\ y(n-1) &= CA^{n-1}x(0) && (8.5) \end{aligned}$$

which can be expressed compactly as

$$\underbrace{\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}}_{M_{\text{obs}}} x(0) = \underbrace{\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(n-1) \end{bmatrix}}_{Y} \tag{8.6}$$

Let us make a definition:

Observability matrix:

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{8.7}$$

(8.6) has a unique solution only if the rank of M_{obs} is n. Therefore:

Observability Criterion:

The system (8.1) – (8.2) is observable if and only if the observability matrix has rank equal to n where n is the order of the system model (the number state variables).

The rank can be checked by calculating the determinant of M_{obs}. If the determinant is non-zero, the rank is full, and hence, the system is observable. If the determinant is zero, system is non-observable.

Non-observability has several concequences:

- The transfer function from the input variable y to the output variable y has an order that is less than the number of state variables (n).
- There are state variables or linear combinations of state variables that do not show any response.
- The steady-state value of the Kalman Filter gain can not be computed. This gain is used to update the state estimates from measurements of the (real) system.

Example 18 ***Observability***

Given the following state space model:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B} u(k) \tag{8.8}$$

$$y(k) = \underbrace{\begin{bmatrix} c_1 & 0 \end{bmatrix}}_{C} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \underbrace{[0]}_{D} u(k) \tag{8.9}$$

The observability matrix is ($n = 2$)

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA^{2-1} = CA \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} c_1 & 0 \end{bmatrix} \\ - - - - - - - - - - - \\ \begin{bmatrix} c_1 & 0 \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} c_1 & 0 \\ c_1 & ac_1 \end{bmatrix} \tag{8.10}$$

The determinant of M_{obs} is

$$\det(M_{\text{obs}}) = c_1 \cdot ac_1 - c_1 \cdot 0 = a\, {c_1}^2 \tag{8.11}$$

The system is observable only if $a\, {c_1}^2 \neq 0$.

- Assume that $a \neq 0$ which means that the first state variable, x_1, contains some non-zero information about the second state variable, x_2. Then the system is observable if $c_1 \neq 0$, i.e. if x_1 is measured.
- Assume that $a = 0$ which means that x_1 contains no information about x_2. In this case the system is non-observable despite that x_1 is measured.

[End of Example 18]

8.3 The Kalman Filter algorithm

The *Kalman Filter* is a state estimator which produces an optimal estimate in the sense that *the mean value of the sum (actually of any linear combination) of the estimation errors* gets a minimal value. In other words, The Kalman Filter gives the following sum of squared errors:

$$E[{e_x}^T(k) e_x(k)] = E\left[{e_{x_1}}^2(k) + \cdots + {e_{x_n}}^2(k)\right] \tag{8.12}$$

a minimal value. Here,

$$e_x(k) = x_{est}(x) - x(k) \tag{8.13}$$

is the estimation error vector. (The Kaman Filter estimate is sometimes denoted the "least mean-square estimate".) This assumes actually that the model is linear, so it is not fully correct for nonlinear models. It is assumed the that the system for which the states are to be estimated is excited by *random ("white") disturbances (*or *process noise*) and that the

measurements (there must be at least one real measurement in a Kalman Filter) contain *random ("white") measurement noise*.

The Kalman Filter has many applications, e.g. in dynamic positioning of ships where the Kalman Filter estimates the position and the speed of the vessel and also environmental forces. These estimates are used in the positional control system of the ship. The Kalman Filter is also used in soft-sensor systems used for supervision, in fault-detection systems, and in Model-based Predictive Controllers (MPCs) which is an important type of model-based controllers.

The Kalman Filter algorithm was originally developed for systems assumed to be represented with a *linear* state-space model. However, in many applications the system model is *nonlinear*. Furthermore the linear model is just a special case of a nonlinear model. Therefore, I have decided to present the Kalman Filter for nonlinear models, but comments are given about the linear case. The Kalman Filter for nonlinear models is denoted the *Extended Kalman Filter* because it is an extended use of the original Kalman Filter. However, for simplicity we can just denote it the Kalman Filter, dropping "extended" in the name. The Kalman Filter will be presented without derivation.

The Kalman Filter presented below assumes that the system model consists of this discrete-time (possibly nonlinear) state space model:

$$x(k+1) = f[x(k),\, u(k)] + Gw(k) \tag{8.14}$$

and this (possibly nonlinear) measurement model:

$$y(k) = g[x(k),\, u(k)] + Hw(k) + v(k) \tag{8.15}$$

A *linear* model is just a special case:

$$x(k+1) = \underbrace{Ax(k) + Bu(k)}_{=f} + Gw(k) \tag{8.16}$$

and

$$y(k) = \underbrace{Cx(k) + Du(k)}_{=g} + Hw(k) + v(k) \tag{8.17}$$

The models above contains the following variables and functions:

- x is the state vector of n state variables:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{8.18}$$

- u is the input vector of m input variables:

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \tag{8.19}$$

  It is assumed that the value of u is known. u includes control variables and known disturbances.

- f is the system vector function:

$$f = \begin{bmatrix} f_1() \\ f_2() \\ \vdots \\ f_n() \end{bmatrix} \tag{8.20}$$

  where $f_i()$ is any nonlinear or linear function.

- w is random (white) disturbance (or process noise) vector:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_q \end{bmatrix} \tag{8.21}$$

  with auto-covariance

$$R_w(L) = Q\delta(L) \tag{8.22}$$

  where Q (a $q \times q$ matrix of constants) is the auto-covariance of w at lag $L = 0$. $\delta(L)$ is the unit pulse function, cf. (6.25). A standard assumption is that

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_{nn} \end{bmatrix} = \text{diag}(Q_{11}, Q_{22}, \cdots, Q_{nn}) \tag{8.23}$$

  Hence, the number q of process disturbances is assumed to be equal to the number n of state variables. Q_{ii} is the variance of w_i.

- G is the process noise gain matrix relating the process noise to the state variables. It is common to assume that $q = n$, making G square:

$$G = \begin{bmatrix} G_{11} & 0 & 0 & 0 \\ 0 & G_{22} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_{nn} \end{bmatrix} \tag{8.24}$$

  In addition it is common to set the elements of G equal to one:

$$G_{ii} = 1 \tag{8.25}$$

  making G an identity matrix:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I_n \tag{8.26}$$

- y is the measurement vector of r measurement variables:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} \tag{8.27}$$

- g is the measurement vector function:

$$g = \begin{bmatrix} g_1() \\ g_2() \\ \vdots \\ g_r() \end{bmatrix} \tag{8.28}$$

  where $g_i()$ is any nonlinear or linear function. Typically, g is a linear function on the form

$$g(x) = Cx \tag{8.29}$$

  where C is the measurement gain matrix.

- H is a gain matrix relating the disturbances *directly* to the measurements (there will in addition be an indirect relation because the disturbances acts on the states, and some of the states are measured). It is however common to assume that H is a zero matrix of dimension $(r \times q)$:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & H_{rq} \end{bmatrix} \tag{8.30}$$

- v is a random (white) measurement noise vector:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_r \end{bmatrix} \tag{8.31}$$

with auto-covariance

$$R_v(L) = R\delta(L) \tag{8.32}$$

where R (a $r \times r$ matrix of constants) is the auto-covariance of v at lag $L = 0$. A standard assumption is that

$$R = \begin{bmatrix} R_{11} & 0 & 0 & 0 \\ 0 & R_{22} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & R_{rr} \end{bmatrix} = \text{diag}(R_{11}, R_{22}, \cdots, R_{rr}) \tag{8.33}$$

Hence, R_{ii} is the variance of v_i.

Note: If you need to adjust the "strength" or power of the process noise w or the measurement noise v, you can do it by increasing the variances, Q and R, respectively.

Here you have the *Kalman Filter:* (The formulas (8.35) – (8.37) below are represented by the block diagram shown in Figure 8.1.)

Kalman Filter state estimation:

1. This step is the *initial step*, and the operations here are executed only once. Assume that the initial guess of the state is x_{init}. The initial value $x_p(0)$ of the predicted state estimate x_p (which is calculated continuously as described below) is set equal to this initial value:

Initial state estimate

$$x_p(0) = x_{init} \tag{8.34}$$

2. Calculate the *predicted measurement estimate* from the predicted state estimate:

Predicted measurement estimate:

$$y_p(k) = g\left[x_p(k)\right] \tag{8.35}$$

(It is assumed that the noise terms $Hv(k)$ and $w(k)$ are not known or are unpredictable (since they are white noise), so they can not be used in the calculation of the predicted measurement estimate.)

3. Calculate the so-called *innovation process or variable* — it is actually the measurement estimate error — as the difference between the measurement $y(k)$ and the predicted measurement $y_p(k)$:

Innovation variable:

$$e(k) = y(k) - y_p(k) \tag{8.36}$$

4. Calculate the *corrected state estimate* $x_c(k)$ by adding the corrective term $Ke(k)$ to the predicted state estimate $x_p(k)$:

Corrected state estimate:

$$\underline{x_c(k) = x_p(k) + Ke(k)} \tag{8.37}$$

Here, K is the *Kalman Filter gain*. The calculation of K is described below.

Note: It is $x_c(k)$ that is used as the state estimate in applications[1].

About terminology: The corrected estimate is also denoted the *posteriori* estimate because it is calculated *after* the present measurement is taken. It is also denoted the *measurement-updated* estimate.

Due to the measurement-based correction term of the Kalman Filter, you can except the errors of the state estimates to be smaller than if there were no such correction term. This correction can be regarded as a *feedback correction of the estimates*, and it is well known from dynamic system theory, and in particular control systems theory, that feedback from measurements reduces errors. This feedback is indicated in Figure 8.1.

5. Calculate the *predicted state estimate* for the next time step, $x_p(k+1)$, using the present state estimate $x_c(k)$ and the known input $u(k)$ in process model:

Predicted state estimate:

$$x_p(k+1) = f[x_c(k),\, u(k)] \tag{8.38}$$

(It is assumed that the noise term $Gv(k)$ is not known or is unpredictable, since it is random, so it can not be used in the calculation of the state estimate.)

About terminology: The predicted estimate is also denoted the *priori* estimate because it is calculated *before* the present measurement is taken. It is also denoted the *time-updated* estimate.

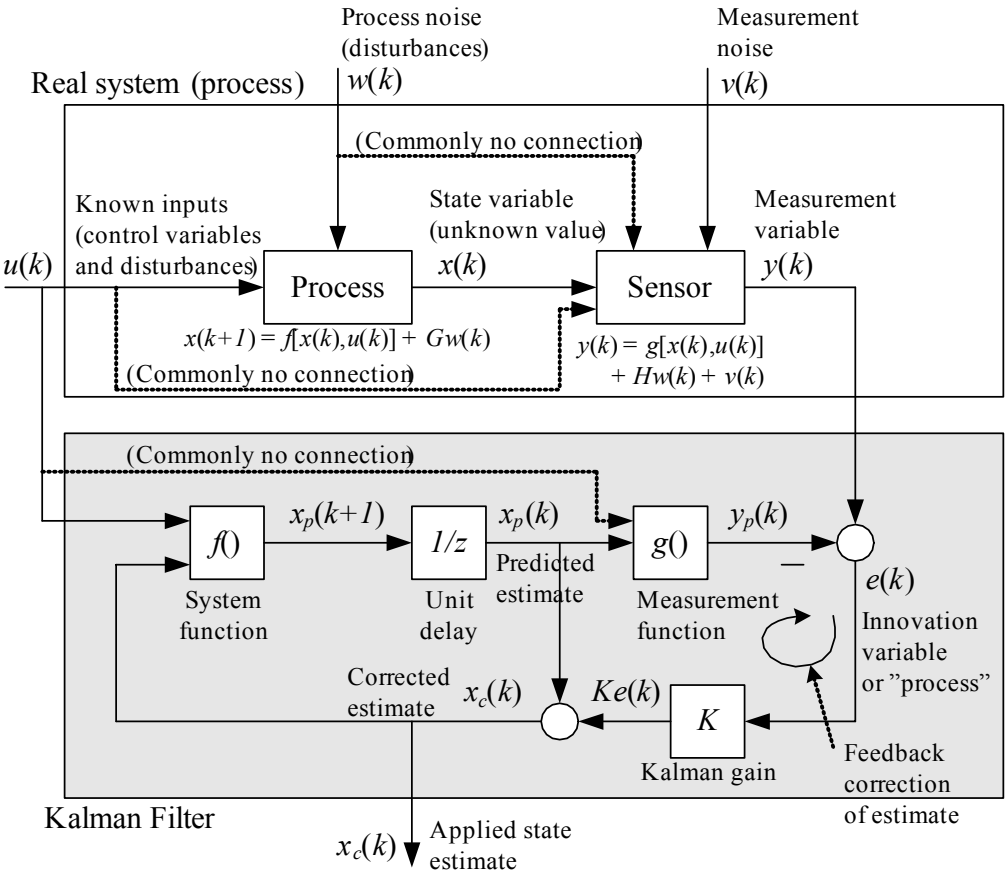

Figure 8.1: The Kalman Filter algorithm (8.35) – (8.38) represented by a block diagram

(8.35) – (8.38) can be represented by the block diagram shown in Figure 8.1.

The *Kalman Filter gain* is a time-varying gain matrix. It is given by the algorithm presented below. In the expressions below the following matrices are used:

- Auto-covariance matrix (for lag zero) of the estimation error of the corrected estimate:

$$P_c = R_{e_{x_c}}(0) = E\left\{(x - m_{x_c})(x - m_{x_c})^T\right\} \tag{8.39}$$

[1] Therefore, I have underlined the formula.

- Auto-covariance matrix (for lag zero) of the estimation error of the predicted estimate:

$$P_d = R_{e_{x_d}}(0) = E\left\{(x - m_{x_d})(x - m_{x_d})^T\right\} \tag{8.40}$$

- The transition matrix A of a linearized model of the original nonlinear model (8.14) calculated with the most recent state estimate, which is assumed to be the corrected estimate $x_c(k)$:

$$A = \left.\frac{\partial f(\cdot)}{\partial x}\right|_{x_c(k),\, u(k)} \tag{8.41}$$

- The measurement gain matrix C of a linearized model of the original nonlinear model (8.15) calculated with the most recent state estimate:

$$C = \left.\frac{\partial g(\cdot)}{\partial x}\right|_{x_c(k),\, u(k)} \tag{8.42}$$

  However, it is very common that $C = g()$, and in these cases no linearization is necessary.

The *Kalman Filter gain* is calculated as follows (these calculations are repeated each program cycle):

Kalman Filter gain:

1. This step is the initial step, and the operations here are executed only once. The initial value $P_p(0)$ can be set to some guessed value (matrix), e.g. to the identity matrix (of proper dimension).
2. Calculation of the *Kalman Gain*:

Kalman Filter gain:

$$K(k) = P_p(k)C^T[CP_p(k)C^T + R]^{-1} \tag{8.43}$$

3. Calculation of *auto-covariance of corrected state estimate error*:

Auto-covariance of corrected state estimate error:

$$P_c(k) = [I - K(k)C]\, P_p(k) \tag{8.44}$$

4. Calculation of *auto-covariance of the next time step of predicted state estimate error*:

Auto-covariance of predicted state estimate error:

$$P_p(k+1) = AP_c(k)A^T + GQG^T \tag{8.45}$$

Here are *comments* to the Kalman Filter algorithm presented above:

1. **Order of formulas in the program cycle**: The Kalman Filter formulas can be executed in the following order:
   - (8.43), Kalman Filter gain
   - (8.36), innovation process (variable)
   - (8.37), *corrected state estimate*, which is the state estimate to be used in applications
   - (8.38), predicted state estimate of next time step
   - (8.44), auto-covarience of error of corrected estimate
   - (8.41), transition matrix in linear model
   - (8.42), measurement matrix in linear model
   - (8.45), auto-covarience of error of predicted estimate of next time step

2. **Steady-state Kalman Filter gain.** If the model is linear and time invariant (i.e. system matrices are not varying with time) the auto-covariances P_c and P_p will converge towards steady-state values. Consequently, the Kalman Filter gain will converge towards a *steady-state Kalman Filter gain* value, K_s[2], which can be pre-calculated. It is quite common to use only the steady-state gain in applications.

   For a nonlinear system K_s may vary with the operating point (if the system matrix A of the linearized model varies with the operating point). In practical applications K_s may be re-calculated as the operating point changes.

   Figure 8.2 illustrates the information needed to compute the steady-state Kalman Filter gain, K_s.

3. **Model errors.** There are always model errors since no model can give a perfect description of a real (practical system). It is possible to analyze the implications of the model errors on the state estimates calculated by the Kalman Filter, but this will not be done in this book. However, in general, the estimation errors are smaller with the Kalman Filter than with a so-called *ballistic state estimator*, which is the state estimator that you get if the Kalman Filter gain K is set to zero. In the latter case there is no correction of the estimates. It is only the predictions (8.38) that make up the estimates. The Kalman Filter then just runs as a simulator.

[2] MATLAB and LabVIEW have functions for calculating the steady-state Kalman Gain.

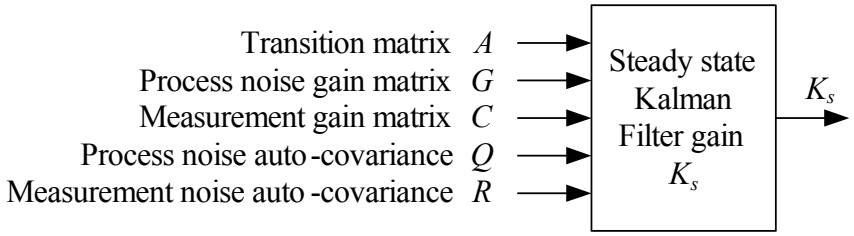

Figure 8.2: Illustration of what information is needed to compute the steady-state Kalman Filter gain, K_s.

Note that you can try to estimate model errors by *augmenting* the states with states representing the model errors. Augmented Kalman Filter is described in Section 8.4.

4. **How to tune the Kalman Filter**: 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. However, since R is relatively easy to calculate from a time series of measurements (using some variance function in for example LabVIEW or MATLAB), we only consider adjusting Q here.

   What is good behaviour of the Kalman Filter? How can good behaviour be observed? It is when the estimates seems to have reasonable values as you judge from your physical knowledge about the physical process. In addition the estimates must not be too noisy! What is the cause of the noise? In real systems it is mainly the measurement noise that introduces noise into the estimates. 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 (remember that the process noise influences on the state variables, cf. (8.15)). Hence, the larger Q the larger Kalman Gain K and the stronger updating of the estimates. But this causes more measurement noise to be added to the estimates because the measurement noise is a term in the innovation process e which is calculated by K:

$$\begin{aligned} x_c(k) &= x_p(k) + Ke(k) && (8.46)\\ &= x_p(k) + K\left\{g\left[x(k)\right] + v(k) - g\left[x_p(k)\right]\right\} && (8.47)\end{aligned}$$

   where v is real measurement noise. So, the main tuning rule is as follows: *Select as large Q as possible without the state estimates becoming too noisy.*

But Q is a matrix! How to select it "large" or "small". Since each of the process disturbances typically are assumed to act on their respective state independently, Q can be set as a diagonal matrix:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_{nn} \end{bmatrix} = \text{diag}(Q_{11}, Q_{22}, \cdots, Q_{nn}) \tag{8.48}$$

where each of the diagonal elements can be adjusted independently. If you do not have any idea about numerical values, you can start by setting all the diagonal elements to one, and hence Q is

$$Q = Q_0 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.49}$$

where Q_0 is the only tuning parameter. If you do not have any idea about a proper value of Q_0 you may initially try

$$Q_0 = 0.01 \tag{8.50}$$

Then you may adjust Q_0 or try to fine tune each of the diagonal elements individually.

5. **The error-model**: Assuming that the system model is linear and that the model is correct (giving a correct representation of the real system), it can be shown that the behaviour of the error of the corrected state estimation, $e_{x_c}(k)$, cf. (8.13), is given by the following *error-model*:[3]

Error-model of Kalman Filter:

$$e_{x_c}(k+1) = (I - KC)\,Ae_{x_c}(k) + (I - KC)\,Gv(k) - Kw(k+1) \tag{8.51}$$

This model can be used to analyze the Kalman Filter.

Note: (8.13) is not identical to the auto-covariance of the estimation error which is

$$P_c(k) = E\{[e_{x_c}(k) - m_{x_c}(k)][e_{x_c}(k) - m_{x_c}(k)]^T\} \tag{8.52}$$

But (8.13) is the *trace* of P_c (the trace is the sum of the diagonal elements):

$$e_{x_c} = \text{trace}\,[\,P_c(k)] \tag{8.53}$$

[3] You can derive this model by subtracting the model describing the corrected state estimate from the model that describes the real state (the latter is simply the process model).

6. **The dynamics of the Kalman Filter.** The error-model (8.51) of the Kalman Filter represents a dynamic system. The dynamics of the Kalman Filter is given can be analyzed by calculating the eigenvalues of the system matrix of (8.51). These eigenvalues are

$$\{\lambda_1, \lambda_2, \ldots, \lambda_n\} = \text{eig}\,[(I - KC)A] \tag{8.54}$$

7. **The stability of the Kalman Filter.** It can be shown that *the Kalman Filter always is an asymptotically stable dynamic system* (otherwise it could not give an optimal estimate). In other words, the eigenvalues defined by (8.54) are always inside the unity circle.

8. **Testing the Kalman Filter before practical use**: As with every model-based algorithm (as controllers) you should test your Kalman Filter with a *simulated process* before applying it to the real system. You can implement a simulator in e.g. LabVIEW or MATLAB/Simulink since you already have a model (the Kalman Filter is model-based).

   Firstly, you should test the Kalman Filter with the *nominal model* in the simulator, including process and measurement noise. This is the model that you are basing the Kalman Filter on. Secondly, you should *introduce some reasonable model errors* by making the simulator model somewhat different from the Kalman Filter model, and observe if the Kalman Filter still produces usable estimates.

9. **Predictor type Kalman Filter.** In the *predictor type of the Kalman Filter* there is only one formula for the calculation of the state estimate:

$$x_{est}(k+1) = Ax_{est}(k) + K[y(k) - Cx_{est}(k)] \tag{8.55}$$

   Thus, there is no distinction between the predicted state estimate and the corrected estimate (it the same variable). (Actually, this is the original version of the Kalman Filter[7].) The predictor type Kalman Filter has the drawback that there is a time delay of one time step between the measurement $y(k)$ and the calculated state estimate $x_{est}(k+1)$.

8.4 Estimating parameters and disturbances with Kalman Filter

In many applications the Kalman Filter is used to estimate parameters and/or disturbances in addition to the "ordinary" state variables. One

example is dynamic positioning systems for ship position control where the Kalman Filter is used to estimate environmental forces acting on the ship (these estimates are used in the controller as a feedforward control signal).

These parameters and/or disturbances *must be represented as state variables*. They represent additional state variables. The original state vector is *augmented* with these new state variables which we may denote the *augmentative states*. The Kalman Filter is used to estimate the augmented state vector which consists of both the original state variables and the augmentative state variables. But how can you model these augmentative state variables? The augmentative model must be in the form of a difference equation because that is the model form of state variables are defined. To set up an augmentative model you must make an assumption about the behaviour of the augmentative state. Let us look at some augmentative models.

- **Augmentative state is (almost) constant**: The most common augmentative model is based on the assumption that the augmentative state variable x_a is slowly varying, almost constant. The corresponding differential equation is

$$\dot{x}_a(t) = 0 \tag{8.56}$$

  Discretizing this differential equation with the Euler Forward method gives

$$x_a(k+1) = x_a(k) \tag{8.57}$$

  which is a difference equation ready for Kalman Filter algorithm. It is however common to assume that the state is driven by some noise, hence the augmentative model become:

$$x_a(k+1) = x_a(k) + w_a(k) \tag{8.58}$$

  where w_a is white process noise with assumed auto-covariance on the form $R_{w_a}(L) = Q_a\delta(L)$. As pointed out in Section 8.3, the variance Q_a can be used as a tuning parameter of the Kalman Filter.

- **Augmentative state has (almost) constant rate**: The corresponding differential equation is

$$\ddot{x}_a = 0 \tag{8.59}$$

  or, in state space form, with $x_{a_1} \equiv x_a$,

$$\dot{x}_{a_1} = x_{a_2} \tag{8.60}$$

$$\dot{x}_{a_2} = 0 \tag{8.61}$$

where x_{a_2} is another augmentative state variable. Applying Euler Forward discretization with sampling interval h [sec] to (8.60) – (8.61) and including white process noise to the resulting difference equations gives

$$x_{a_1}(k+1) = x_{a_1}(k) + hx_{a_2}(k) + w_{a_1}(k) \tag{8.62}$$

$$x_{a_2}(k+1) = x_{a_2}(k) + w_{a_2}(k) \tag{8.63}$$

Once you have defined the augmented model, you can design and implement the Kalman Filter in the usual way. The Kalman Filter then estimates both the original states and the augmentative states.

The following example shows how the state augementation can be done in a practical application. The example also shows how to use functions in LabVIEW and in MATLAB to calculate the steady state Kalman Filter gain.

Example 19 ***Kalman Filter***

Figure 8.3 shows a liquid tank. We will design a steady state Kalman Filter to estimate the outflow F_{out}. The level h is measured.

Mass balance of the liquid in the tank is (mass is ρAh)

$$\rho A_{tank}\dot{h}(t) = \rho K_p u - \rho F_{out}(t) \tag{8.64}$$

$$= \rho K_p u - \rho F_{out}(t) \tag{8.65}$$

After cancelling the density ρ the model is

$$\dot{h}(t) = \frac{1}{A_{tank}}\left[K_p u - F_{out}(t)\right] \tag{8.66}$$

We assume that the unknown outflow is slowly changing, almost constant. We define the following augmentative model:

$$\dot{F}_{out}(t) = 0 \tag{8.67}$$

The model of the system is given by (8.66) – (8.67). Although it is not necessary, it is convenient to rename the state variables using standard names. So we define

$$x_1 = h \tag{8.68}$$

$$x_2 = F_{out} \tag{8.69}$$

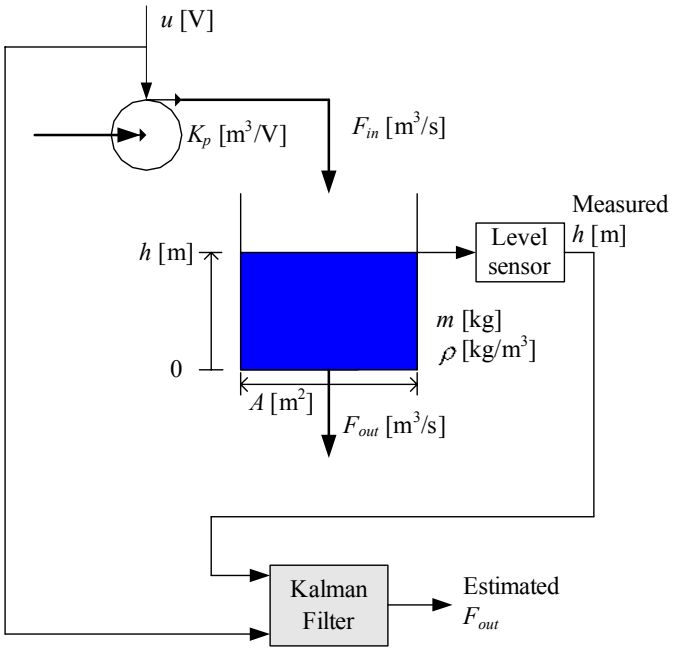

Figure 8.3: Example 19: Liquid tank

The model (8.66) – (8.67) is now

$$\dot{x}_1(t) = \frac{1}{A_{tank}} \left[K_p u(t) - x_2(t) \right] \tag{8.70}$$

$$\dot{x}_2(t) = 0 \tag{8.71}$$

Applying Euler Forward discretization with time step T and including white disturbance noise in the resulting difference equations yields

$$x_1(k+1) = \underbrace{x_1(k) + \frac{T}{A_{tank}} \left[K_p u(k) - x_2(k) \right]}_{f_1(\cdot)} + w_1(k) \tag{8.72}$$

$$x_2(k+1) = \underbrace{x_2(k)}_{f_2(\cdot)} + w_2(k) \tag{8.73}$$

or

$$x(k+1) = f\left[x(k), u(k) \right] + w(k) \tag{8.74}$$

w_1 and w_2 are independent (uncorrelated) white process noises with assumed variances $R_{w_1}(L) = Q_1 \delta(L)$ and $R_{w_2}(L) = Q_2 \delta(L)$ respectively. Here, Q_1 and Q_2 are variances. The multivariable noise model is then

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{8.75}$$

with auto-covariance

$$R_w(L) = Q\delta(L) \tag{8.76}$$

where

$$Q = \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \tag{8.77}$$

Assuming that the level x_1 is measured, we add the following measurement equation to the state space model:

$$y(k) = g\left[x_p(k), u(k)\right] + v(k) = x_1(k) + v(k) \tag{8.78}$$

where v is white measurement noise with assumed variance

$$R_v(L) = R\delta(L) \tag{8.79}$$

where R is the measurement variance.

The following numerical values are used:

$$\text{Sampling time: } T = 0.1 \text{ s} \tag{8.80}$$

$$A_{tank} = 0.1 \text{ m}^2 \tag{8.81}$$

$$K_p = 0.001 \text{ (m}^3\text{/s)/V} \tag{8.82}$$

$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \text{ (initally, may be adjusted)} \tag{8.83}$$

$$R = 0.0001 \text{ m}^2 \text{ (Gaussian white noise)} \tag{8.84}$$

We will set the initial estimates as follows:

$$x_{1_p}(0) = x_1(0) = y(0) \text{ (from the sensor)} \tag{8.85}$$

$$x_{2_p}(0) = 0 \text{ (assuming no information about initial value} \tag{8.86}$$

The Kalman Filter algorithm is as follows: The predicted level measurement is calculated according to (8.35):

$$y_p(k) = g\left[x_p(k), u(k)\right] = x_{1_p}(k) \tag{8.87}$$

with initial value as given by (8.85). The innovation variable is calculated according to (8.36), where y is the level measurement:

$$e(k) = y(k) - y_p(k) \tag{8.88}$$

The corrected state estimate is calculated according to (8.37):

$$x_c(k) = x_p(k) + Ke(k) \tag{8.89}$$

or, in detail:

$$\begin{bmatrix} x_{1_c}(k) \\ x_{2_c}(k) \end{bmatrix} = \begin{bmatrix} x_{1_p}(k) \\ x_{2_p}(k) \end{bmatrix} + \underbrace{\begin{bmatrix} K_{11} \\ K_{21} \end{bmatrix}}_{K} e(k) \tag{8.90}$$

This is the applied esimate!

The predicted state estimate for the next time step, $x_p(k+1)$, is calculated according to (8.38):

$$x_p(k+1) = f[x_c(k),\, u(k)] \tag{8.91}$$

or, in detail:

$$\begin{bmatrix} x_{1_p}(k+1) \\ x_{2_p}(k+1) \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} x_{1_c}(k) + \frac{T}{A_{tank}} \left[K_p u(k) - x_{2_c}(k)\right] \\ x_{2_c}(k) \end{bmatrix} \tag{8.92}$$

To calculate the steady state Kalman Filter gain K_s the following information is needed, cf. Figure 8.2:

$$A = \left.\frac{\partial f(\cdot)}{\partial x}\right|_{x_p(k),\, u(k)} \tag{8.93}$$

$$= \left.\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}\right|_{x_p(k),\, u(k)} \tag{8.94}$$

$$= \begin{bmatrix} 1 & -\frac{T}{A_{tank}} \\ 0 & 1 \end{bmatrix} \tag{8.95}$$

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2 \text{ (identity matrix)} \tag{8.96}$$

$$C = \left.\frac{\partial g(\cdot)}{\partial x}\right|_{x_p(k),\, u(k)} \tag{8.97}$$

$$= \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{8.98}$$

$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \text{ (initially, may be adjusted)} \tag{8.99}$$

$$R = 0.001 \text{ m}^2 \tag{8.100}$$

Figure 8.4 shows the front panel of a LabVIEW simulator of this example. The outflow $F_{out} = x_2$ was changed during the simulation, and the Kalman Filter estimates the correct steady state value (the estimate is however

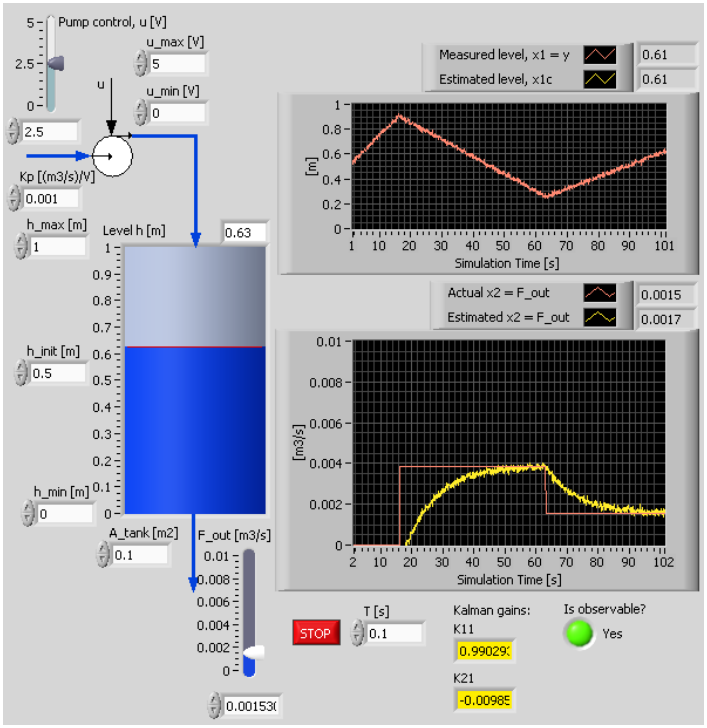

Figure 8.4: Example 19: Front panel of a LabVIEW simulator of this example

noisy). During the simulations, I found that (8.99) gave too noise estimate of $x_2 = F_{out}$. I ended up with

$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 10^{-6} \end{bmatrix} \tag{8.101}$$

as a proper value.

Figure 8.5 shows how the steady state Kalman Filter gain K_s is calculated using the **Kalman Gain** function. The figure also shows how to check for observability with the **Observability Matrix** function. Figure 8.6 shows the implementation of the Kalman Filter equations in a Formula Node.

Below is MATLAB code that calculates the steady state Kalman Filter gain. It is calculated with the **dlqe**[4] function (in Control System Toolbox).

```
A=[1,-1;0,1]
G=[1,0;0,1]
C=[1,0]
```

[4] dlqe = Discrete-time linear quadratic estimator.

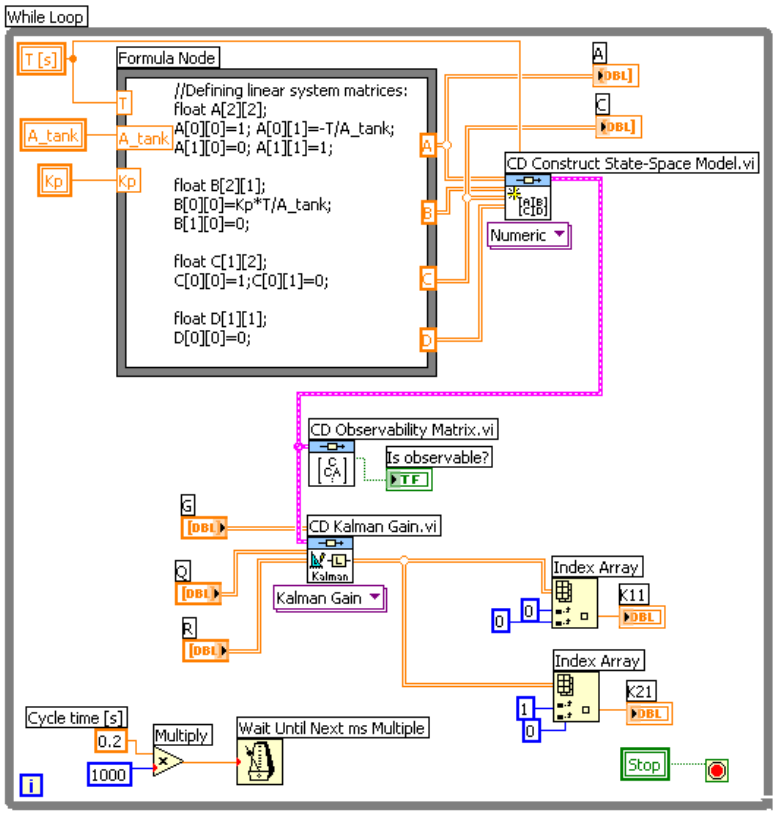

Figure 8.5: Example 19: Calculation of the steady state Kalman Filter gain with the Kalman Gain function, and checking for observability with the Observability Matrix function.

```
Q=[0.01,0;0,1e-6]
R=[0.0001];
[K,Pp,Pc,E] = dlqe(A,G,C,Q,R)
```

`K` is the steady state Kalman Filter gain. (`Pp` and `Pc` are steady state estimation error auto-covariances, cf. (8.45) and (8.44). `E` is a vector containing the eigenvalues of the Kalman Filter, cf. (8.54).)

MATLAB answers

```
K =
0.9903
-0.0099
```

Which is the same as calculated by the LabVIEW function **Kalman Gain**, cf. Figure 8.5.

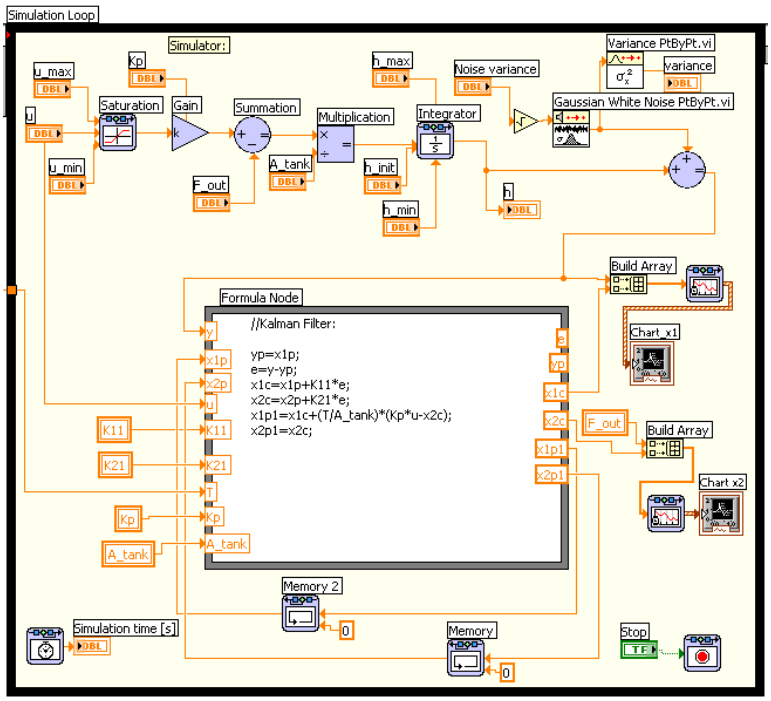

Figure 8.6: Example 19: Implementation of the Kalman Filter equations in a Formula Node. (The Kalman Filter gain is fetched from the While loop in the Block diagram using local variables.)

[End of Example 19]

8.5 State estimators for deterministic systems: Observers

Assume that the system (process) for which we are to calculate the state estimate has *no process disturbances*:

$$w(k) \equiv 0 \tag{8.102}$$

and *no measurement noise signals*:

$$v(k) \equiv 0 \tag{8.103}$$

Then the system is not a stochastic system any more. In stead we can say is is *non-stochastic or deterministic*. An *observer* is a state estimator for a deterministic system. It is common to use the same estimator formulas in

an observer as in the Kalman Filter, namely (8.35), (8.36), (8.37) and (8.38). For convenience these formulas are repeated here:

Predicted measurement estimate:

$$y_p(k) = g\left[x_p(k)\right] \tag{8.104}$$

Innovation variable:

$$e(k) = y(k) - y_p(k) \tag{8.105}$$

Corrected state estimate:

$$\underline{x_c(k) = x_p(k) + Ke(k)} \tag{8.106}$$

Predicted state estimate:

$$x_p(k+1) = f[x_c(k),\, u(k)] \tag{8.107}$$

The *estimator gain* K (corresponding to the Kalman Filter gain in the stochastic case) is calculated from a specification of the eigenvalues of the error-model of the observer. The error-model is given by (8.51) with $w(k) = 0$ and $v(k) = 0$, hence:

Error-model of observers:

$$e_{x_c}(k+1) = \underbrace{(I - KC)\, A}_{A_e} e_{x_c}(k) = A_e e_{x_c}(k) \tag{8.108}$$

where A_e is the transition matrix of the error-model. The error-model shows how the state estimation error develops as a function of time from an initial error $e_{x_c}(0)$. *If the error-model is asymptotically stable, the steady-state estimation errors will be zero* (because (8.108) is an autonomous system.) The dynamic behaviour of the estimation errors is given by the eigenvalues of the transition matrix A_e (8.108).

But how to calculate a proper value of the estimator gain K? We can no longer use the formulas (8.43) – (8.45). In stead, K can be calculated from *the specified eigenvalues* $\{z_1, z_2, \ldots, z_n\}$ of A_e:

$$\text{eig}\left[A_e\right] = \text{eig}\left[(I - KC)\, A\right] = \{z_1, z_2, \ldots, z_n\} \tag{8.109}$$

As is known from mathematics, the eigenvalues are the λ-roots of the characteristic equation:

$$\det\left[zI - (I - KC)\, A\right] = (z - z_1)(z - z_2)\cdots(z - z_n) = 0 \tag{8.110}$$

In (8.110) all terms except K are known or specified. To actually calculate K from (8.109) you should use computer tools as Matlab or LabVIEW. In

Matlab you can use the function **place** (in Control System Toobox). In LabVIEW you can use the function **CD Place.vi** (in Control Design Toolkit), and in MathScript (LabVIEW) you can use **place**. But how to specify the eigenvalues $\{z_i\}$? One possibility is to specify discrete-time Butterworth eigenvalues $\{z_i\}$. Alternatively you can specify continuous-time eigenvalues $\{s_i\}$ which you transform to discrete-time eigenvalues using the transformation

$$z_i = e^{s_i h} \tag{8.111}$$

where h [s] is the time step.

Example 20 *Calculating the observer gain K*

The following MathScript-script or Matlab-script calculates the estimator gain K using the **place** function. Note that **place** calculates the gain K_1 so that the eigenvalues of the matrix $(A_1 - B_1 K_1)$ are as specified. **place** is used as follows:

K=place(A,B,z)

But we need to calculate K so that the eigenvalues of $(I - KC)\,A = A - KCA$ are as specified. The eigenvalues of $A - KCA$ are the same as the eigenvalues of

$$(A - KCA)^T = A^T - (AC)^T K^T \tag{8.112}$$

Therefore we must use **place** as follows:

```
K1=place(A',(C*A)',z);
K=K1'
```

Here is the script:

```
h=0.05; %Time step
s=[-2+2j, -2-2j]'; %Specified s-eigenvalues
z=exp(s*h); %Corresponding z-eigenvalues
A = [1,0.05;0,0.95]; %Transition matrix
B = [0;0.05]; %Input matrix
C = [1,0]; %Output matrix
D = [0]; %Direct-output matrix
K1=place(A',(C*A)',z); %Calculating gain K1 using place
K=K1' %Transposing K
```

The result is

```
K =
0.13818
0.22376
```

The function **CD Place.vi** on the Control Design Toolkit palette in LabVIEW can also be used to calculate the estimator gain K. Figure 8.7 shows the front panel and Figure 8.8 shows the block diagram of a LabVIEW program that calculates K for the same system as above. The same K is obtained.

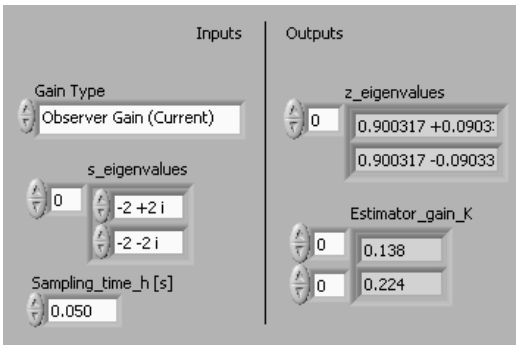

Figure 8.7: Example 20: Front panel of the LabVIEW program to calculate the estimator gain K

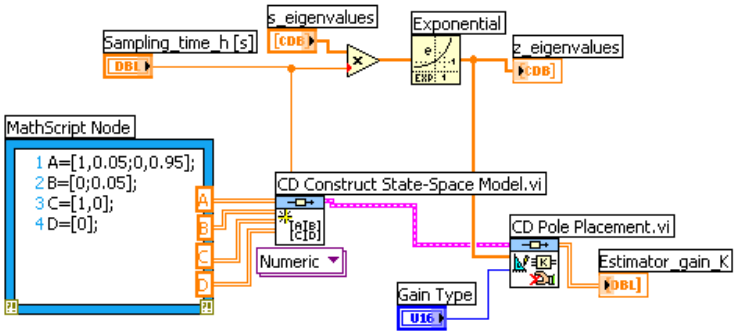

Figure 8.8: Example 20: Block diagram of the LabVIEW program to calculate the estimator gain K

[End of Example 20]

8.6 Kalman Filter or Observer?

Now you have seen two ways to calculate the estimator gain K of a state estimator:

- K is the Kalman Gain in a *Kalman Filter*
- K is the estimator gain in an *observer*

The formulas for calculating the state estimate are the same in both cases:

- Kalman Filter: (8.35), (8.36), (8.37), (8.38).
- Observer: (8.104), (8.105), (8.106), (8.107).

The observer may seem simpler than the Kalman Filter, but there is a potential danger in using the observer: The observer does not calculate any optimal state estimate in the least mean square sense, and hence the state estimate may become too noisy due to the inevitable measurement noise if you specify too fast error models dynamics (i.e. too large absolute value of the error-model eigenvalues). This is because fast dynamics requires a large estimator gain K, causing the measurement noise to give large influence on the state estimates. In my opinion, the Kalman Filter gives a better approach to state estimation because it is easier and more intuitive to tune the filter in terms of process and measurement noise variances than in terms of eigenvalues of the error-model.