Predictive Control
Laboratory Experiment 2
Direct Adaptive Control Structures

Department of Automatic Control
Lund Institute of Technology

1. Introduction

The purpose of this laboratory exercise is to design different types of direct adaptive controllers. In the first part of the laboratory, a DC-servo will be controlled. In particular, the case of direct self tuning controllers and iterative learning controllers (ILC) will be considered. In the last part of the laboratory, an adaptive minimum variance controller will be designed to cope with time varying disturbances.

You will find **7** exercises, labeled ***Preparations***, that are supposed to be solved before the laboratory session. Some of them are simulation exercises that need files from the course homepage at:

http://www.control.lth.se/course/FRTN15/

2. Control of the DC-servo

The Process We will design controllers for a process that consists of a DC-servo with a flywheel that will be controlled to follow a desired angular position.

The process is shown in Figure 1. It is a DC-servo that should be controlled to follow a desired angular position. The process interface is customized

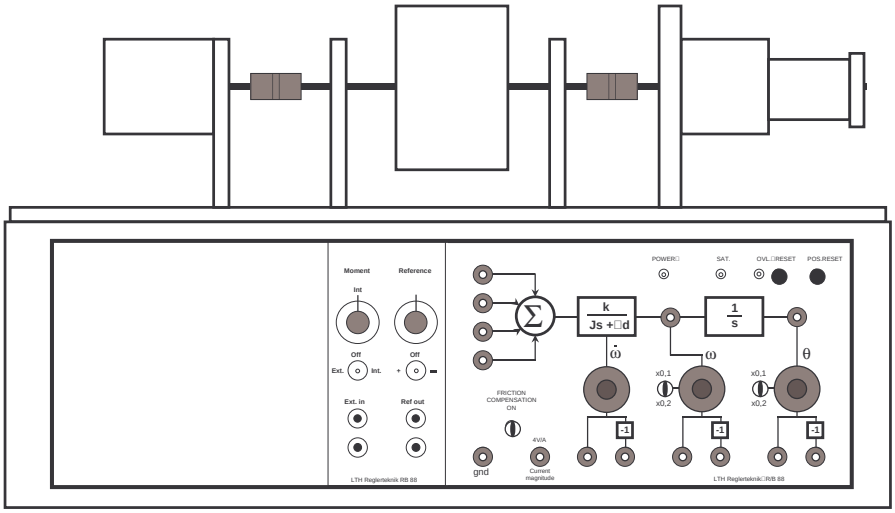

Figure 1 Front panel of the DC-servo.

to fit the AD- and DA-converters on the computer. All signals are in the interval $\pm$ 10 V. The position y is given by

$$J\ddot{y} = -d\dot{y} + ku + v \tag{1}$$

where u is the control signal and v is an optional disturbance that can affect the system. The disturbance is activated by a switch on the process front panel. Process variations on the damping d and the moment of inertia J may be introduced using the potentiometers on the front panel together with local feedback.

The nominal continuous time transfer function from u to y is

$$G(s) = \frac{11.2}{s(s+0.12)} \tag{2}$$

The corresponding pulse transfer operator is on the form

$$H(q) = \frac{B(q)}{A(q)} = \frac{b_1 q + b_2}{q^2 + a_1 q + a_2} \tag{3}$$

(*Hint:* Use Matlab to obtain the numerical values in the pulse transfer function.)

The closed-loop characteristic polynomials $A_m(q)$ and $A_o(q)$ are conveniently defined as the discrete-time counterparts to the continuous time polynomials

$$A_{mc}(s) = s^2 + 2\zeta_m \omega_m s + \omega_m^2 \tag{4}$$
$$A_{oc}(s) = s + \omega_0 \tag{5}$$

We will use the sampling interval $T_s = 0.1$ s here. With fixed relative damping, e.g, $\zeta_m = 0.7$, the closed looped system is parameterized by ω_m and ω_o. As a *nominal design*, we choose $\omega_m = 5$ and $\omega_o = 7$.

The Controller The structure of the used controllers is fixed to that of PID-type controllers, where the control signal is given by the expression:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int e(t)\mathrm{d}t + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right),$$

where $e(t)$ is the control error.

One way to cope with parameter uncertainties in the process is by using adaptive algorithms in the control structure. In this laboratory exercise we will examine two ways of adapting a PID-type control structure. First, a direct adaptive control strategy is used for online adaptation of the control parameters. This is the so called Self-TUning PID controller (STUPID). Second, an ILC scheme is used in order to obtain improved tracking performance with a fixed parameter PID.

2.1 Fixed PID control

At first, try to control the angular position using a fixed PID controller. You can test the robustness of your tuning by changing the process parameters.

Preparation 1 Discretize a PID controller using finite difference approximation (use backward difference for the derivative and forward difference for the integral part) and determine the resulting pulse transfer function. What are the orders of the numerator and denominator?
Hint:

$$\text{Forward difference: } s \approx \frac{q-1}{h}, \quad \text{Backward difference: } s \approx \frac{q-1}{qh}$$

Exercise 1 Using the simulated process, tune a PID controller with an acceptable control performance (damped response and bandwidth of approximately 5 rad/s). What are the parameters K, T_i, T_d? Test the robustness of your tuning by changing the process parameters during the simulation.

Exercise 2 When you have found a controller that works well in simulations, you can try it on the real process. Vary the process parameters (e.g., increase the moment of inertia of the plant) using connections on the process panel. How does the control performance change?

2.2 Direct Adaptive Control

This section describes a method for online adaptation of the parameters for a PID-type controller. The approach taken is that of a Direct Self Tuning Regulator.

With a PID controller one can make an arbitrary pole placement for a system of order no greater than 2. This assumption holds for the DC-servo process considered.

Use the design method for a Direct Self Tuning Regulator proposed in the text book. By imposing that the controller should contain integral action and considering a plant of second order, one obtains a PID-type structure.

This is the so called Self-TUning PID (STUPID). Following the steps in the text-book, the underlying Diophantine equation is:

$$AR + BS = B^+ A_o A_m, \quad \text{with} \quad B = B^+ B^- = B^+ b_0, \quad R = \underbrace{R' B^+}_{R^+} \underbrace{(q-1)}_{\Delta}$$

The degree conditions are:

$$\deg A_o = \deg A - \deg B^+ - 1 + 1 = \deg A - \deg B^+ = d_0 = 1,$$
$$\deg R = \deg S = 2 \rightarrow R' = 1$$

Thus,

$$A_o A_m = AR'\Delta + B^- S \;|y$$
$$A_o A_m y = \Delta B u + b_0 S y$$

Using backward-shift operator:

$$A_o A_m y = \Delta B u + b_0 S y \;|q^{-(n+d_0)}$$
$$A_o^* A_m^* q^{d_0} y = \Delta^* B^* u + b_0 S^* y$$

In order to fulfill the steady-state condition imposed by the Diophantine equation, S is parameterized as:

$$b_0 S^* = A_m(1) A_0(1) + \Delta^* S'^*$$

Preparation 2 Derive the linear regression model for the Direct Self-Tuning PID and the resulting control signal (use feedforward term also).

Preparation 3 Since the controller output is limited by a saturation nonlinearity, the integrator could give rise to the wind-up phenomena. Introduce the anti-windup measure with the anti-windup observer polynomial A_{ow}. Specify the order of the anti-windup polynomial.

Exercise 3 Simulate the Self-Tuning PID and find a suitable parameter tuning. How does the control signal behave? Can you explain? Change the process parameters during simulation and observe the behavior of your controller.

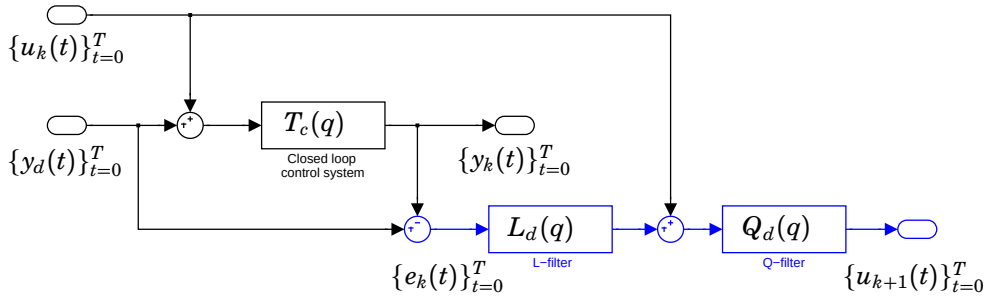

Figure 2 ILC setup: The update of the correction signal sequence $\{u_{k+1}\}$ is made in batch after each run.

Exercise 4 The relative degree of the process d_0, enters as a design parameter in our control system. Investigate the effect of this parameter in the simulated model.

Exercise 5 Test your best controller setup on the real process. Try to vary the process parameters (e.g. vary the moment of inertia of the plant). How does the controller behave? How much variation in the parameters can you handle? Investigate the effect of the forgetting factor λ in your controller.

2.3 Iterative Learning Control — ILC

In this part we will examine another method of improving the reference following for repeated tasks, called Iterative Learning Control (ILC). Errors in the reference following can have many causes; unmodeled dynamics, poorly tuned control parameters, different kinds of disturbances like load disturbances, measurement noise, friction etc. If the system response to a certain reference signal will be roughly the same whenever it is repeated (*i.e.,* a time-invariant response) ILC can be used to decrease the control error. However, if stochastic disturbances dominate or if the dynamics vary from one run to another, ILC might not help.

The method is in short: Apply the desired reference signal sequence $\{\mathbf{y_d}\}$ to the system and record the whole error sequence $\{\mathbf{e_k}\} = \{\mathbf{y_d}\} - \{\mathbf{y_k}\}$, see Fig. 2.

Based on the error sequence, modify the control signal $\{\mathbf{u_{k+1}}\}$ to improve the response for the next iteration. Repeat this until the deviation from the reference is acceptable (*i.e.,* low enough).

Note that the ILC-correction $\{\mathbf{u_k}\}$ is pre-determined and applied in "open-loop" during one run, but is updated from feedback information between the runs (batch). Compare the different time-scales between ordinary feedback, adaptive parameter updates, and Iterative Learning Control.

We have seen different kinds of update laws for Iterative Learning Control in the lectures. Here we will try with a version of the so called *heuristic approach*.

$$\{\mathbf{u_{k+1}}\} = \mathbf{Q_d}(\{\mathbf{u_k}\} + \mathbf{L_d}\{\mathbf{e_k}\})$$

where $\{\mathbf{e_k}\}$ is the sequence of errors from previous run, $\{\mathbf{u_{k+1}}\}$ is the sequence of new ILC-corrections, Q_d and L_d are linear discrete time filters.

Note: We need not to do causal filtering as the filtering is made "off-line" between two runs, when we have access to the whole sequences of data.

Preparation 4 Study the Matlab script `ILC_setup.m` and make sure you know the difference between *causal* and *acausal* filtering. Try to filter a sequence $[1 : 10]$ with the filter $G_1 = z^3$ using `noncausalfilter.m` and with the filter $G_2 = 1/z^3$ using `filter.m`, respectively.

You can modify the filters Q_d and L_d by changing in the matlab-script `ILC_setup.m` and also have a look at the ILC-update in `run_ILC_iteration.m`.

Exercise 6 Try different filters in simulation (see Figure 6). Change the constant load disturbance to see if the method will be able to compensate for it.

Exercise 7 Run the controller with ILC on the real process. How will the system perform? Can you mention any reason for why the method starts to degenerate after a number of iterations?

To run experiments on the DC-servo, type

```
>>ILC_setup    % contains filter definitions
>>ILC_pidDCreal
```

and thereafter perform a number of ILC-iterations by double-clicking on the yellow sub-system "ILC-iteration".

3. Minimum-variance Control

The purpose of this section is to illustrate some properties of a direct adaptive controller on a control problem with disturbances whose characteristics are changing. We will compare its performance with non-adaptive controllers in simulations.

The Problem A common reason for using an adaptive controller is to adapt to changing disturbance patterns. A block diagram of the system we will investigate is shown in Figure 3.

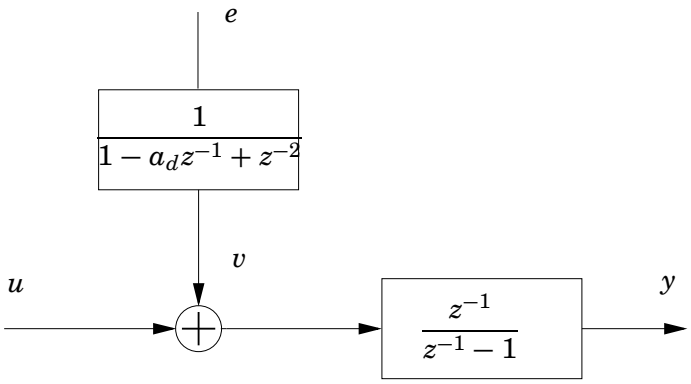

Figure 3 A blockdiagram of the process. The signal e is white noise and u is the control signal.

The process dynamics is thus very simple, an integrator with known gain. The sampling period is specified to 1 sec. The disturbance is narrow band noise. The difficulty is that the frequency of the disturbance can be quite close to the the bandwidth of the controller. With a traditional controller it is then difficult to obtain a sufficiently high gain at the frequency of the disturbance. However with a model of the disturbance it is possible to design a controller that is tuned to the disturbance. Such a controller can have a very high gain at the disturbance frequency. It will however be sensitive to the disturbance model. This difficulty can be avoided by adaptation. The lab illustrates this issue in a simple setting.

Process Model The sampled process dynamics are

$$y(t+h) = ay(t) + b(u(t) + v(t))$$

with $a = 1$ and $b = h$ and the disturbance v is generated by

$$v(t) - a_d v(t-h) + v(t-2h) = e(t)$$

with $a_d = 2\cos(\omega h)$. Nominal parameter values are $a = b = 1$, $\omega = 0.1$ and $h = 1$.

3.1 Controller Design

We will compare three different controllers for the process. The first is a conventional PI-controller which does not have enough complexity to sufficiently reduce the effects of the disturbance. The second is a minimum variance controller which relies on an accurate disturbance model to get good performance. The last controller is a direct adaptive controller based on the minimum variance controller.

Design of PI controller A PI-controller is first designed. This controller has a simple structure.

$$(q-1)u(t) = -(s_0 q + s_1)y(t)$$

With the process model

$$(q-a)y(t) = bu(t)$$

the pole placement design equation becomes

$$(q-a)(q-1) + b(s_0 q + s_1) = q(q-a_m)$$

where the observer pole is placed in the origin. The controller parameters s_0 and s_1 are found from

$$\begin{array}{ll} q^1: & -1-a+bs_0 = -a_m \\ q^0: & a+bs_1 = 0 \end{array}$$

This gives

$$\begin{aligned} s_0 &= (1+a-a_m)/b \\ s_1 &= -a/b \end{aligned}$$

and the controller becomes

$$u(t) = u(t-1) - s_0 y(t) - s_1 y(t-1)$$

Design of Minimum Variance Controller To account for the disturbance in a better way we need a more complex controller which incorporates the disturbance model in the design procedure. We will therefore design a minimum variance controller for the problem.

The noise model is given by

$$A_d(q)v(t) = v(t+2) - a_d v(t+1) + v(t) = e(t+2)$$

where $a_d = 2\cos(\omega h)$ and the process model is

$$A(q)y(t) = y(t+1) - ay(t) = b(u(t) + v(t))$$

so that

$$AA_d y(t) = bA_d u(t) + be(t+2)$$

We write this as

$$A'y = B'u + C'\bar{e}$$

with

$$\begin{aligned} A' &= AA_d = (q-a)(q^2 - a_d q + 1) \\ B' &= BA_d = b(q^2 - a_d q + 1) \\ C' &= q^3 \\ \bar{e}(t) &= be(t-1) \end{aligned}$$

The Diophantine equation we have to solve for the minimum variance controller is

$$A'R + B'S = q^3 B'$$

with the constraint that B' is a factor of R. To get a monic $R(q)$ we divide the right hand side by b. This will give us a solution, $R(q)$ and $S(q)$, that is a factor b smaller which does not affect the controller $-S(q)/R(q)$. Hence we instead solve

$$A'R + B'S = q^3 B'/b$$

with

$$R = R'B'/b$$

Cancellation of B'/b gives

$$(q-a)(q^2 - a_d q + 1)R' + bS = q^3$$

The minimum degree solution has $\deg R' = 0$ and $\deg S = 2$. We get $R' = 1$ and the coefficients in $S(q)$ can be computed from

$$\begin{array}{ll} q^2: & -a - a_d + bs_0 = 0 \\ q^1: & 1 + a_d a + bs_1 = 0 \\ q^0: & -a + bs_2 = 0 \end{array}$$

This gives

$$\begin{aligned} R(q) &= q^2 + r_1 q + r_2 = R'(q)B'(q)/b = q^2 - a_d q + 1 \\ S(q) &= s_0 q^2 + s_1 q + s_2 = ((a + a_d)q^2 + (-1 - a_d a)q + a)/b \end{aligned}$$

and the control law becomes

$$u(t) = a_d u(t-1) - u(t-2) + \frac{1}{b}\Big(-(a+a_d)y(t) + (1+a_d a)y(t-1) - ay(t-2)\Big)$$

Design of Direct Adaptive Controller The minimum variance controller was designed using the disturbance model and is therefore tuned to that model. If the characteristics of the disturbance change, the controller might perform badly. To cope with changing disturbances we will therefore design an adaptive version of the controller. If we let the design equation (with A_d canceled) operate on the output y(t) we get

$$\big(A(q)R(q) + B(q)S(q)\big)y(t) = B(q)q^3/by(t)$$

Using the input-output relation $A(q)y(t) = B(q)u(t)$ we get

$$B(q)(R(q)u(t) + S(q)y(t)) = B(q)q^3/by(t)$$

and we can cancel $B(q)$. This results in

$$R(q)u(t) + S(q)y(t) = q^3/by(t)$$

or

$$b((q^2 + r_1 q + r_2)u(t) + (s_0 q^2 + s_1 q + s_2)y(t)) = q^3 y(t)$$

The relation above holds for the correct controller and we can therefore use it to directly estimate the unknown controller parameters. The parameter b is assumed to be known so we can rewrite the relation as

$$y(t)/b - u(t-1) = \phi^T(t-1)\theta$$

where the left hand side is known and

$$\phi^T(t-1) = \begin{pmatrix} u(t-2) & u(t-3) & y(t-1) & y(t-2) & y(t-3) \end{pmatrix}$$

and

$$\theta = \begin{pmatrix} r_1 \\ r_2 \\ s_0 \\ s_1 \\ s_2 \end{pmatrix}$$

The parameter vector can now be updated using a recursive least-squares algorithm. The estimates are then used in the control law derived for the minimum variance controller.

Preparation 5 Study the minimum variance controller. Plot a Bode diagram for the controller transfer function $S(z)/R(z)$ and for the transfer function from disturbance v to the output y, i.e. $B(z)R(z)/(A(z)R(z) + B(z)S(z))$. How are they related to the disturbance characteristics?
Hint: Use the command `bode` *to plot the Bode diagrams. See e.g.* `mv_bode`.

3.2 Simulation Exercises

The three different controllers will now be investigated on the process in Figure 3. In order to compare the different controllers quantitatively we need a measurement of how well they perform. The criterion we will use is:

$$V(t) = \sum_{i=1}^{t} y^2(i)$$

Preparation 6 Investigate the performance of the PI-controller. Tune the controller to get as good performance you can. How sensitive is the PI controller to changing ω?

Preparation 7 Use the tuned minimum variance controller. Compare the performance with the PI controller. How sensitive is the tuned minimum variance controller to changing ω?

Exercise 8 Investigate the adaptive minimum variance controller. How does it behave? Do the parameters converge? How does it adapt to changing ω? What is a good value of the forgetting factor? (This is a trade-off between noise sensitivity and ability to track the varying ω.)

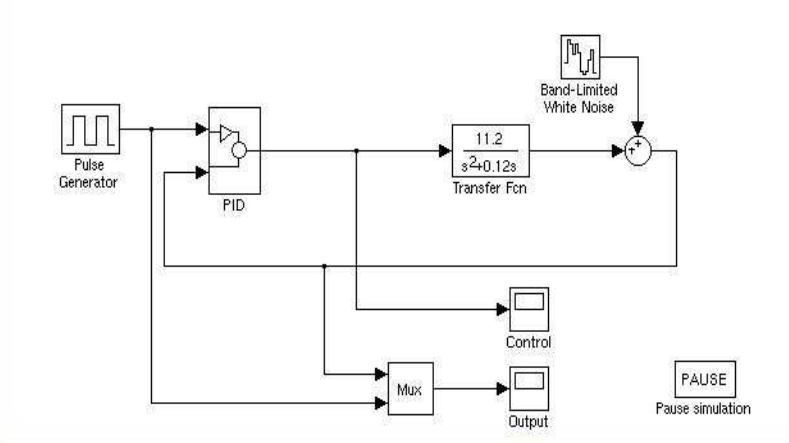

Figure 4 Simulink model *pidDCsim* used for simulation of fixed-PID-controlled DC-servo.

4. Simulation and Testing

During the lab you will make use of six Simulink models. For each control strategy you have available two Simulink models. One for simulations and one for testing on the real plant.

4.1 The DC-servo system

Simulation The fixed-PID-controlled process can be simulated using the model called *pidDCsim* (see Figure 4). You can change the controller parameter by double-clicking on the block PID. You can pause the simulation by specifying time instances in the PAUSE block. This is useful when you want to change the parameters of the plant during simulation.

The STUPID-controlled process can be simulated using the model called *stupidDCsim* (see Figure 5). You can change the controller parameter by double-clicking on the block STUPID. The following parameters are available:

- desired closed loop bandwidth (continuous-time parameter),
- closed loop damping (continuous-time parameter),
- initial estimates for $\tilde{R}$, $\tilde{S}$ (discrete-time parameter),
- forgetting factor in the Recursive Least-Square algorithm
- observer pole (discrete-time parameter),
- sampling time,
- d_0 prediction horizon in the Direct Self Tuning algorithm
- the maximum amplitude of the control signal from the controller.

You can pause the simulation by specifying time instances in the PAUSE block. This is useful when you want to change the parameters of the plant

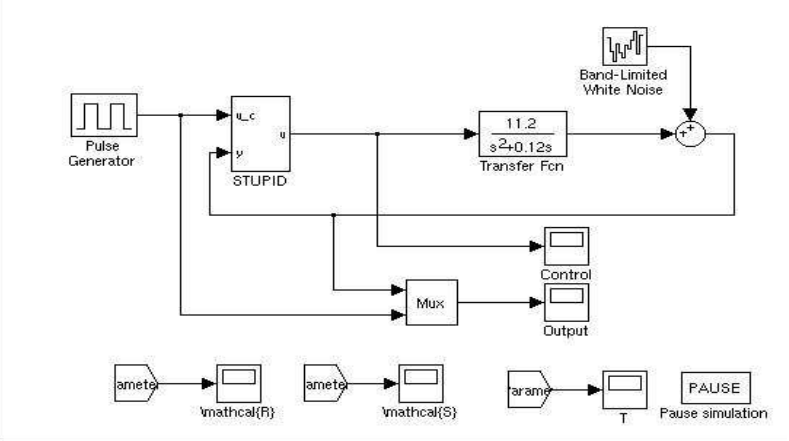

Figure 5 Simulink model *stupidDCsim* used for simulation of STUPID-controlled DC-servo.

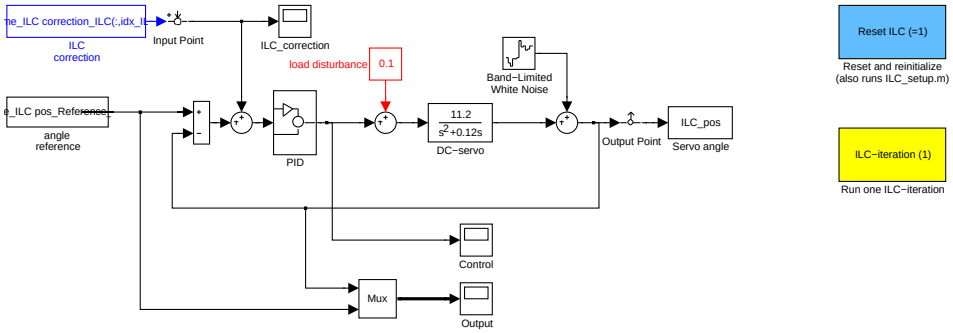

Figure 6 Simulation model `ILC_pidDCsim` using Iterative Learning Control (ILC) for improving reference following.

during simulation. The scopes denoted *mathcalR, mathcalS, T* show the parameters for $\mathcal{R}, \mathcal{S}, T$.

The ILC-controlled process can be simulated using the Simulink model called *ILC_pidDCsim* (see Figure 6). Start your simulations by typing

```
>>ILC_setup          % contains filter definitions
>>ILC_pidDCsim       % open Simulink model
```

and thereafter perform a number of ILC-iterations by double-clicking on the yellow sub-system "ILC-iteration"[1], see Figure 6.

[1]Each time you double-click on the yellow sub-system "ILC-iteration', the matlab-script `run_ILC_iteration.m` will be executed.

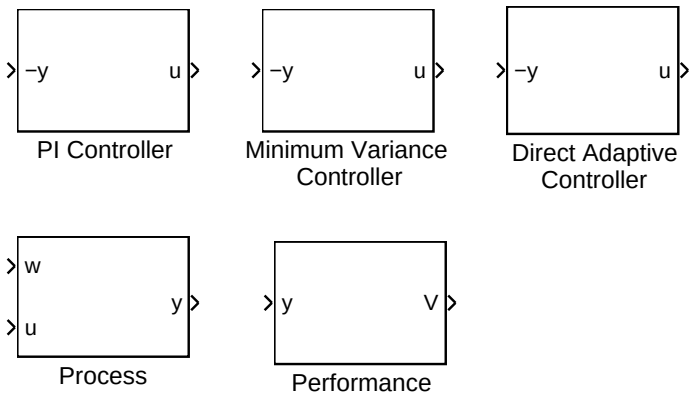

Figure 7 Library blocks

Testing your controllers on the real process The Simulink models used for testing the controllers on the real process are similar to those used for simulation. They are called *pidDCreal, stupidDCreal, ILC_pidDCreal* respectively.

Before each experiment, especially in the case of the self tuning controller, you need to reset the current position to zero on the front panel of the servo. By pressing the position reset button on the servo each time the controller is started, the initial transient due to the control error is reduced. You can induce the parameter variations in the process by using local feedback from the connectors on the panel. In particular, for changing the moment of inertia for the system you can use feedback from the angular velocity sensor.

4.2 Minimum-variance Control

This last part of the laboratory exercise will not be tested on any real process. The verifications are done using simulations solely.

- In Matlab write `labmvlib` to display the needed library. This contains the controllers, process and the performance computation blocks (see Figure 7).

- The system needed for the simulation has the structure shown in Figure 8, and is implemented in the Simulink file named `system_sim`.

We shall compare the performance of the different controllers, we are interested in running the same noise sequence in all three simulations. Implement a significant variation of the noise characteristic beginning with given time t (*Hint:* look at the natural frequency of the filter that gives the colored noise).

There are three useful functions implemented, `pi_bode` which plots the Bode diagram of the PI controller, `mv_bode` that does the same thing for the minimum variance controller and finally `da_bode` and `da_par` which plot the Bode diagram of the direct adaptive controller respectively the parameters of the direct adaptive controller.

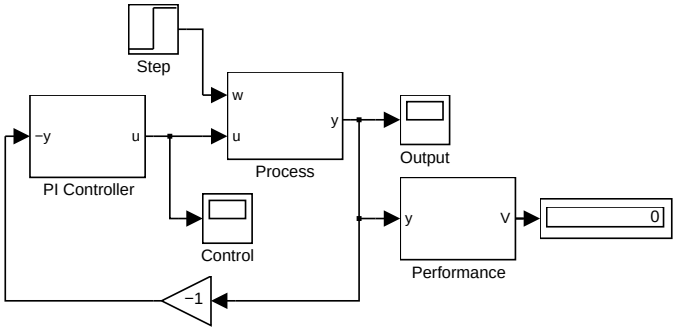

Figure 8 Structure of the Simulink file `system_sim`.

5. Matlab help

The following is a list of useful commands in Matlab and the Control Systems Toolbox:

help Matlab help, try for example *help control*

tf Create transfer function model.

bode Plot Bode diagram.

margin Plot Bode diagram with gain- and phase margins.

c2d Convert continuous-time system to discrete-time.

conv Convolution, polynomial multiplication.

poly Create polynomials with specified roots.

6. Experimental setup

The wiring diagram is shown in Figure 9. The wiring is carried out prior to the laboratory experiment because of the restricted time available.

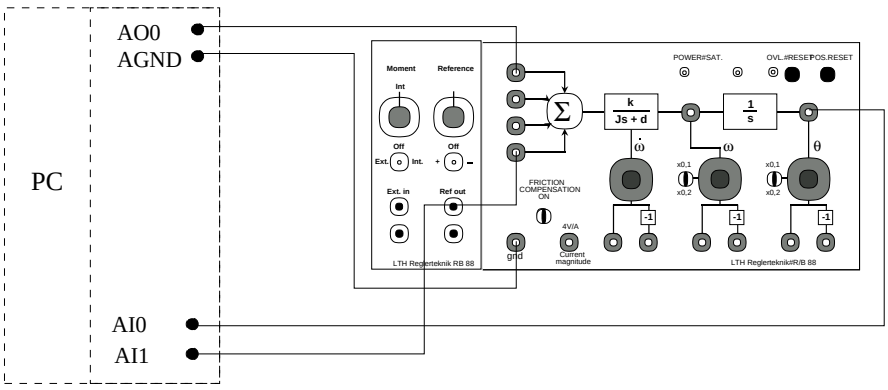

Figure 9 Wiring diagram.

Document history:
Created: September 2004 by Stefan Solyom and Anders Robertsson
Additional material by: K. J. Åström and H. Olsson