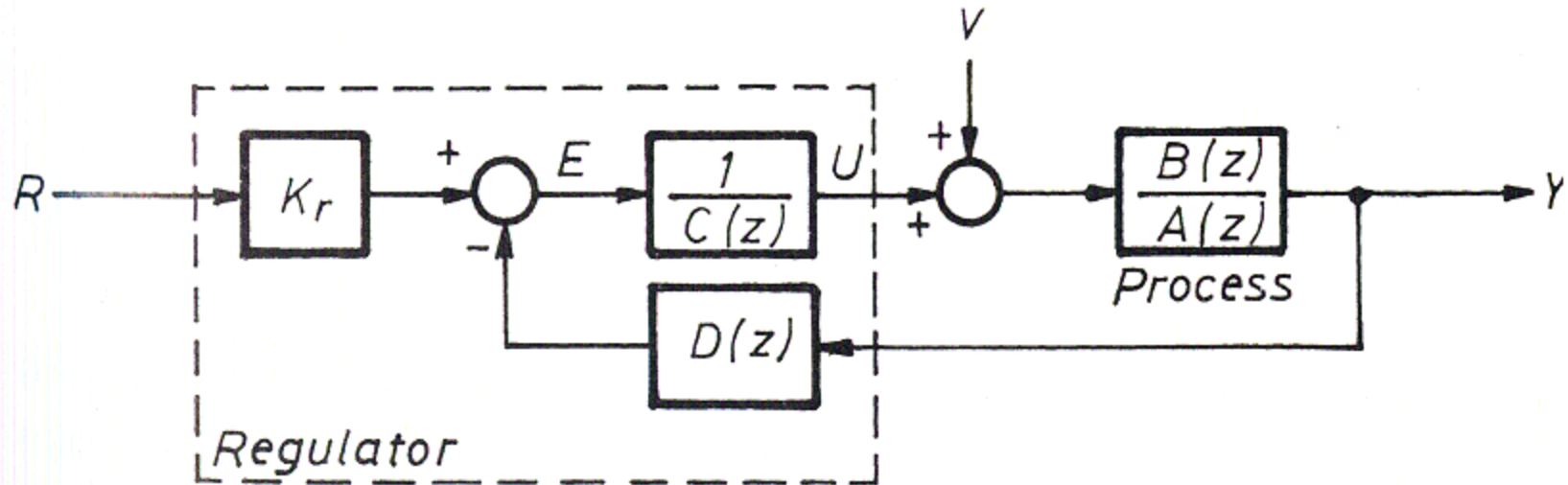


# How to create Matlab Script and Simulink Model for Designing a Pole Placement Controller

# Pole Placement Controller Design



$$H_{tot} = \frac{Y(z)}{R(z)} = \frac{K_r \cdot B(z)}{A(z)C(z) + B(z)D(z)}$$

$$P(z) = A(z)C(z) + B(z)D(z)$$

# Polplacement Equation

$$A(z)C(z) + B(z)D(z) = P(z)$$

Choose degree:  $n_p = n_a + n_b - 1$  [degree]

$$n_c = n_b - 1 \quad n_d = n_a - 1$$

Set:  $C(z) = 1 + c_1 z^{-1} + c_2 z^{-2} + \dots + c_{nc} z^{-n}$

$$D(z) = d_0 + d_1 z^{-1} + d_2 z^{-2} + \dots + d_{nd} z^{-n}$$

$$K_r = \frac{P(z=1)}{B(z=1)}$$

# Pole Placement Example

Bertil Thomas, Modern reglerteknik

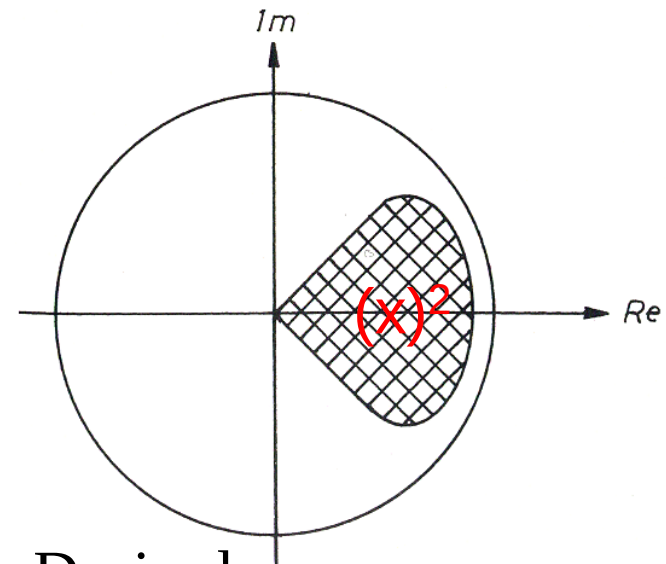
$$y(k) = 0,82y_{k-1} + 0,5u_{k-1} + 0,5u_{k-2} \quad \{Z : \}$$

$$Y = 0,82Yz^{-1} + 0,5Uz^{-1} + 0,5Uz^{-2}$$

$$H(z) = \frac{U}{Y} = \frac{0,5z^{-1} + 0,5z^{-2}}{1 - 0,82z^{-1}}$$

$$B(z) = 0,5z^{-1} + 0,5z^{-2}$$

$$A(z) = 1 - 0,82z^{-1}$$



Desired:  
dubbele pole in  $z = 0,5$

# Pole Placement Example

$$B(z) = 0,5z^{-1} + 0,5z^{-2}$$

$$A(z) = 1 - 0,82z^{-1}$$

Desired:

dubble pole in  $z = 0,5$

$$P(z) = (1 - 0,5z^{-1})^2 = 1 - z^{-1} + 0,5z^{-2} \Rightarrow n_p = 2$$

$$n_c = n_b - 1 = 2 - 1 = 1 \Rightarrow C(z) = 1 + c_1z^{-1}$$

$$n_d = n_a - 1 = 1 - 1 = 0 \Rightarrow D(z) = d_0$$

$$AC + BD = P:$$

$$(1 - 0,82z^{-1})(1 + c_1z^{-1}) + (0,5z^{-1} + 0,5z^{-2})(d_0) = 1 - z^{-1} + 0,25z^{-2}$$

# Pole Placement Example

$$\begin{aligned}
 &(1 - 0,82z^{-1})(1 + c_1z^{-1}) + (0,5z^{-1} + 0,5z^{-2})(d_0) = \\
 &= 1 + c_1z^{-1} - 0,82z^{-1} - 0,82c_1z^{-2} + 0,5d_0z^{-1} + 0,5d_0z^{-2} = \\
 &= 1 + \boxed{(c_1 - 0,82 + 0,5d_0)}z^{-1} + \boxed{(0,5d_0 - 0,82c_1)}z^{-2} \\
 &= 1 \boxed{-}z^{-1} + \boxed{0,25}z^{-2} \quad \leftarrow \text{AC+BD = P}
 \end{aligned}$$

$$c_1 - 0,82 + 0,5d_0 = -1 \quad \Leftrightarrow \quad c_1 + 0,5d_0 = -0,18$$

$$0,5d_0 - 0,82c_1 = 0,25 \quad \Leftrightarrow \quad -0,82c_1 + 0,5d_0 = 0,25$$

# Pole Placement Example

$$c_1 - 0,82 + 0,5d_0 = -1 \quad \Leftrightarrow \quad c_1 + 0,5d_0 = -0,18$$

$$0,5d_0 - 0,82c_1 = 0,25 \quad \Leftrightarrow \quad -0,82c_1 + 0,5d_0 = 0,25$$

$$\begin{pmatrix} 1 & 0,5 \\ -0,82 & 0,5 \end{pmatrix} \bullet \begin{pmatrix} c_1 \\ d_0 \end{pmatrix} = \begin{pmatrix} -0,18 \\ 0,25 \end{pmatrix}$$

$$c_1 = -0,236 \quad d_0 = 0,113$$

```
» X=[1, 0.5; -0.82, 0.5]\[-0.18, 0.25]'
```

```
X = 

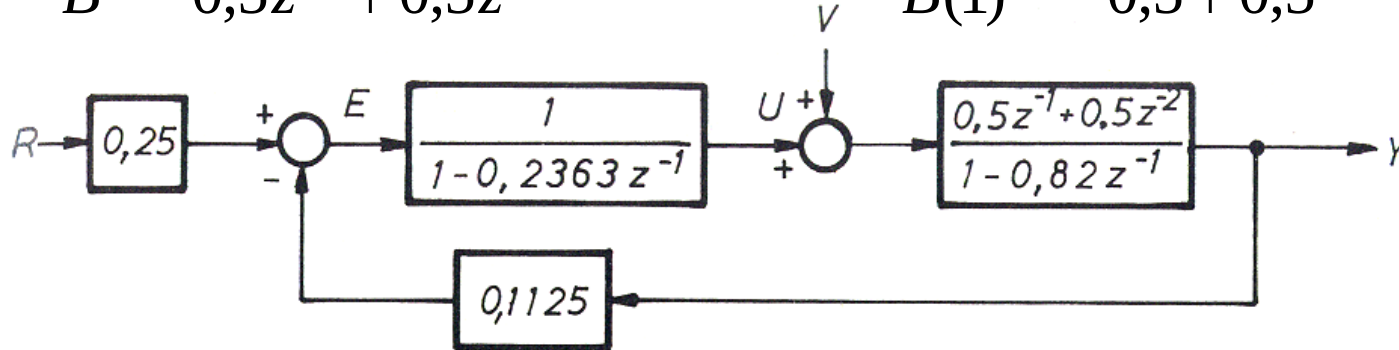
|         |                |
|---------|----------------|
| -0.2363 | c <sub>1</sub> |
| 0.1125  | d <sub>0</sub> |


```

```
»
```

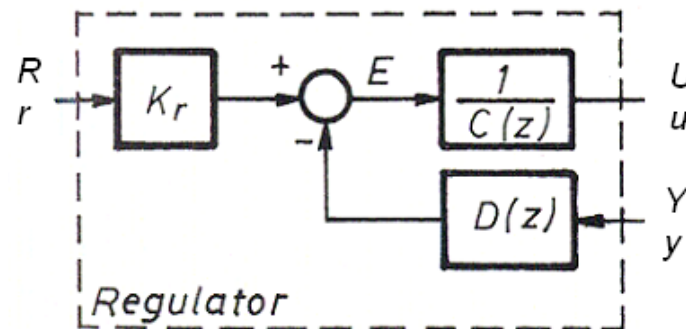
# Pole Placement Example

$$\frac{P}{B} = \frac{1 - z^{-1} + 0,25z^{-2}}{0,5z^{-1} + 0,5z^{-2}} \Rightarrow K_r = \frac{P(1)}{B(1)} = \frac{1 - 1 + 0,25}{0,5 + 0,5} = 0,25$$



$$E = R \cdot K_r - Y \cdot D$$

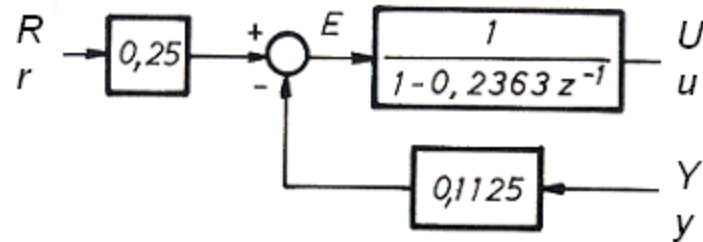
$$U = \frac{E}{C}$$



# Pole Placement Example

$$E = R \cdot K_r - Y \cdot D$$

$$U = \frac{E}{C}$$



$$E = 0,25 R - 0,1125 Y$$

$$U = \frac{E}{1 - 0,2363 z^{-1}} \Rightarrow U(1 - 0,2363 z^{-1}) = E$$

$$\Rightarrow U = E - 0,2363 U z^{-1}$$

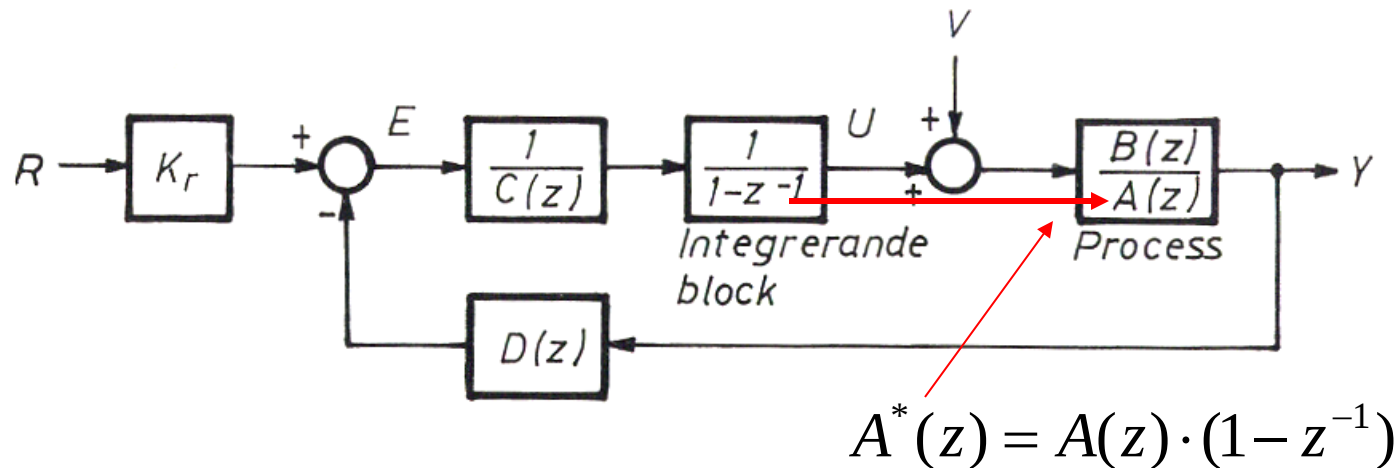
- Differens equations:

$$e_k = 0,25 r_k - 0,1125 y_k \quad u_k = e_k - 0,2363 u_{k-1}$$



# Integrating Block

To avoid residual error we need an **integrating controller**.



**During calculations** the integrator is placed in  $A(z)$ , called  $A^*(z)$ . ( = easiest )

# With Integrating Block

Searched  $P(z)$  has dubble pole  $z = 0,5$  and extra pole in origo.

$$B(z) = 0,5z^{-1} + 0,5z^{-2}$$

$$A(z) = 1 - 0,82z^{-1} \Rightarrow A^*(z) = (1 - 0,82z^{-1})(1 - z^{-1})$$

$$P(z) = (1 - 0,5z^{-1})^2 \cdot (1 - 0 \cdot z^{-1}) = 1 - z^{-1} + 0,5z^{-2} \Rightarrow n_p = 2$$

$$n_c = n_b - 1 = 2 - 1 = 1 \Rightarrow C(z) = 1 + c_1z^{-1}$$

$$n_d = n_a - 1 = 2 - 1 = 1 \Rightarrow D(z) = d_0 + d_1z^{-1}$$

$$A^*C + BD = P:$$

$$(1 - 0,82z^{-1})(1 - z^{-1})(1 + c_1z^{-1}) + (0,5z^{-1} + 0,5z^{-2})(d_0) = 1 - z^{-1} + 0,25z^{-2}$$

# With Integrating Block

$$\begin{aligned}
 (1 - 0,82z^{-1})(1 - z^{-1})(1 + c_1z^{-1}) + (0,5z^{-1} + 0,5z^{-2})(d_0) = \\
 = 1 + (c_1 - 1,82 + 0,5d_0) z^{-1} + (0,82 - 1,82c_1 + 0,5d_1 + 0,5d_0) z^{-2} + \\
 + (0,82c_1 + 0,5d_1)z^{-3} = 1 - z^{-1} + 0,25 z^{-2}
 \end{aligned}$$

$$(c_1 - 1,82 + 0,5d_0) = -1$$

$$c_1 + 0,5d_0 + 0d_1 = 0,82$$

$$(0,82 - 1,82c_1 + 0,5d_1 + 0,5d_0) = 0,25$$

$$-1,82c_1 + 0,5d_0 + 0,5d_1 = -0,57$$

$$(0,82c_1 + 0,5d_1) = 0$$

$$0,82c_1 + 0d_0 + 0,5d_1 = 0$$

» X=[1, 0.5, 0; -1.82, 0.5, 0.5; 0.82, 0, 0.5]\[0.82, -0.57, 0]'

X = 

|        |                |
|--------|----------------|
| 0.382  | c <sub>1</sub> |
| 0.876  | d <sub>0</sub> |
| -0.626 | d <sub>1</sub> |

»

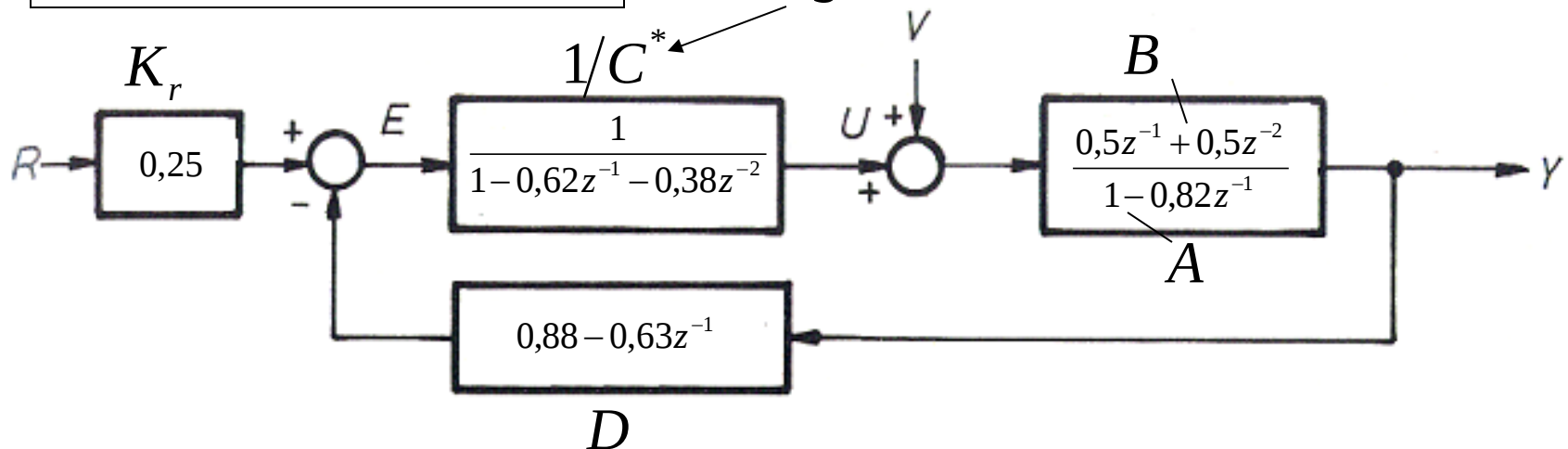
# With Integrating Block

$$\frac{P}{B} = \frac{1 - z^{-1} + 0,25z^{-2}}{0,5z^{-1} + 0,5z^{-2}} \Rightarrow K_r = \frac{P(1)}{B(1)} = \frac{1 - 1 + 0,25}{0,5 + 0,5} = 0,25$$

$$C^* = C(z) \cdot (1 - z^{-1}) = (1 - z^{-1})(1 + 0,382z^{-1}) = 1 - 0,618z^{-1} - 0,382z^{-2}$$

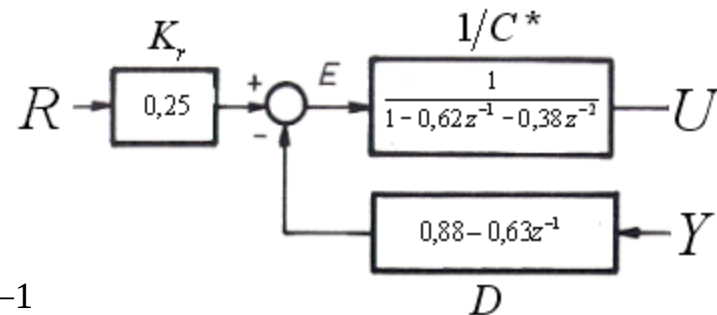
$$D(z) = 0,876 - 0,626z^{-1}$$

*Integrator now in C-block!*



# Controller Difference Equations

$$\begin{aligned} E &= R \cdot K_r - Y \cdot D \\ U &= \frac{E}{C^*} \end{aligned}$$



$$E = 0,25 R - 0,88 Y + 0,63 Y z^{-1}$$

$$U = \frac{E}{1 - 0,62 z^{-1} - 0,38 z^{-2}} \Leftrightarrow U - 0,62 U z^{-1} - 0,38 U z^{-2} = E$$

$$U = E + 0,62 U z^{-1} + 0,38 U z^{-2}$$

- Difference Equations:

$$e_k = 0,25 r_k - 0,88 y_k + 0,63 y_{k-1} \quad u_k = e_k + 0,62 u_{k-1} + 0,38 u_{k-2}$$



# Matlab Script for Lab 3



Pole placement requires extensive calculations every time you want to try new poles. This is something you do not have time to do during the lab – you have to automate calculations using a Matlab script.

When solving the preparation tasks, develop a MATLAB script to be able to quickly redo the calculations during the lab with the values of the real process.

# Matlab Script for Lab 3



```
% poleplace.m
%
% Preparation for IE1304 Lab 3: Pole placement controller.
%
%
%           +   E           U           GP
%           R -->K r-->0--->1/C(z)--->B(z)/A(z)---+---> Y
% Htot(z):           -^
%                   |
%                   +-----D(z)<-----+
%
% Process transfer function GP has two time constants
```

Start the script with a comment (%). That comment is printed when writing **help poleplace** in the Matlab command window – good practice to show what the script does.

# Matlab Script for Lab 3



Prior to the lab, we assume that process gain and time constants have the following values:

```
% Process transfer function GP
```

```
% has two time constants
```

```
Kp =3      % Change process gain if needed
```

```
T1=6       % Change first time constant if needed
```

```
T2=21      % Change second time constant if needed
```

```
s = tf('s');
```

```
GP = Kp / (1 + T1*s) / (1 + T2*s)
```

```
[GpNum, GpDen]=tfdata(GP, 'v')
```

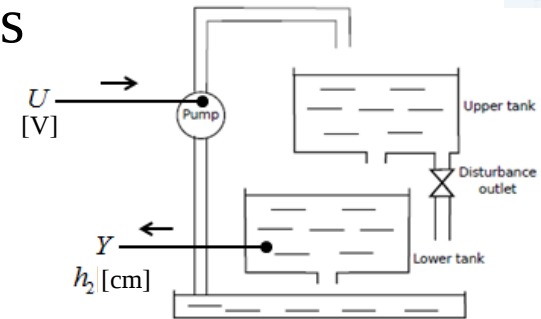


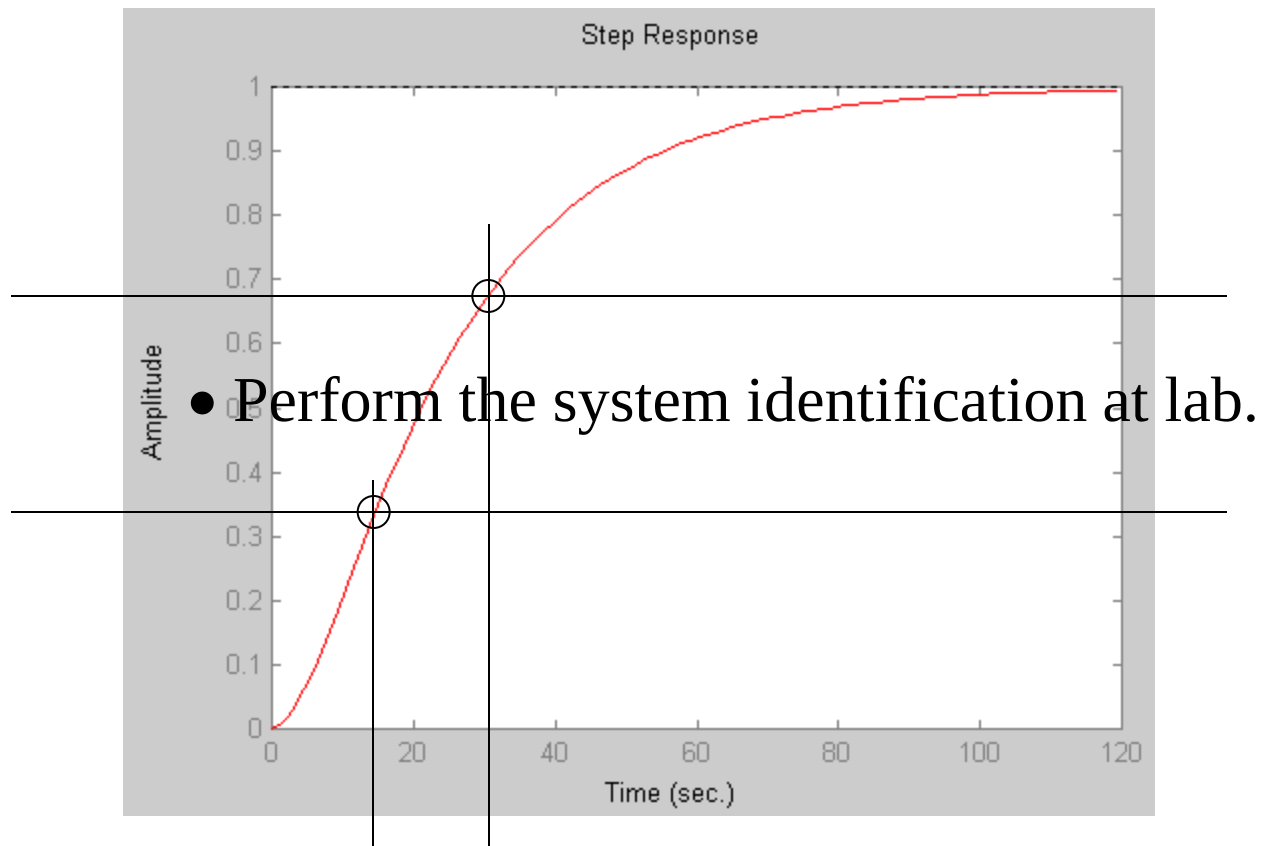
Figure 1: The controlled process

*These values must be changed to actual process values at the lab*

- Assumed values prior to lab.

$$G_P = \frac{K_P}{(1+T_1s)(1+T_2s)} = \{K_P = 3 \quad T_1 = 6 \quad T_2 = 21\} =$$

$$= \frac{3}{(1+6s)(1+21s)} = \frac{3}{126s^2 + 27s + 1}$$



# Matlab Script for Lab 3



It is common practice to insert results from the command window as commands in the script!

```
Kp = 3    % Change process gain if needed
T1 = 6    % Change first time constant if needed
T2 = 21   % Change second time constant if needed
```

```
s = tf('s');
GP = Kp / (1 + T1*s) / (1 + T2*s)
[GpNum,GpDen]=tfdata(GP, 'v')
```



```
% Default transfer function:
%              3
% -----
% 126 s^2 + 27 s + 1
```

```
% GpNum =      0      0      3
% GpDen =     126     27     1
```

# Matlab Script for Lab 3



For pole placement calculations, we need the process' discrete-time model. Matlab can print the model in positive ( $\mathbf{z}^n$ ) or negative ( $\mathbf{z}^{-n}$ ) form.

```
% Discretize the process
disp('Now discretizing')
h = 0.6; % sampling interval
HP=c2d(GP,h);
fprintf('\nHP Negative notation')
HP.variable = 'z^-1'
fprintf('\nHP Positive notation')
HP.variable = 'z'
```

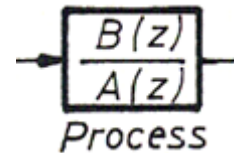


```
% Discrete transfer function:
% HP Negative notation
% Transfer function:
% 0.004107 z^-1 + 0.003935 z^-2
% -----
% 1 - 1.877 z^-1 + 0.8794 z^-2
% Sampling time (seconds): 0.6
%
% HP Positive notation
% Transfer function:
% 0.004107 z + 0.003935
% -----
% z^2 - 1.877 z + 0.8794
% Sampling time (seconds): 0.6
```

# Matlab Script for Lab 3



We need process' transfer function in negative form, split into denominator (A) and numerator (B).



```
% HP Negative notation
% Transfer function:
% 0.004107 z^-1 + 0.003935 z^-2
% -----
% 1 - 1.877 z^-1 + 0.8794 z^-2
```

```
% HP=B/A
```

```
[B,A]=tfdata(HP, 'v')
```

```
% B = [ 0 0.0041 0.0039]
% A = [ 1.0000 -1.8767 0.8794]
```

# Matlab Script for Lab 3



A and B are both of second degree and for now we use the following preliminary poles to create the pole placement polynomial:  $z = 0.7 \pm 0.2i$  and  $z = 0$ .

**% 3 poles suggestion:**

```
p1=0.7+0.2i; % Change if needed
p2=0.7-0.2i; % Change if needed
p3=0;
```

**% Prompt user for pole values**

```
% p1 = input('First pole (a+bi): ');
% p2 = input('Second pole (a-bi): ');
% display('Third pole at origo 0+0i');
```

**% Poleplacement polynomial**

```
P = poly([p1,p2,p3])
```

```
% P = [1.0000 -1.4000 0.5300 0]
```

$$P(z) =$$

$$A(z)C(z) + B(z)D(z)$$

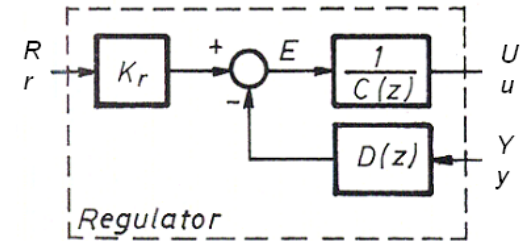
*When trying many different pole values it might be convenient to have the script prompt the user!*

# Matlab Script for Lab 3



Controller polynomials C and D:

```
% Controller polynomials C and D  
% C = 1 + c_1*z^-1  
% D = d_0 + d_1*z^-1
```



$$A(z)C(z) + B(z)D(z) = P(z)$$

Parameters **c\_1** **d\_0** och **d\_1** are calculated by identification of the controller polynomial's equivalent terms. This algebra can not be performed with Matlab (when the Maple module is missing) but may be done with "paper and pencil" or with Mathematica. When the algebra is made, Matlab can solve the equation system numerically.

***You must present the algebraic calculations at the lab!***

- Parameter names in Matlab

```
% P = [1.0000 -1.4000 0.5300 0] =  
= [P(4) P(3) P(2) P(1)] =  
= P(4)*z^-3 + P(4)*z^-2 + P(4)*z^-1 + P(4)*z^-0
```

```
% HP=B/A
```

```
[B,A]=tfdata(HP, 'v')
```

```
% B = [0 0.0041 0.0039] = [B(3) B(2) B(1)] =  
= B(3)*z^-2 B(2)*z^-1 B(1)*z^-0
```

```
% A = [1.0000 -1.8767 0.8794] = [A(3) A(2) A(1)] =  
= A(3)*z^-2 A(2)*z^-1 A(1)*z^-0
```

```
% Controller polynomials C and D
```

```
% C = [1 c_1] = 1 + c_1*z^-1
```

```
% D = [d_0 d_1 ] = d_0 + d_1*z^-1
```

Identify:  $A(z)C(z) + B(z)D(z) = P(z)$

# Matlab Script for Lab 3



Describe your *algebraic* calculations at the lab!

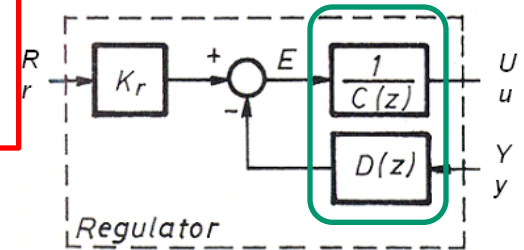


This  
Matlab  
can solve

```
left = [ 1      B(2)  0
         A(2)  B(3)  B(2)
         A(3)  0      B(3) ];
right = [P(2)-A(2), P(3)-A(3), P(4)]' ;
left \ right;
c_1 = ans(1), d_0 = ans(2), d_1 = ans(3)
```

```
% c_1 = 0.2148
% d_0 = 63.7665
% d_1 = -48.0050
```

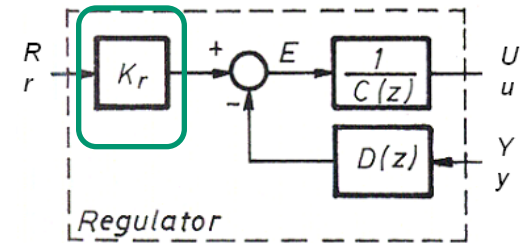
Now we have the  
parameters **c\_1**    **d\_0**  
**d\_1** calculated!



# Matlab Script for Lab 3



$$K_r = \frac{P(1)}{B(1)}$$



```
% Kr=P(1)/B(1)
```

```
Kr = polyval(P,1) / polyval(B,1)
```

```
% Kr = 16.1664
```

Now we have *all* the parameters!

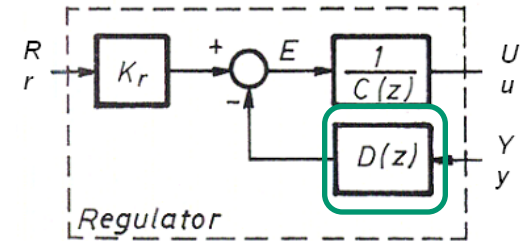
# Matlab Script for Lab 3



## % Controller blocks

```
D = [d_0  d_1 ]
D_block = tf( D,[1 0],h);
fprintf('\nBlock D positive notation')
D_block.variable = 'z'
```

```
% D =  63.7665  -48.0050
%
% Block D positive notation
% Transfer function:
% 63.77 z - 48.01
% -----
%          z
% Sampling time (seconds): 0.6
```



# Matlab Script for Lab 3



**% Controller blocks**

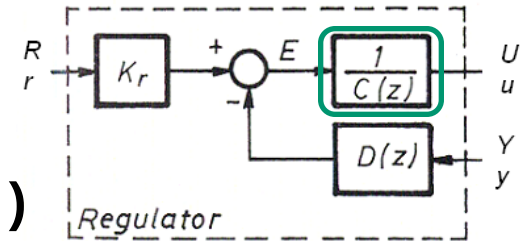
```
C = [1  c_1]
```

```
C_block = tf([1 0],C,h );
```

```
fprintf('\nBlock C positive notation')
```

```
C_block.variable = 'z'
```

```
% C = 1.0000    0.2148
%
% Block C positive notation
% Transfer function:
%      z
% -----
% z + 0.2148
% Sampling time (seconds): 0.6
```



# Matlab Script for Lab 3



```
% Total transfer function
```

```
%      Kr*B
```

```
% Htot = -----
```

```
%      A*C + B*D
```

```
% Transfer function of controlled system:
```

$$H_{tot} = \frac{K_r \cdot B(z)}{A(z)C(z) + B(z)D(z)}$$

```
fprintf('\nTotal system, positive notation')
```

```
Htot = tf( Kr*B, conv(A,C) + conv(B,D),h );
```

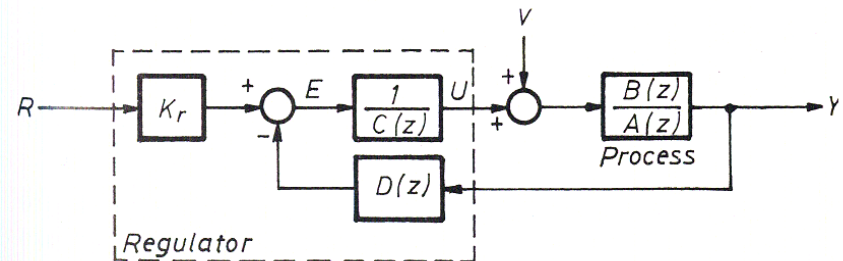
```
Htot.variable = 'z'
```

```
% Transfer function:
```

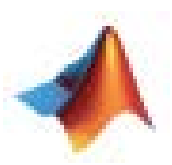
```
%      0.06639 z + 0.06361
```

```
% -----
```

```
% z^3 - 1.4 z^2 + 0.53 z + 2.776e-017 ≈ 0
```



# Matlab Script for Lab 3



Commands to plot system info.

```
% Print system info.  
figure('name','Step Response of Process','numbertitle','off')  
step(GP) % step response of process  
opt = stepDataOptions('StepAmplitude', 1.5, 'InputOffset', 1.5);  
figure('name','Step Response of Feedback Loop','numbertitle','off')  
step(Htot, opt) % step response of feedback loop  
figure('name','Margins of Feedback Loop','numbertitle','off')  
margin(Htot) % Gain and phase margins of feedback loop  
[Y,T]=step(Htot, opt); stepinfo(Y,T,'SettlingTimeThreshold', 0.05)
```

# Matlab Script for Lab 3



How do you check that the script is correct and that the poles are correctly placed?

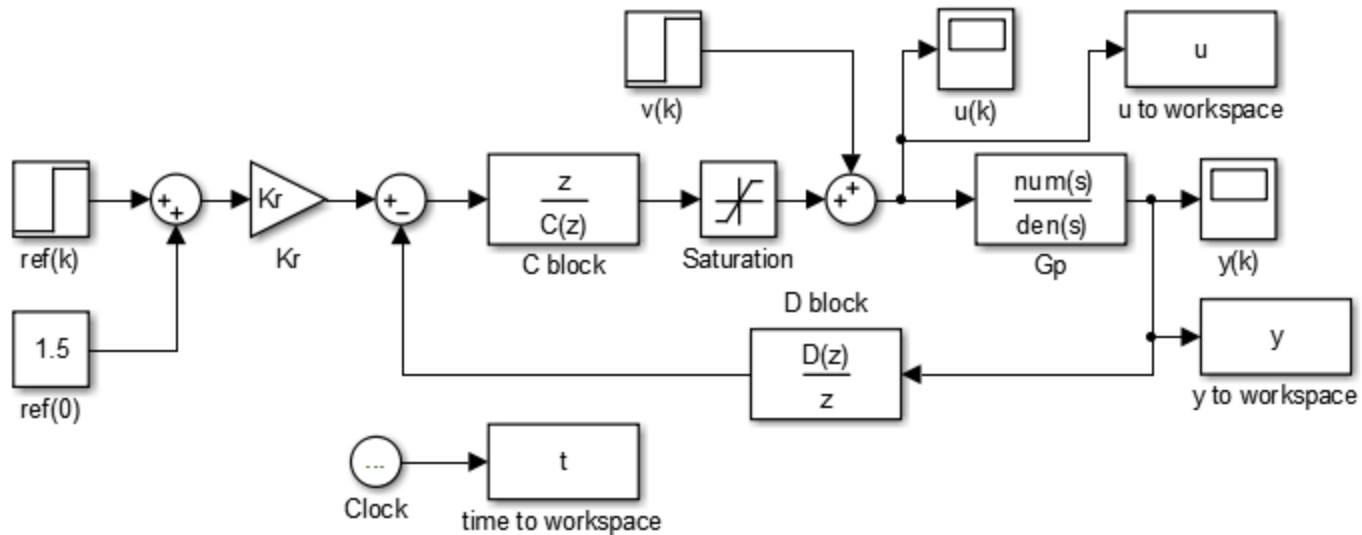
**[P,Z]=pzmap(Htot)**

```
% P =  
%      0  
%      0.7000+ 0.2000i  
%      0.7000- 0.2000i  
% Z = -0.9580
```

*Poles are as desired, the script is correct!*



# Simulink Model



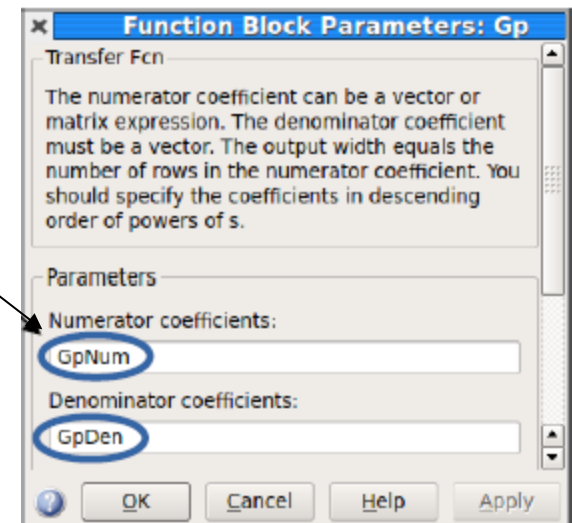
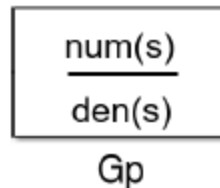


# Export transfer functions to Simulink

Simulink can simulate continuous-time and discrete-time systems together. For the process it is convenient to use the continuous-time model.

```
[GpNum, GpDen]=tfdata(GP, 'v')
```

**GpNum** and **GpDen**  
polynomials are  
imported in Simulink  
model.



# Export transfer functions to Simulink

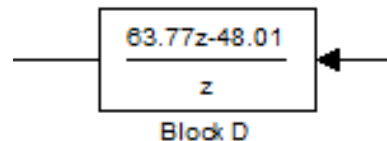
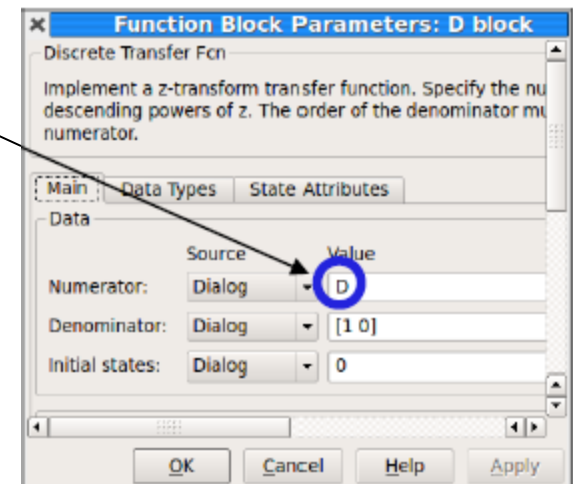
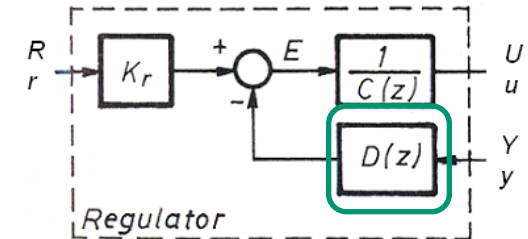


## % Controller blocks

```
D = [d_0  d_1 ]
D_block = tf( D,[1 0],h);
fprintf('\nBlock D positive notation')
D_block.variable = 'z'
```

```
% D = 63.7665 -48.0050
%
% Block D positive notation
% Transfer function:
% 63.77 z - 48.01
% -----
%          z
% Sampling time (seconds): 0.6
```

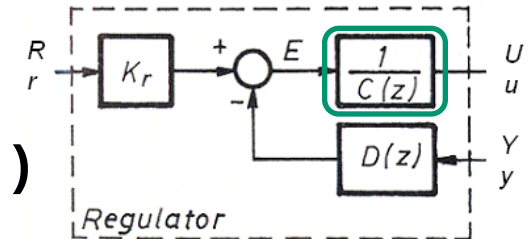
D polynomial  
is imported in  
Simulink  
model.



# Export transfer functions to Simulink

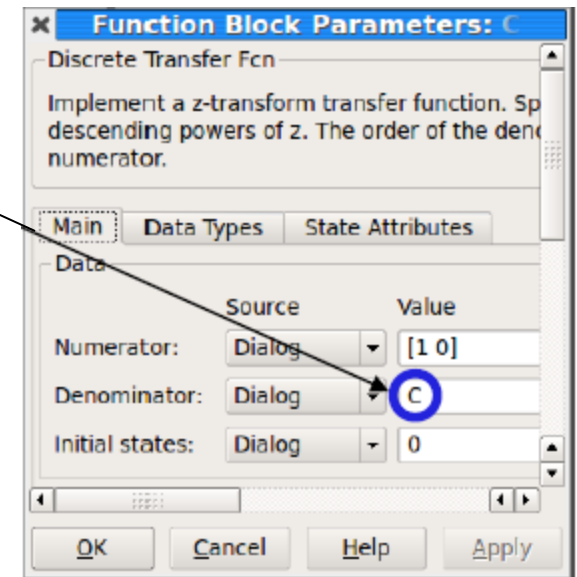
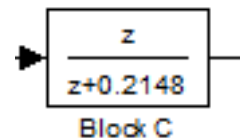


```
C = [1  c_1]
C_block = tf([1 0],C,h );
fprintf('\nBlock C positive notation')
C_block.variable = 'z'
```



```
% C = 1.0000 0.2148
%
% Block C positive notation
% Transfer function:
%      z
% -----
% z + 0.2148
% Sampling time (seconds): 0.6
```

C polynomial  
is imported in  
Simulink  
model.



# Export transfer functions to Simulink



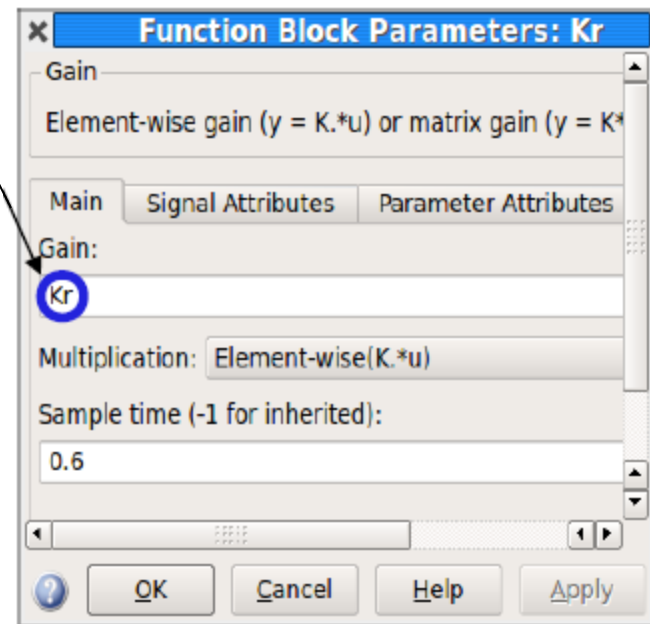
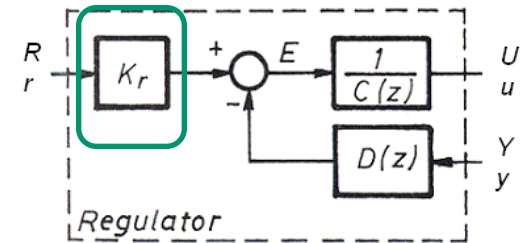
```
% Kr=P(1)/B(1)
```

$$K_r = \frac{P(1)}{B(1)}$$

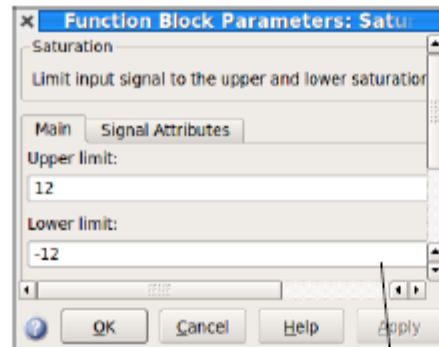
```
Kr = polyval(P,1) / polyval(B,1)
```

```
% Kr=16.1664
```

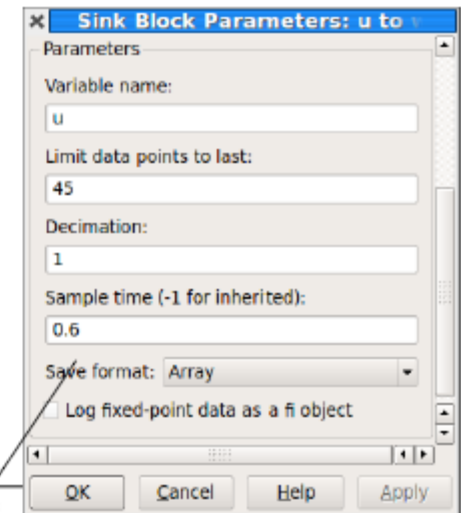
**Kr** value is imported in  
Simulink model.



# Simulink Model

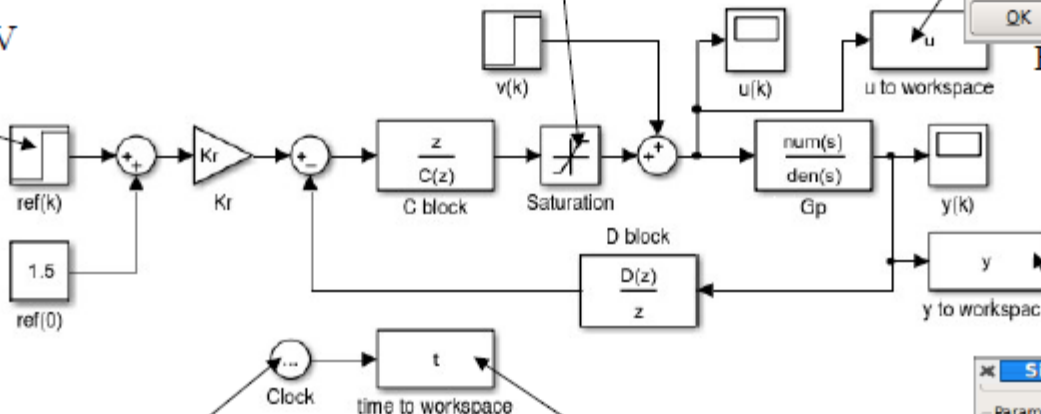
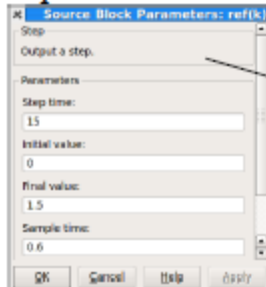


Max control input  $\pm 12$  V



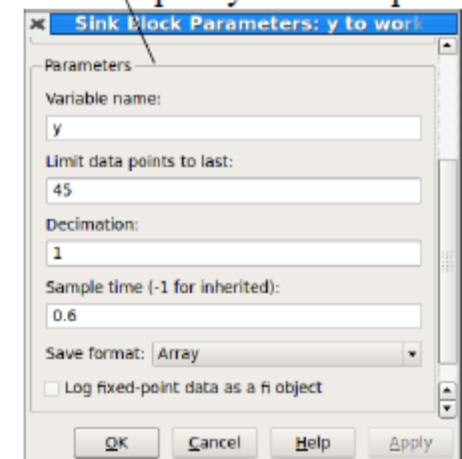
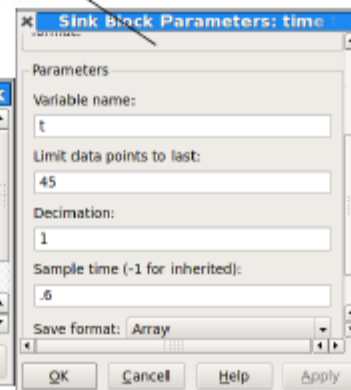
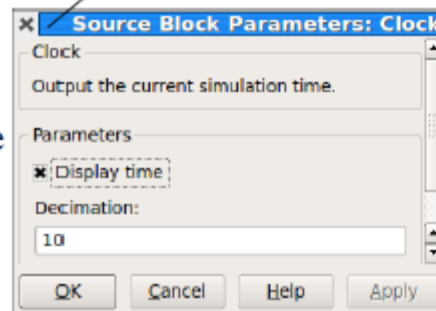
Export u to workspace

Step from 1.5 to 3 V



Export y to workspace

Time for workspace plots





# Plot Simulation Result in Matlab



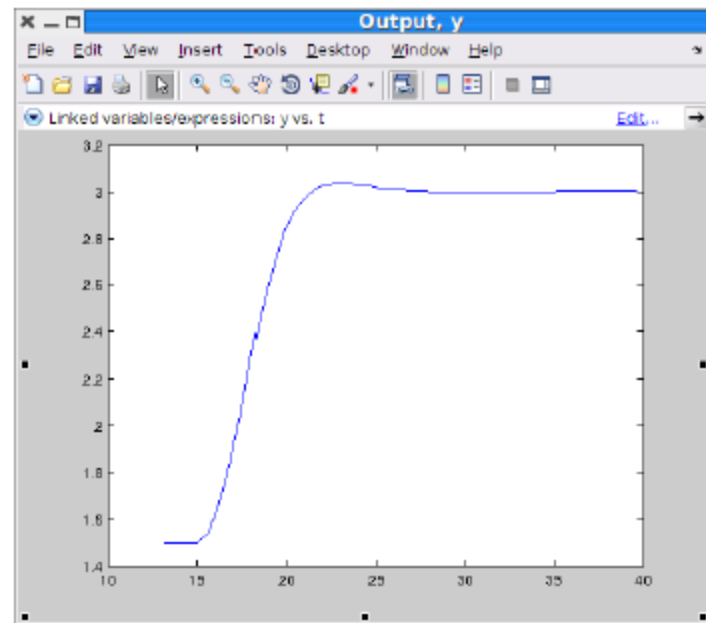
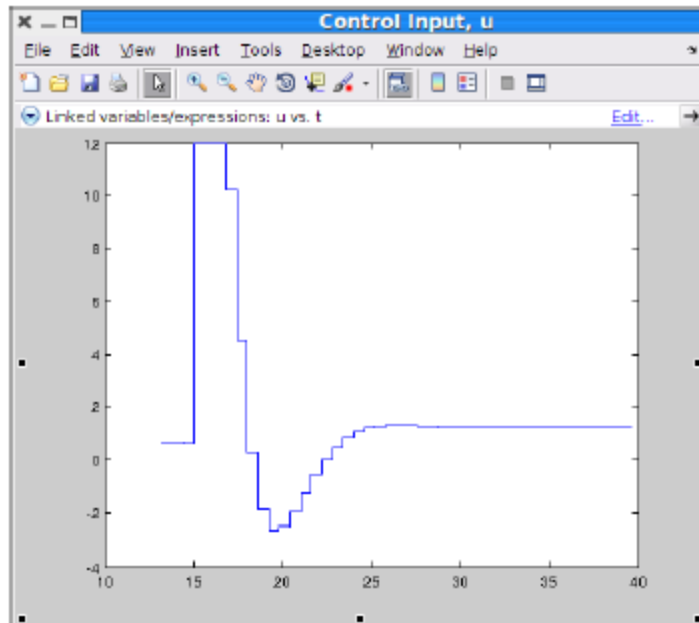
The following Matlab script creates one figure for control input and another for process output. The plots are updated when a new simulation is made.

```
% stepplot.m
% Plots output (y) and control input (u).
figure('name','Output, y','numbertitle','off')
plot(t,y)
linkdata on
figure('name','Control Input,u','numbertitle','off')
stairs(t,u)
linkdata on
```

# Simulation Plots



Simulation result when reference value is changed from 1.5V to 3V after 15s. Process output  $y(k)$  is continuous, while control input  $u(k)$  is discrete.





# Matlab script for C-code

## (Not part of the course.)



Matlab can write to a file! For example, the pole placement controller can be printed as a C function, which can be included when compiling a C program.

```
fid=fopen('ppc.c','w');
fprintf(fid,'float PolePlaceController(float r, float y)\n');
fprintf(fid,'{\n');
fprintf(fid,'    static float pre_u = 0;\n');
fprintf(fid,'    static float pre_y = 0;\n');
fprintf(fid,'    float e;\n');
fprintf(fid,'    float u;\n\n');
fprintf(fid,'    e = %6.3f*r - %6.3f*y - %6.3f*pre_y;\n',Kr,d_0,d_1);
fprintf(fid,'    u = e - %6.3f*pre_u;\n\n',c_1);
fprintf(fid,'    pre_u = u;\n');
fprintf(fid,'    pre_y = y;\n\n');
fprintf(fid,'    return u;\n}\n');
fclose(fid);
```

# C-function (not part of the Course)

A screenshot of the Notepad++ text editor. The title bar shows the file path as \*C:\ppc.c - Notepad++. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and Help. The toolbar contains various icons for file operations and editing. The editor has two tabs: main.c and ppc.c, with ppc.c being the active tab. The code in ppc.c is a C function named PolePlaceController that takes two float arguments, r and y, and returns a float. The function uses static variables pre\_u and pre\_y to store previous values of u and y respectively. The calculation for u is based on a linear combination of r, y, and pre\_y, with a feedback term from pre\_u. The function updates pre\_u and pre\_y before returning u. The console window at the bottom is empty. The status bar at the bottom shows the cursor is at line 15, column 2, and the encoding is ANSI.

```
1 float PolePlaceController(float r, float y)
2 {
3     static float pre_u = 0;
4     static float pre_y = 0;
5     float e;
6     float u;
7
8     e = 16.166*r - 63.766*y - -48.005*pre_y;
9     u = e - 0.215*pre_u;
10
11     pre_u = u;
12     pre_y = y;
13
14     return u;
15 }
```