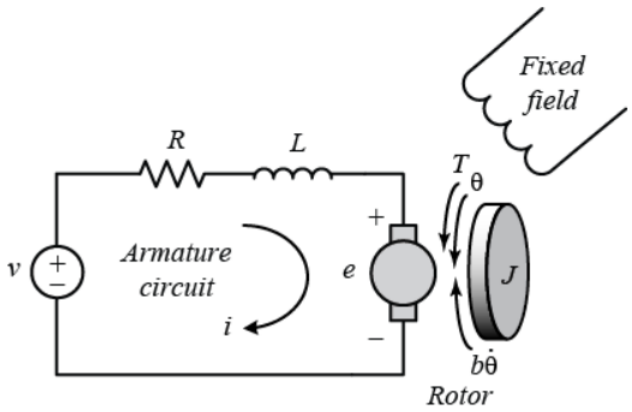
DC MOTOR SPEED MODELLING AND CONTROL



Dynamics Equations

$$T = K * i$$

$$e = K * \dot{\theta}$$

$$K * i = J * \ddot{\theta} + b * \dot{\theta}$$

$$K * \dot{\theta} + R * i + L * \frac{di}{dt} = V$$

Transfer Function

$$KI(s) = Js\dot{\theta}(s) + b\dot{\theta}(s)$$

$$RI(s) + LsI(s) = V(s) - K\dot{\theta}(s)$$

$\frac{\dot{\theta}(s)}{V(s)}$ give dc motor speed transfer function equation by eliminating $I(s)$.

```
%transfer function
syms s Is theta_s J b K R L
Is = ((s*J+b)/K) * theta_s;
Vs = (R+L*s)*Is + K*s*theta_s;
mspeed_tf = theta_s/Vs;
pretty(collect(mspeed_tf,s))
```

$$\frac{\dot{\theta}(s)}{V(s)} = \frac{K}{((J*L*s^2)+(K^2+L*b+J*R)*s+R*b)}$$

```
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;
%transfer function
syms s Is theta_s
Is = ((s*J+b)/K) * theta_s;
Vs = (R+L*s)*Is + K*s*theta_s;
mspeed_tf = theta_s/Vs;
pretty(collect(mspeed_tf,s))
```

```
              100
--------------------
    2
50 s  + 601 s + 1000
```

State Space

$$\frac{d}{dt}\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} = \begin{bmatrix}-b/J & K/J\\ -K/L & -R/L\end{bmatrix} * \begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{L}\end{bmatrix} * V$$

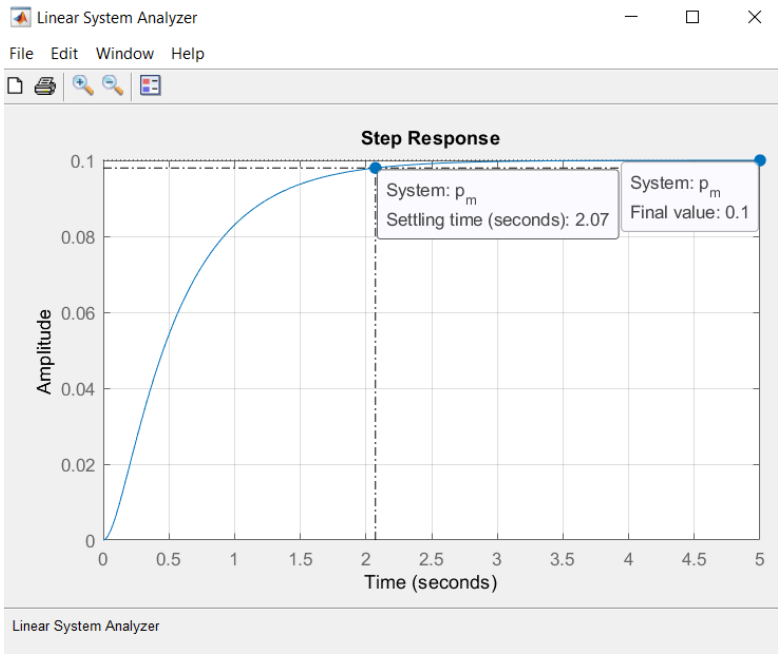
$$y = [1 \quad 0] * \begin{bmatrix}\dot{\theta}\\ i\end{bmatrix}$$

```
%state space
A = [-b/J   K/J
    -K/L   -R/L];
B = [0
    1/L];
C = [1   0];
D = 0;
motor_ss = ss(A,B,C,D)
```

```
motor_ss =

  A =
         x1     x2
   x1    -10      1
   x2  -0.02     -2

  B =
       u1
   x1   0
   x2   2

  C =
       x1  x2
   y1   1   0

  D =
       u1
   y1   0

Continuous-time state-space model.
```

Analysis

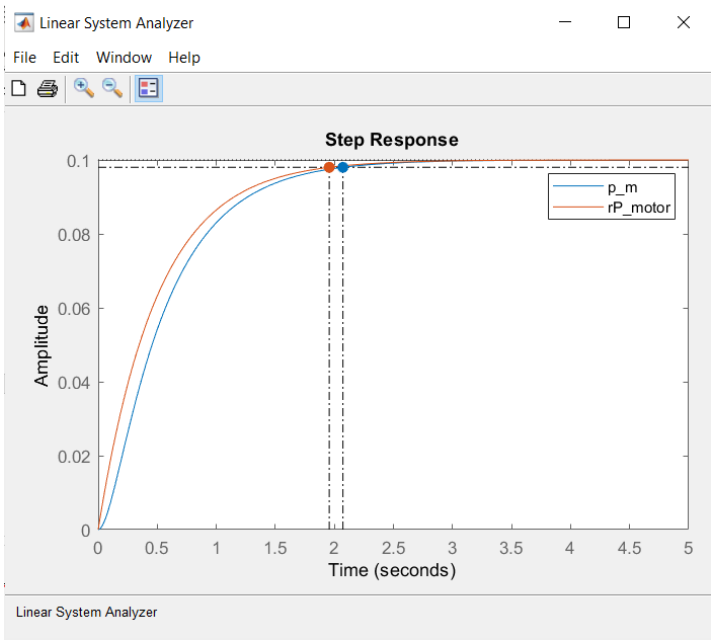
```
%analysis
s =tf('s');
p_m = K/((J*L*s^2)+(K^2 +L*b+J*R)*s + R*b);
%%% t=0:0.1:5;
%%% step(p_m,t)
linearSystemAnalyzer('step', p_m, 0:0.01:5);
```



To evaluate the performance of the original open-loop system, you can use the Linear System Analyzer command in MATLAB. By specifying the system and the type of response plot you want to generate, you can view the performance characteristics of the system. In this case, the system under evaluation is p_m, and a step response plot is generated by passing the string 'step' to the Linear System Analyzer function. The range of numbers 0:0.1:5 specifies the time values for which data points are included in the plot, in steps of 0.1 seconds. By examining the resulting plot, you can view important performance characteristics such as the settling time and steady-state behavior of the system.

To access the pole-zero map, we can right-click on the plot area in the Linear System Analyzer and select "Plot Types > Pole/Zero" from the menu.By examining the pole-zero map, we can see that the open-loop transfer function has two real poles at s = -2 and s = -10, which means that there is no oscillation in the step response and the slower pole dominates the dynamics. In this case, the pole at s = -2 primarily determines the speed of response of the system, making the system behave similarly to a first-order system. To see how well a first-order model approximates the original motor model, we can build a first-order transfer function with a pole at s = -2 and steady-state value matching the original transfer function. We can do this by using the following command in MATLAB:

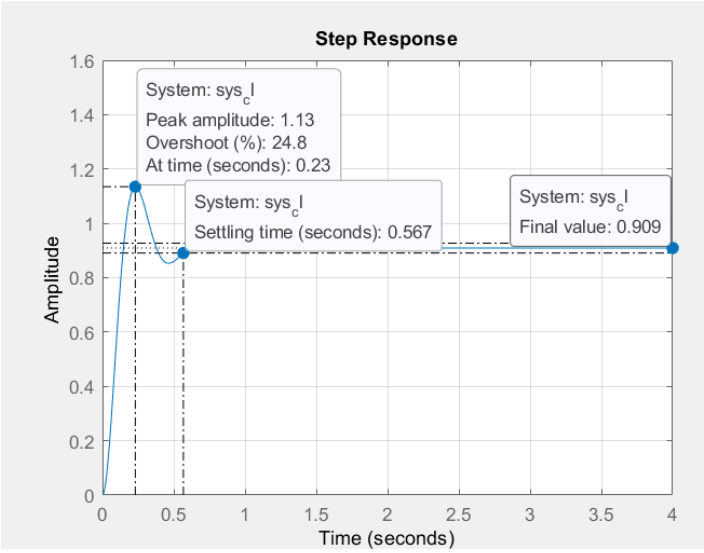
```
s =tf('s');
p_m = K/((J*L*s^2)+(K^2 +L*b+J*R)*s + R*b);
rP_motor = 0.1/(0.5*s+1);
linearSystemAnalyzer('step', p_m,rP_motor, 0:0.01:5);
```



PID Controller Design

Let's start by attempting to use a proportional controller with a gain of 100, which can be represented by the transfer function C(s) = 100. We can find the closed-loop transfer function by using the feedback command in MATLAB. After that, we can analyze the closed-loop step response by generating a plot, which should look similar to the one shown in the figure. To view specific characteristics of the system, we can right-click on the plot and select "Characteristics" from the menu. The annotations in the figure indicate Settling Time, Peak Response, and Steady State.

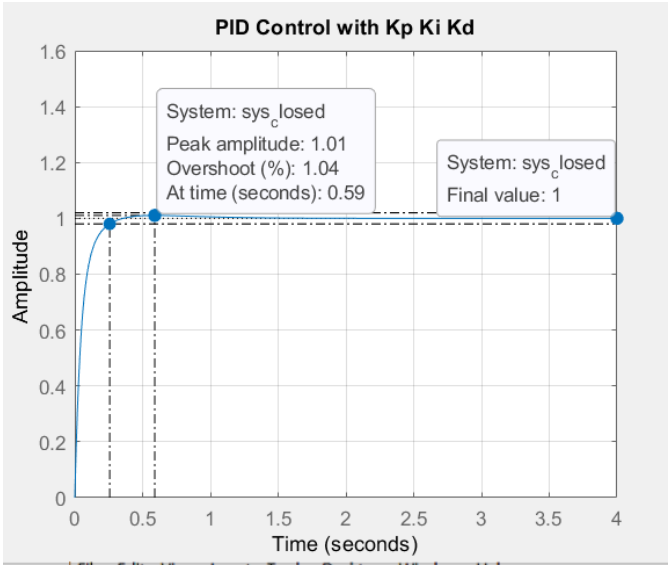
```
%propotional
Kp = 100;
C = pid(Kp);
sys_cl = feedback(C*p_m,1);
t = 0:0.01:4;
step(sys_cl,t)
%%% linearSystemAnalyzer('step',sys_cl,0:0.01:2)
grid
title('Step Response with Proportional Control')
```



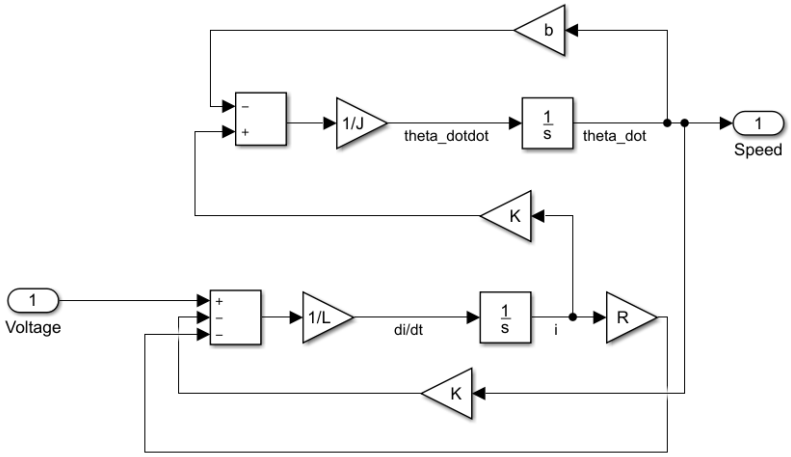
From the plot, we can observe that both the steady-state error and the overshoot are too large. To reduce the steady-state error, we can increase the proportional gain Kp. However, it's important to note that increasing Kp often results in increased overshoot, which means that a simple proportional controller may not be sufficient to meet all of the design requirements.

To reduce the steady-state error to a step reference, it is often necessary to add an integral term to the controller. Similarly, adding a derivative term can help to reduce the overshoot. In this case, we can try using a PID controller with values for k_i and k_d. To do so, we need to modify the MATLAB code to include the new terms. By adding the integral term, the steady-state error is eliminated much more quickly than before. However, the large value for k_i has greatly increased the overshoot. To compensate for this, we can increase the value of k_d to reduce the resulting overshoot. With the values for k_p = 100, k_i = 200, and k_d = 10, we should be able to achieve our desired performance requirements.

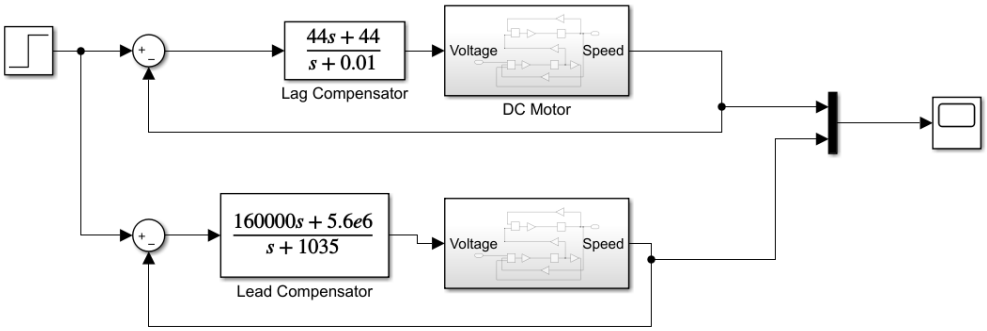
```
%pid
kp=100; ki=200; kd=10;
C = pid(kp,ki,kd);
sys_closed = feedback(C*p_m,1);
step(sys_closed,[0:0.01:4])
title('PID Control with Kp Ki Kd')
```



Simulink Modelling



Simulink Controller Design



When running the simulation and viewing the scope output, you will notice that both the lead-compensated and lag-compensated systems exhibit a steady-state error that tends towards zero. Upon comparing the two graphs, you will observe that the lead-compensated system (indicated by the blue response) has a significantly smaller settling time compared to the lag-compensated system (indicated by the yellow response). Additionally, the lead-compensated system has a slightly larger, yet comparable, overshoot when compared to the lag-compensated system.