ME – 190 MECHATRONICS SYSTEM DESIGN
Lab 9*: Basic DC Motor Feedback Control

Objectives:

- Simulate and implement basic control concepts
- Carry out a closed-loop DC motor speed controller
- Carry out a closed-loop DC motor position controller

Part 1: Ideal 2nd order DC Motor Model

Background Information:

The relationship between the applied voltage and the angular speed of an ideal DC motor can be modeled as a 2nd order transfer function:

$$\frac{\dot{\theta}}{v} = \frac{K_t}{LJs^2 + (RJ + BL)s + RB + K_tK_b}$$

This lab will be using this transfer function to create a *Simulation* of the DC motor. The ultimate objective of this lab is to design feedback controllers that control the motor speed (simulation) and motor position (experiment) to desired (reference) commands.

Simulink Model of the Motor Transfer Function

Create the following Simulink diagram. Notice the motor subsystem contains the 2nd order transfer and as well as an integrator to obtain position output. Therefore, the transfer function from voltage to position is third order.

The contents of the motor subsystem are shown in Figure 2.

Note: If you have trouble entering the transfer function, remember to place only the coefficients between brackets '[ ]'. Select all components and right click to create a subsystem. You can then drag this subsystem block to a new system model window.

*Courtesy of Dr. Joshua Hurst at RPI/MinSeg

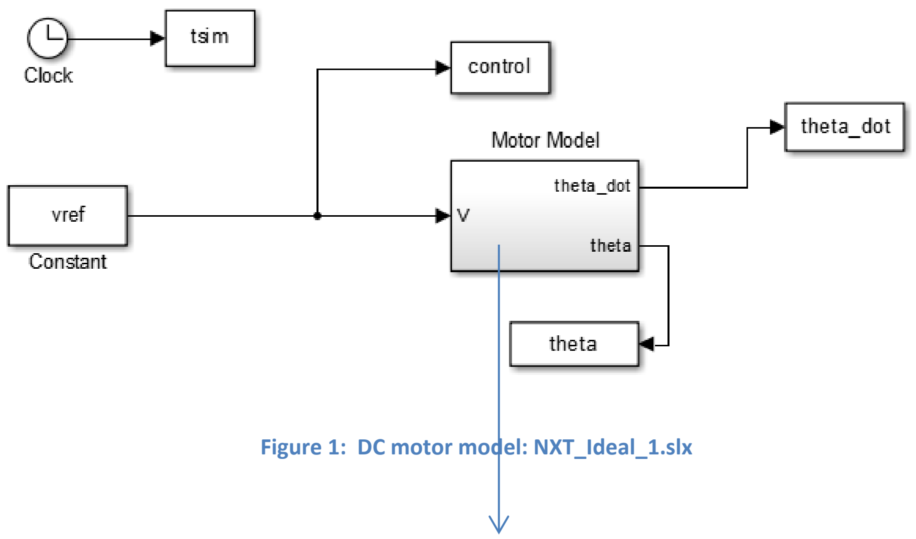


Figure 1: DC motor model: NXT_Ideal_1.slx

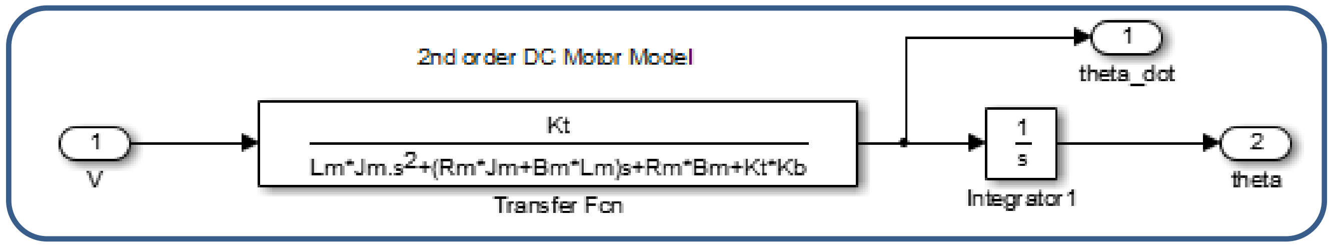
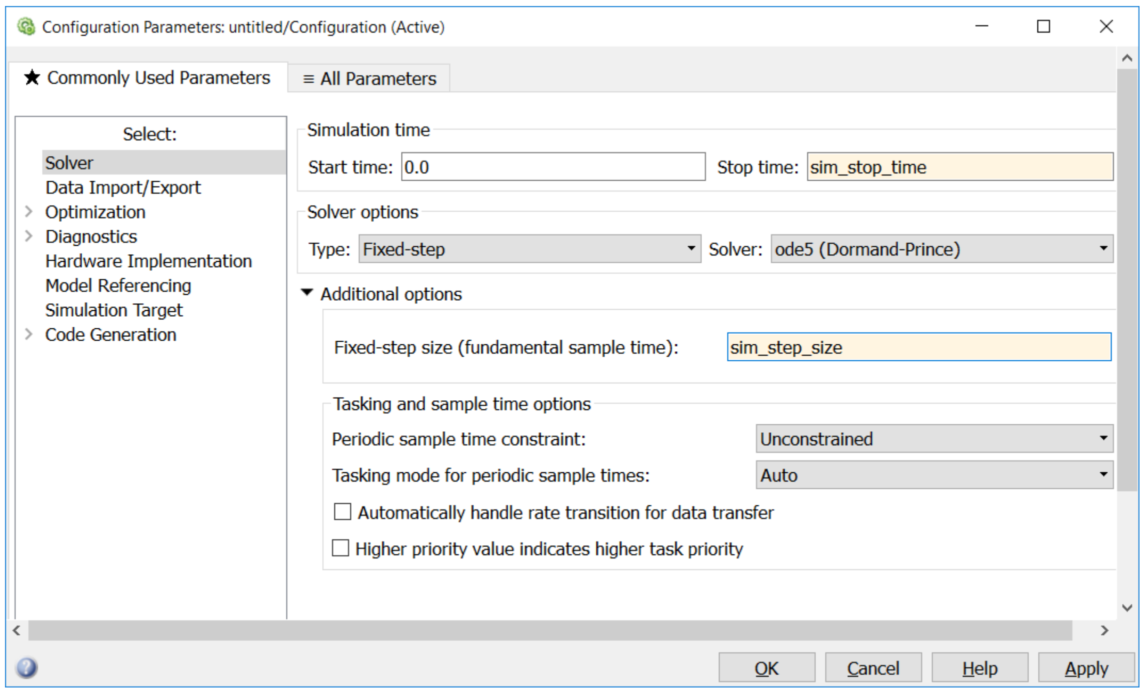


Figure 2: DC motor model subsystem

This Simulink diagram will be used to simulate the system response. The "To Workspace" blocks are used to log the results of this simulation. The system needs to be sampled fast enough to capture features of interest. Since the step-response is on the order of milliseconds set the sample time of these blocks to 0.0001 seconds. This way we "solve" the system with enough resolution to view any effects that may be simulated. To solve the system use the following solver settings:



Simulation time

Start time: 0.0 Stop time: sim_stop_time

Solver options

Type: Fixed-step Solver: ode5 (Dormand-Prince)

Additional options

Fixed-step size (fundamental sample time): sim_step_size

Tasking and sample time options

Periodic sample time constraint: Unconstrained

Tasking mode for periodic sample times: Auto

☐ Automatically handle rate transition for data transfer

☐ Higher priority value indicates higher task priority

The solver is ode5 using a fixed-step method.

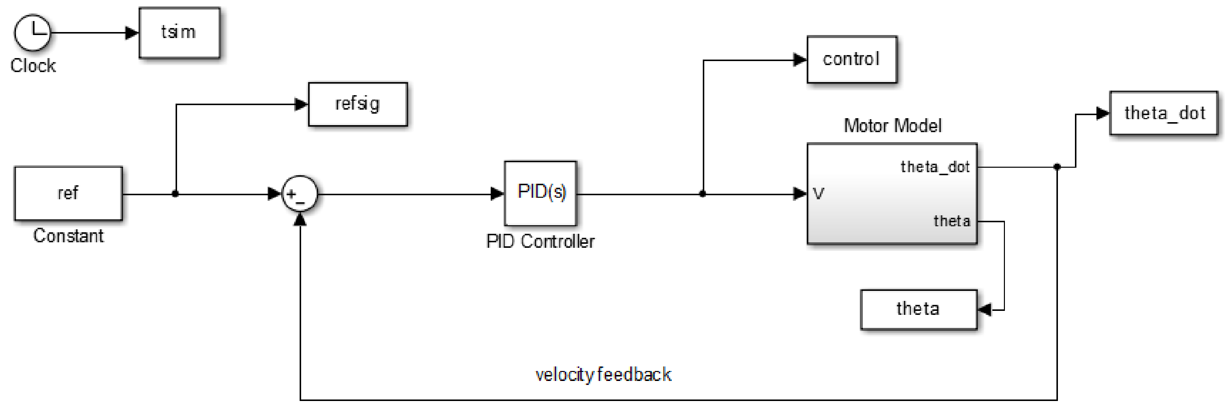
Define the motor parameters run the simulink diagram, and plot the results with the following m-file:

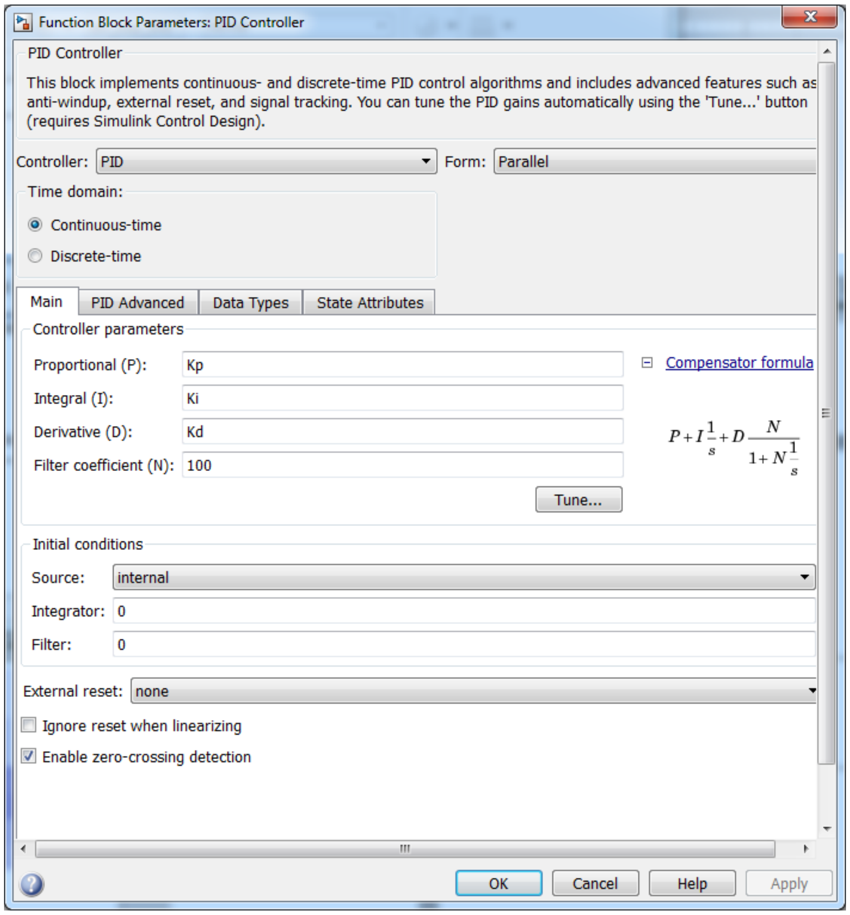
```
% NXT motor parameters:
Rm = 5.262773292;      % ohms
Kb = 0.4952900056;     % Vs/rad
Kt = 0.3233728703;     % Nm/A
Bm = 0.0006001689451; % Nms/rad (viscous friction)
Lm = 0.0047;           % H
Jm = 0.001321184025;  % kgm^2 (combined J)

% simulation parameters
sim_stop_time=.3; % simulation length - seconds
sim_step_size = .0001; % how often log simulation data

% plot the setp response:
vref=9;  % reference step in voltage

% run simulink diagram to solve system response
sim('NXT_Ideal_1.slx')

% plot results
figure(1), hold on
h_step=plot(tsim, theta_dot/(2*pi)*60, 'b');
xlabel('time seconds')
ylabel('speed RPM')
title('Ideal 2nd order step response')
```

Part 2: Ideal PI Speed Control of DC Motor Model (Simulation)

Background Information:

Using the transfer function for the DC motor model, a PI controller can be designed. Implement the ideal PI controller with the following Simulink diagram [the 'PID Controller' block is found under the "Continuous" Simulink library]. Save this new diagram as "NXT_Ideal_PID_2.slx".




Function Block Parameters: PID Controller

PID Controller

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID   Form: Parallel

Time domain:
- (●) Continuous-time
- ( ) Discrete-time

Main | PID Advanced | Data Types | State Attributes

Controller parameters

Proportional (P):	Kp
Integral (I):	Ki
Derivative (D):	Kd
Filter coefficient (N):	100

Compensator formula

$$P+I\frac{1}{s}+D\frac{N}{1+N\frac{1}{s}}$$

Initial conditions

Source:	internal
Integrator:	0
Filter:	0

External reset: none

- [ ] Ignore reset when linearizing
- [x] Enable zero-crossing detection

Use a reference speed of 12 rad/sec, and enter the PID parameters as shown below. Now, update the m-file to run the simulation and create the response plot.

```
% ref speed
ref=12; % rad/sec

% Run simulation with Prop settings;
Kp=.9 ; Ki=0; Kd= 0;        % for units of rad/sec to Volts
sim('NXT_Ideal_PID_2.slx')

% plot results
figure(1), hold on
hKp=plot(tsim,theta_dot,'g');
hv_demand_Kp=plot(tsim, control, 'm');
hKp_ref=plot([0 tsim(end)], [ref ref], 'r');
plot(tsim, refsig, 'r');
xlabel('time (sec)')
ylabel('response')
title('Speed Control for Ideal 2nd order transfer function Kp=0.9 Ki=0')

% create legend
legend([h_step hKp_ref hKp hv_demand_Kp],...
    {'Open Loop Step Response (rad/sec)',...
    'reference signal (rad/sec)',...
    'proportional control response (rad/sec)',...
    'proportional control demand (volts)'})
```

Questions:

- What do you notice about the steady state error with just a proportional controller?
- What is the maximum voltage the controller uses to obtain the response?

Now, compare the performance of the system with PI control. Add the following code into the m-file, and update the legend appropriately:

```
% Run simulation with PI settings;
Kp=.9 ; Ki=38; Kd= 0;        % for units of rad/sec to Volts
sim('NXT_Ideal_PID_2.slx')
hKi=plot(tsim,theta_dot,'g--');
hv_demand_Ki=plot(tsim, control, 'm--');
```

Questions:

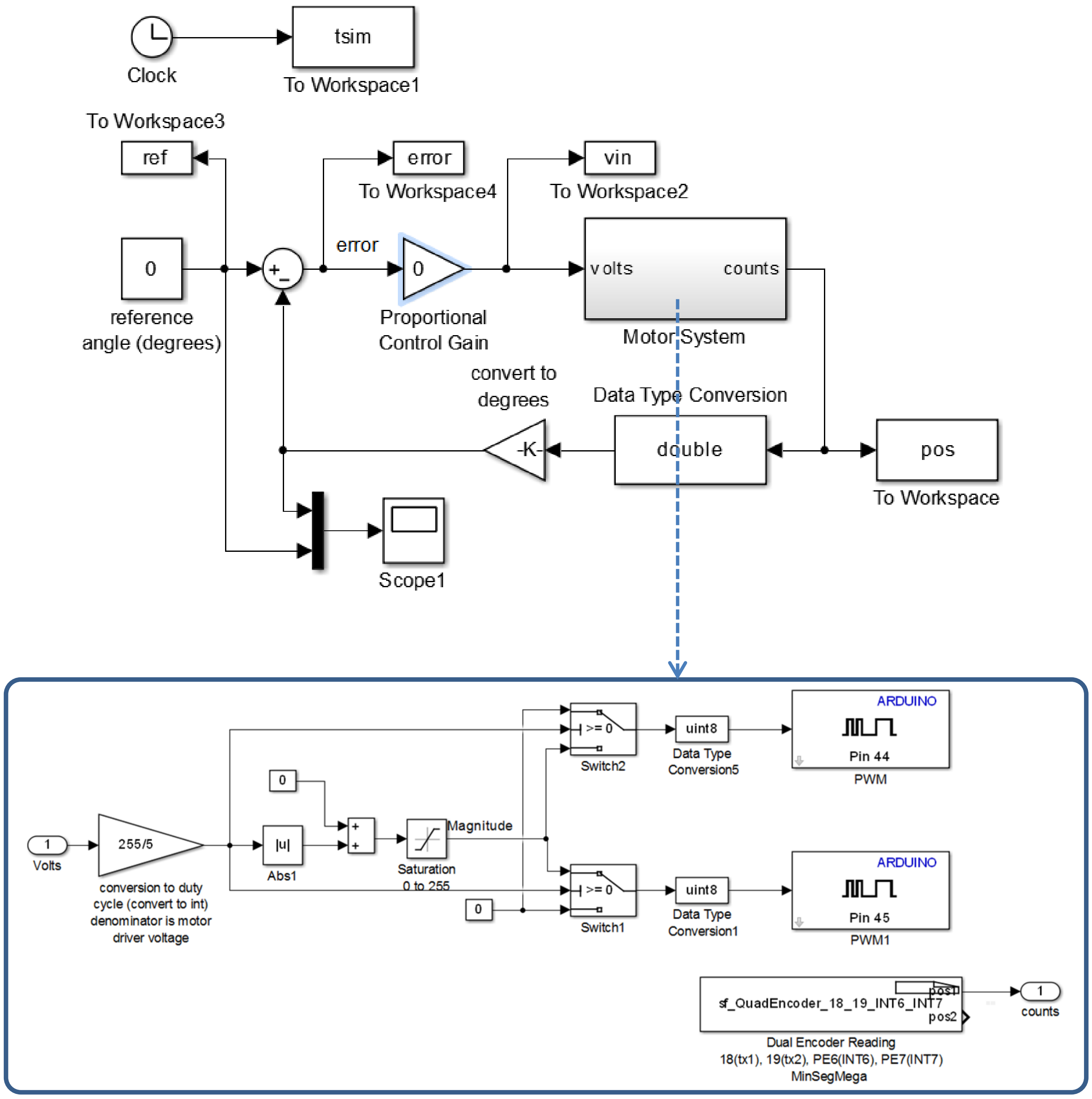
- How does the PI controller perform?

Part 3: Closed Loop Position Control (MinSeg)

Measuring the output of interest and using this information in the control design is called closed loop feedback control. The basic strategy is to measure the output of interest, compare this to the desired output to generate an error signal and then try to reduce this error.

Proportional Control:

Build the following Simulink diagram:



* What value should be used to convert from the output in counts to degrees?

- Run the Simulink
    - Use a time step size of .03 seconds in external mode
    - Since the initial gain is zero the system should remain at rest
    - Change the proportional gain to 0.05 then change the reference to 360

    * How does the position of the motor respond to the step command?

    - Change the reference back to zero
    - Change the proportional gain to 0.1 then change the reference to 360

    * How does the response of the system change?

    - Change the reference back to zero

- Modify simulation settings
    - Change the time step size to 0.1 seconds in external mode
    - Change the proportional gain to 0.1 then change the reference to 360

    * What do you notice?

Check Point:

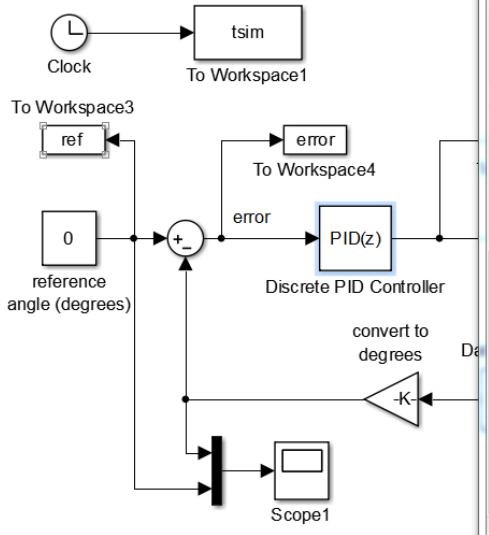
- Create a single plot that shows error vs. time for the following cases (you may want to use the 'save' command to save your data between each experiment for plotting later)
    - Step size of 0.03, proportional gain of 0.05
    - Step size of 0.03, proportional gain of 0.1
    - Step size of 0.1, proportional gain of 0.1

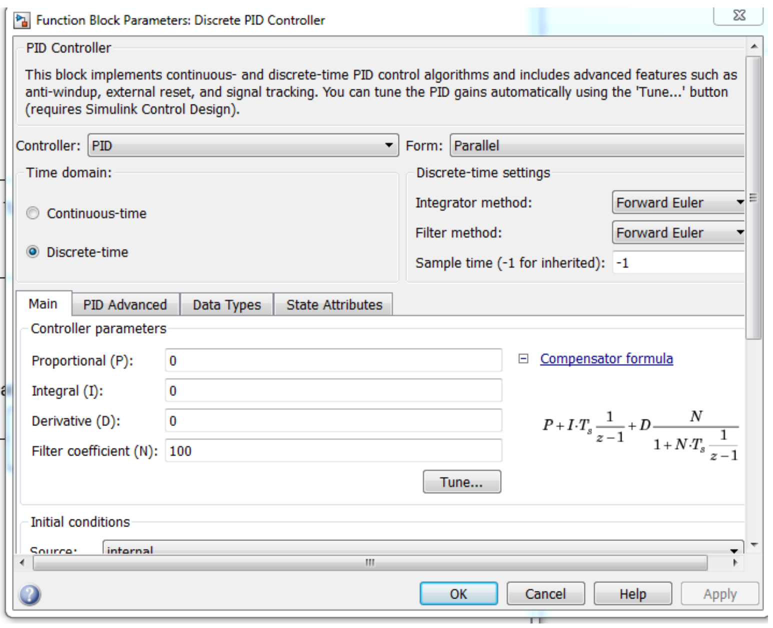
    Comment on the effect of the step size (delay) and the gain.

Proportional Plus Integral Control:

Proportional control alone does not eliminate the steady state error. If the error signal is integrated this value will keep increasing as long as there is a steady state error and a control effort can be applied that is proportional to the integral of the error

Build the following Simulink diagram:





Function Block Parameters: Discrete PID Controller

PID Controller

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID   Form: Parallel

Time domain:
- Continuous-time
- [✓ Discrete-time]

Discrete-time settings
- Integrator method: Forward Euler
- Filter method: Forward Euler
- Sample time (-1 for inherited): -1

Main | PID Advanced | Data Types | State Attributes

Controller parameters

Proportional (P):	0
Integral (I):	0
Derivative (D):	0
Filter coefficient (N):	100

Compensator formula

$$P + I \cdot T_s \frac{1}{z-1} + D \frac{N}{1 + N \cdot T_s \frac{1}{z-1}}$$

Initial conditions

Source: internal

- o Run this on the system.
  - o Use a time step size of 0.03
  - o Change the proportional gain to 0.05, then the reference to 360 – the response should be the same as before with a steady state error
  - o Change the integral gain to 0.1 – what do you observe?

Checkpoint

- Add to the previous plot to include the response from the PI controller.

Question:

How do you compare a P and a PI controller? Which one would you use for a motor position control application?