RENSSELAER MECHATRONICS

Realistic DC Motor Speed Control Simulation

Objectives:

- Create a realistic truth model for the DC motor control system including most of the hardware implementation elements
  - Time Delay (limited microcontroller sampling rate)
  - Saturation (limited supply voltage)
  - Discretization of measurement readings (Encoder resolution)
  - Discretization of actuator output (PWM resolution)
  - Unit Conversions
  - Overflow
- This simulation will not address
  - Backlash in the gears

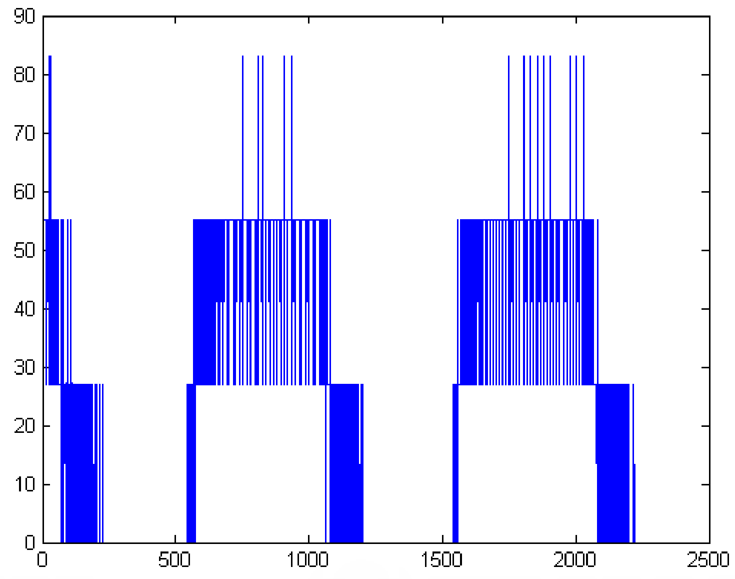
Part 1: Ideal Step Response of 2nd order DC Motor Model

Background Information:

An ideal DC motor can be modeled as a 2nd order transfer function. This will be the basis for control development:

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

This lab will be using this transfer function to create a *Simulation* of the DC motor and its hardware implementation. The actual experimental step response for the hardware sampled at .003 seconds is shown below:



The objective of this lab is to investigate the elements of the system that create this hardware obtained experimental measured response, and explore the implication of this on the resulting control system.

Simulink Model

Create the following Simulink diagram. Notice the motor subsystem contains the 2nd order transfer and it also computes the position as this will be useful for position control in the future. The contents of the subsystem are shown in Figure 2.

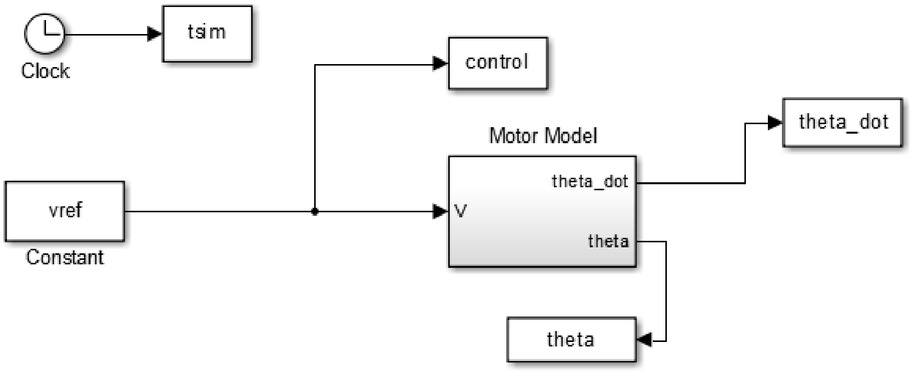


Figure 1: 2nd order DC motor model: NXT_Ideal_1.slx

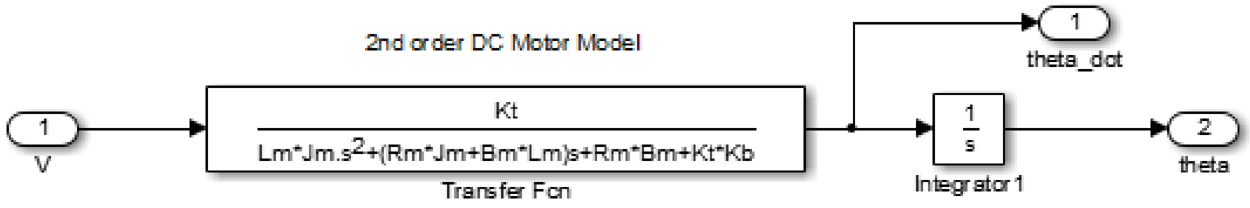
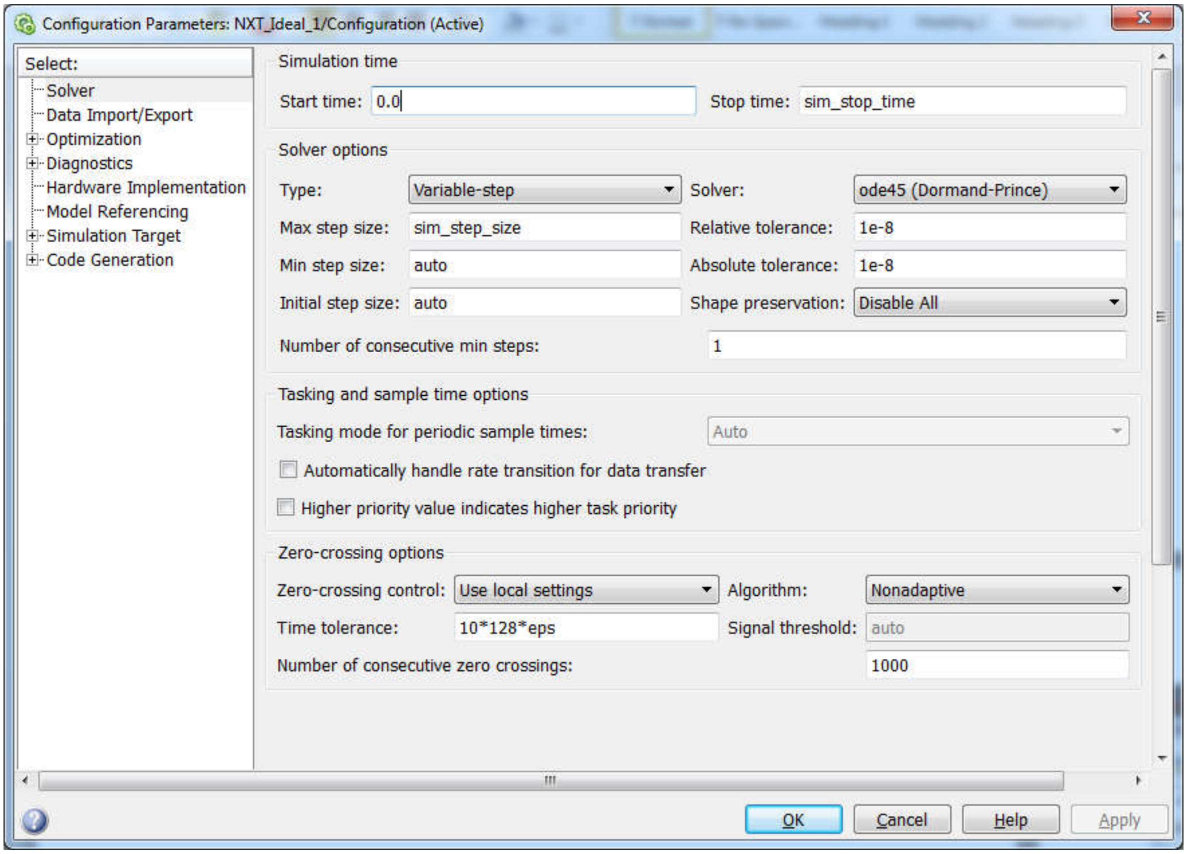


Figure 2: 2nd order DC motor model subsystem

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.

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 .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:



Configuration Parameters: NXT_Ideal_1/Configuration (Active)

Simulation time

Start time: 0.0
Stop time: sim_stop_time

Solver options

Type: Variable-step
Solver: ode45 (Dormand-Prince)
Max step size: sim_step_size
Relative tolerance: 1e-8
Min step size: auto
Absolute tolerance: 1e-8
Initial step size: auto
Shape preservation: Disable All
Number of consecutive min steps: 1

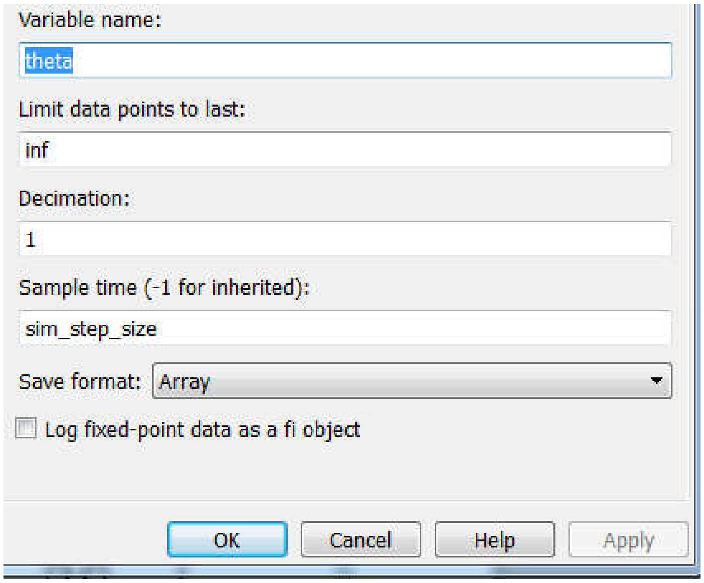
Tasking and sample time options

Tasking mode for periodic sample times: Auto
□ Automatically handle rate transition for data transfer
□ Higher priority value indicates higher task priority

Zero-crossing options

Zero-crossing control: Use local settings
Algorithm: Nonadaptive
Time tolerance: 10*128*eps
Signal threshold: auto
Number of consecutive zero crossings: 1000

Notice the variable sim_step_size. This is how often data is logged. This variable is also used in the "To Workspace" blocks. Make sure you apply this to each output block.



Variable name:

theta

Limit data points to last:

inf

Decimation:

1

Sample time (-1 for inherited):

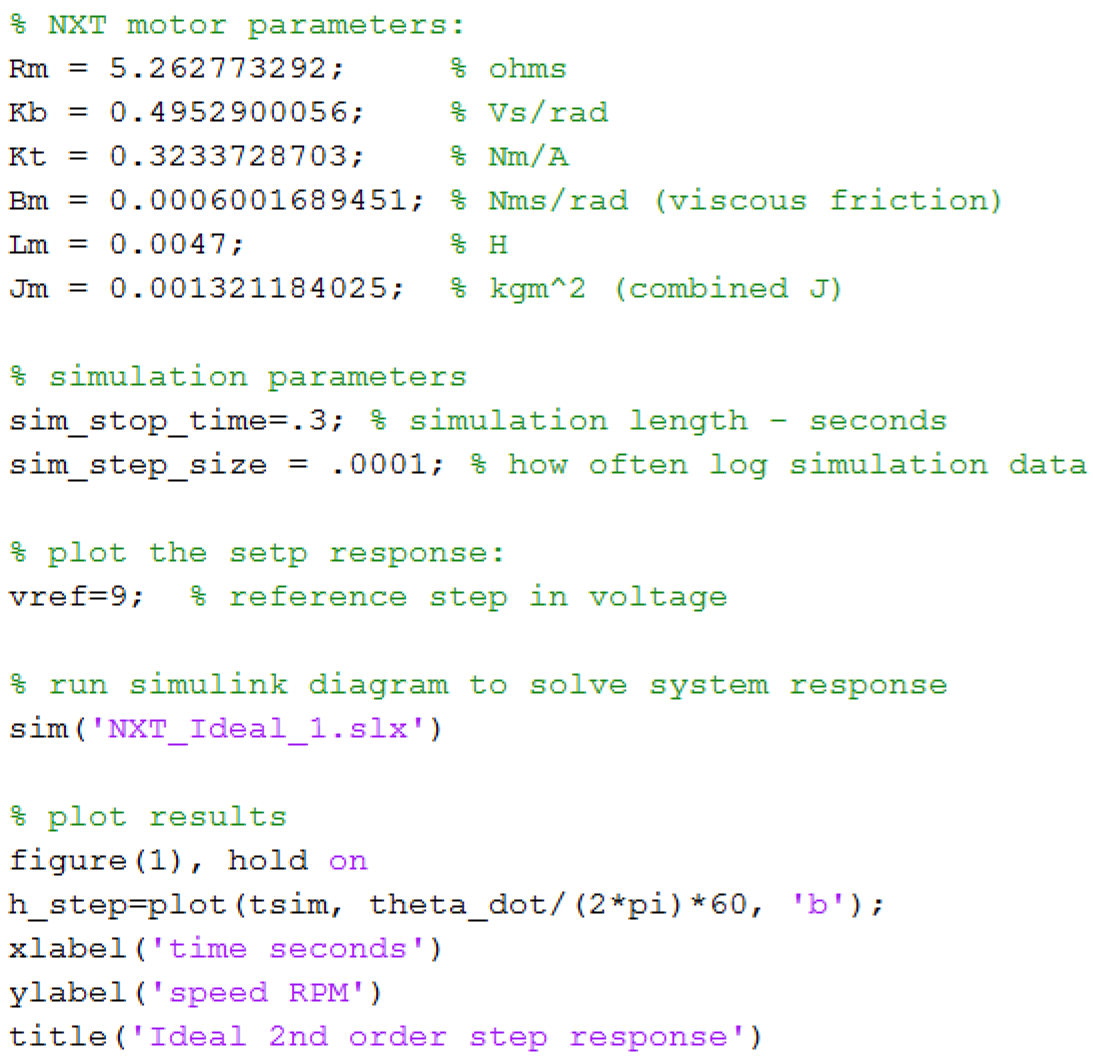
sim_step_size

Save format: Array

□ Log fixed-point data as a fi object

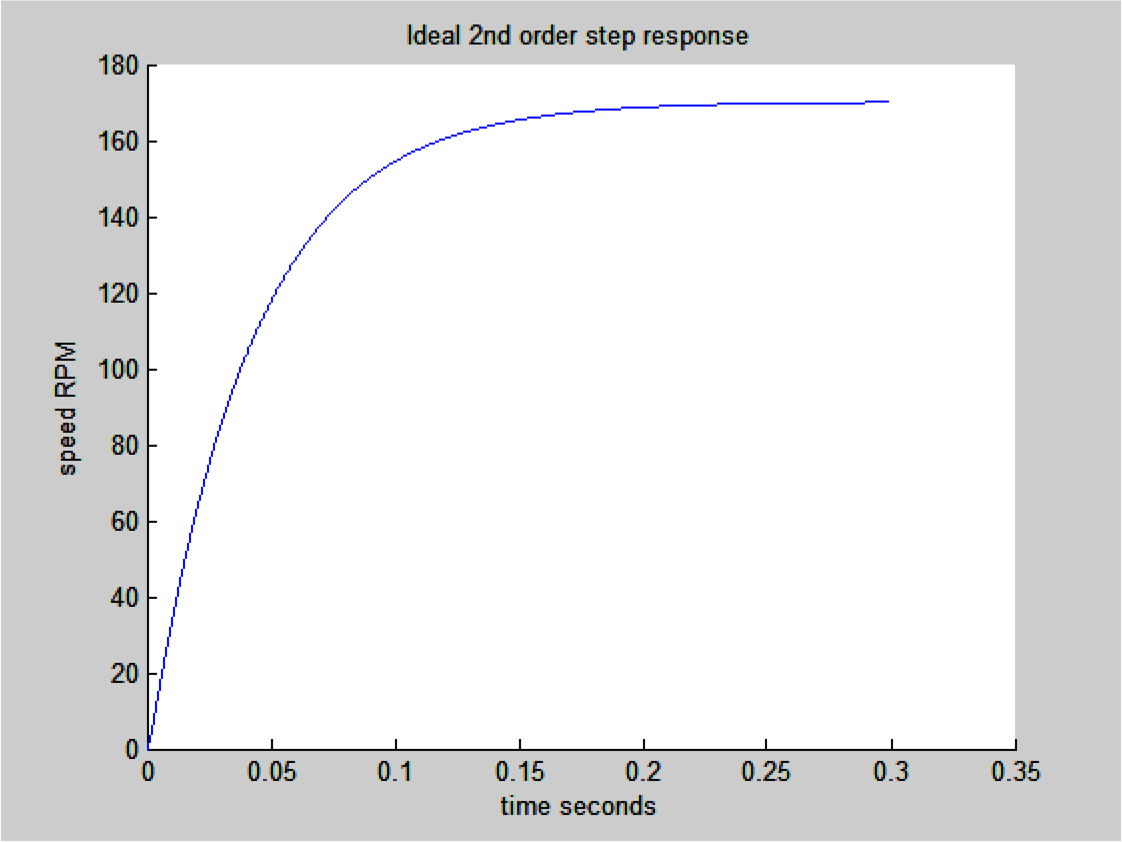
The solver is ode45 using a variable-step method.

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

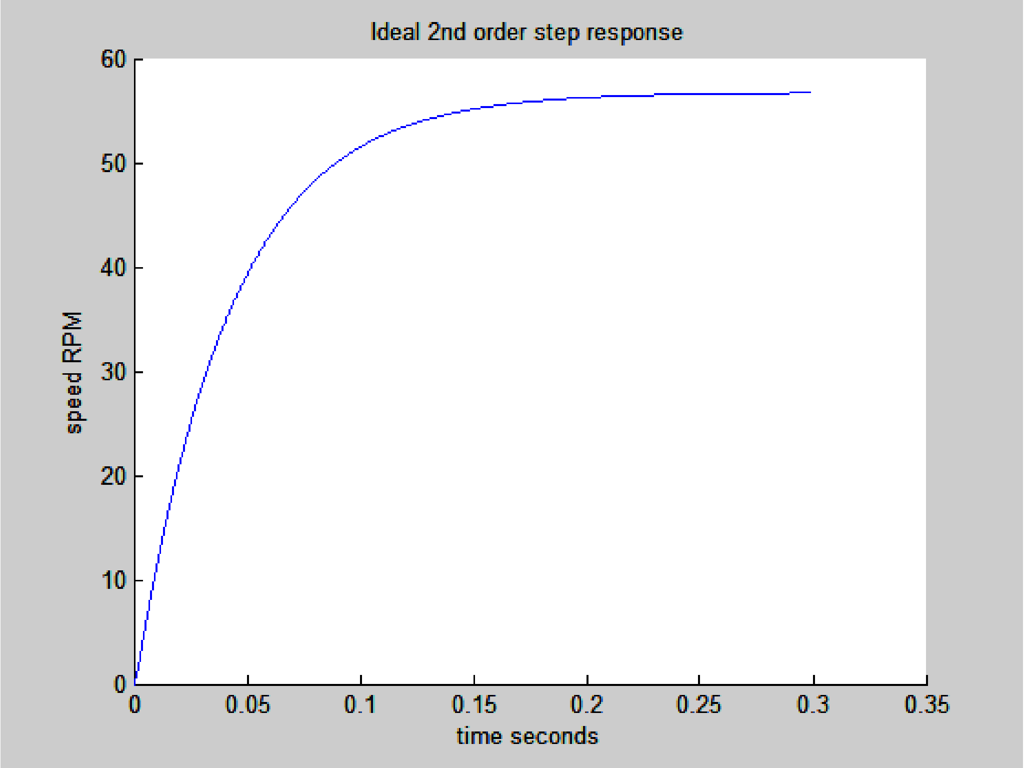


```
% NXT motor parameters:
Rm = 5.262773292;      % ohms
Kb = 0.4952900056;     % Vs/rad
Kt = 0.3233728703;     % Nm/A
Bm = 0.0006001689451; % Nms/rad (viscous friction)
Lm = 0.0047;           % H
Jm = 0.001321184025;  % kgm^2 (combined J)

% 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')
```

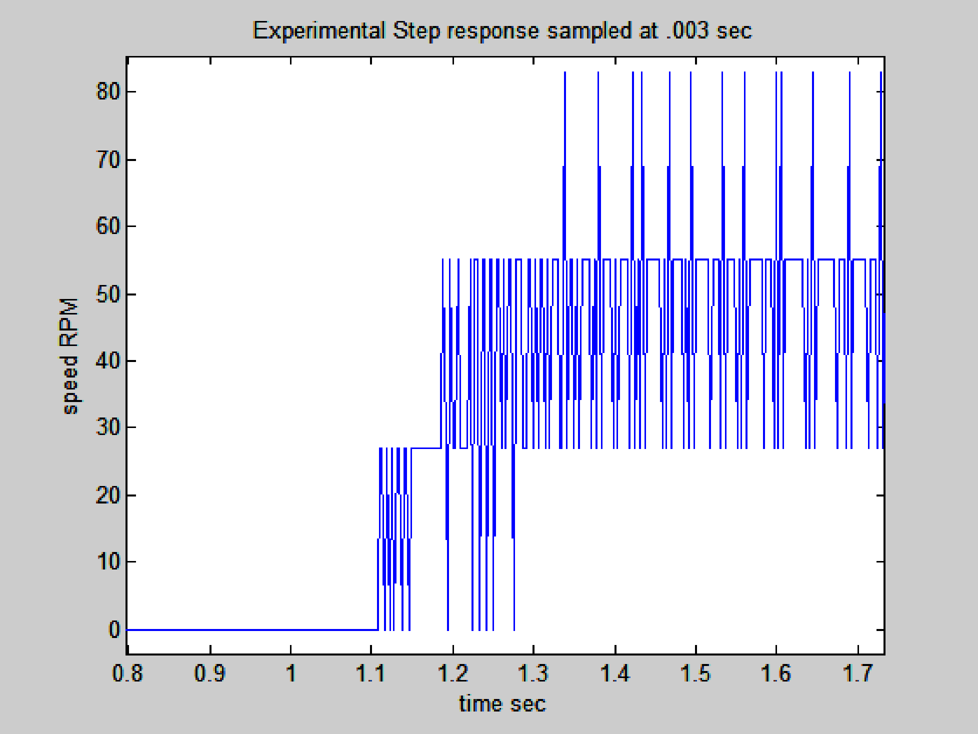
The familiar step response is obtained and the max velocity is seen to be about 170 RPM at 9v:


Ideal 2nd order step response
speed RPM
0
20
40
60
80
100
120
140
160
180
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
time seconds

At 3 volts:


Ideal 2nd order step response
speed RPM
0
10
20
30
40
50
60
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
time seconds

Note that this does NOT look like the step response captured from the experimental step response:

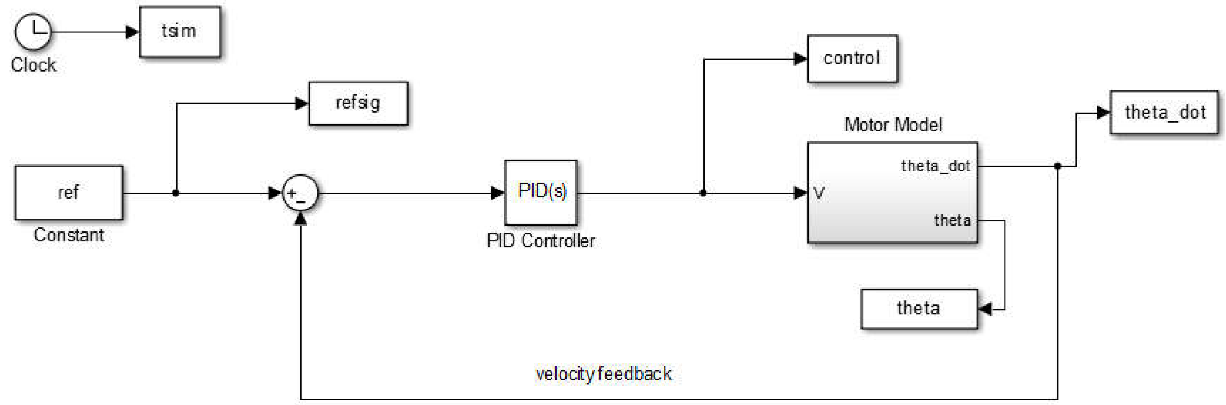


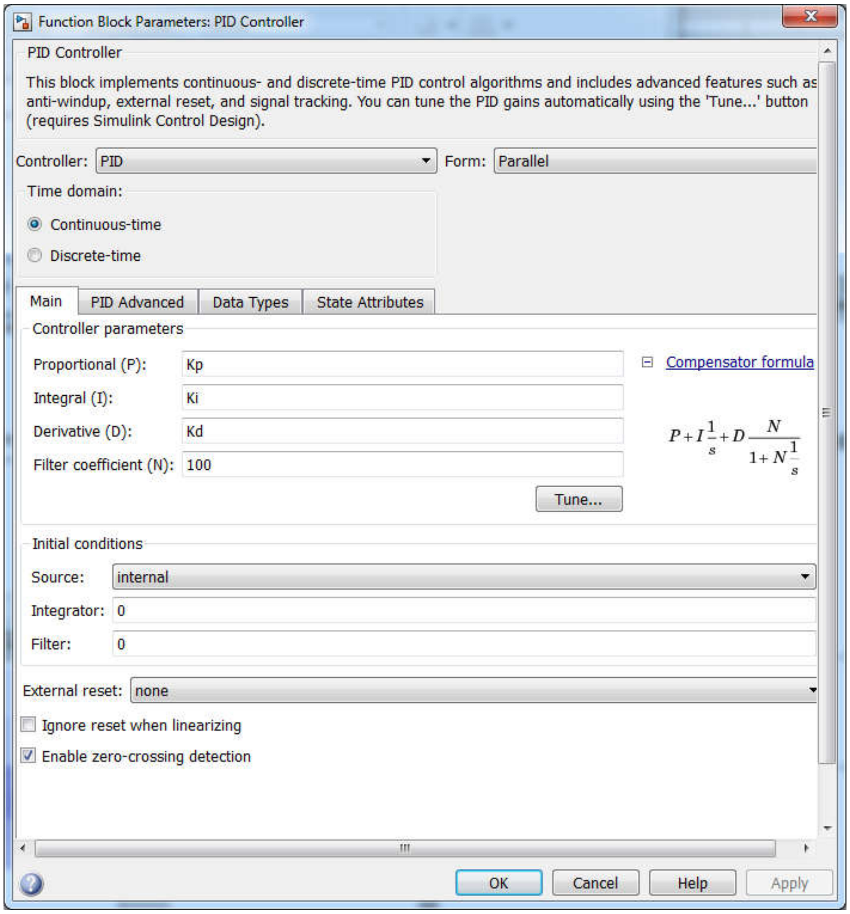
The goal of this lab is to simulate this response by adding in all the hardware characteristics to the simulation to be able to predict this, and examine the effects on the control system.

Part 2: Ideal PI Speed Control of 2nd order DC Motor Model

Background Information:

Using the transfer function for a 2nd order 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

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

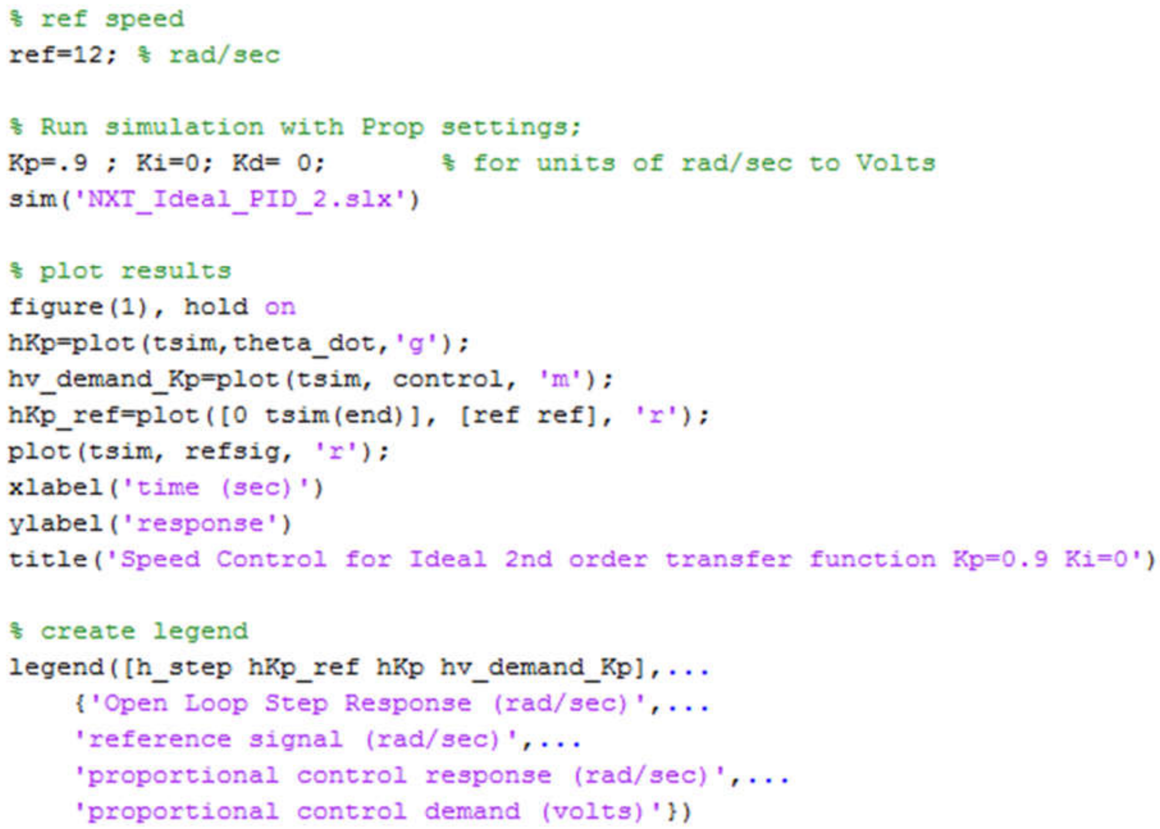
Filter: 0

External reset: none

□ Ignore reset when linearizing

☑ 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.

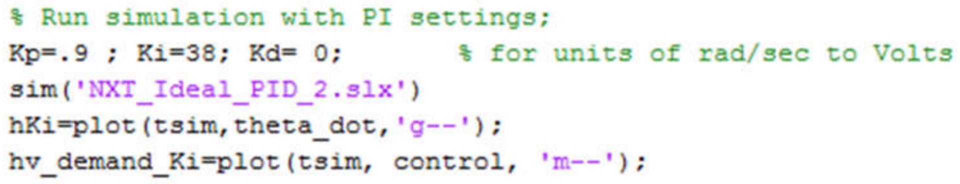


```
% ref speed
ref=12; % rad/sec

% 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:

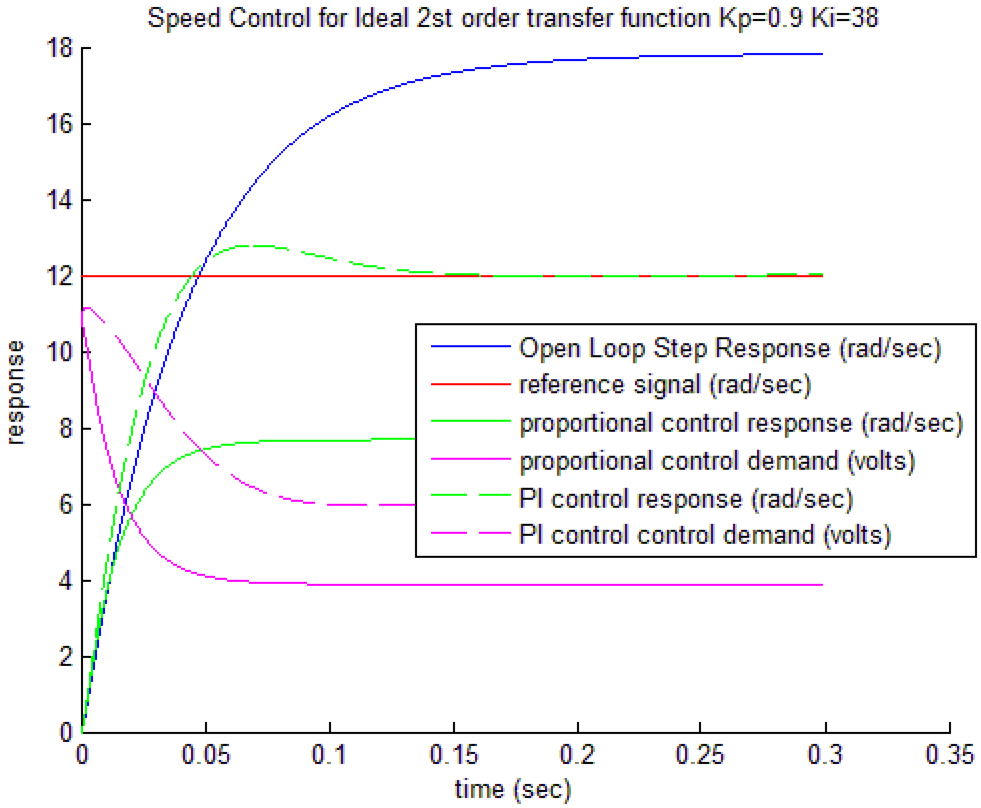


```
% Run simulation with PI settings;
Kp=.9 ; Ki=38; Kd= 0;        % for units of rad/sec to Volts
sim('NXT_Ideal_PID_2.slx')
hKi=plot(tsim,theta_dot,'g--');
hv_demand_Ki=plot(tsim, control, 'm--');
```

The results below show the results for both a proportional controller and for a PI controller.

Questions:

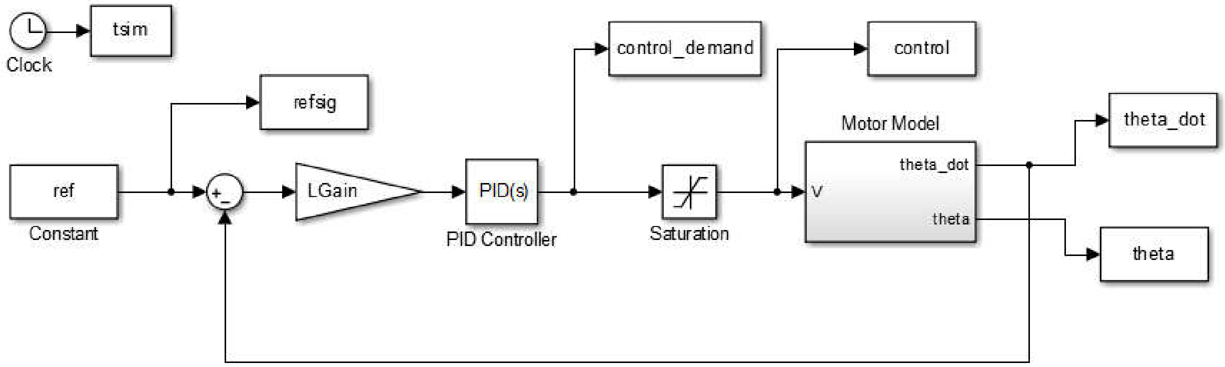
- How does the PI controller perform?



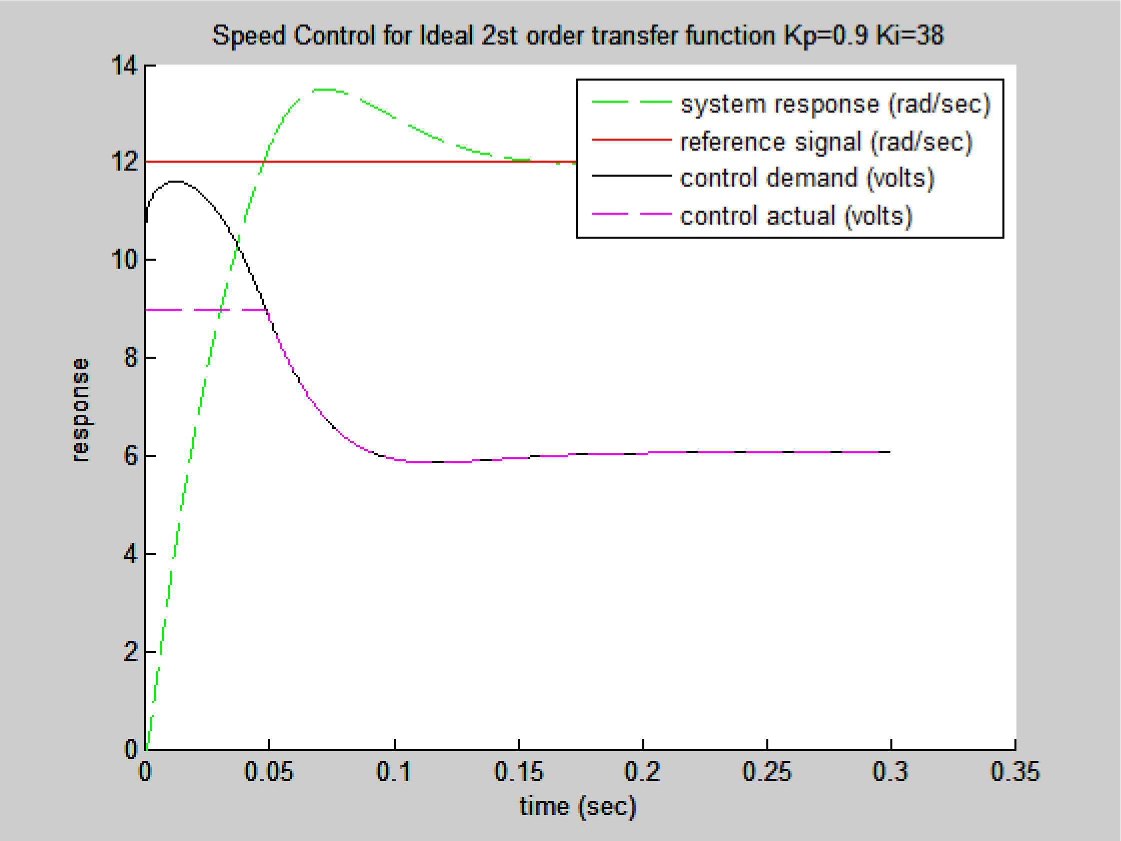
Notice the actual voltage applied (solid pink) is greater than 9v. In the real system this is not possible. It cannot apply more voltage than the available supply.

Part 3: Saturation and Loop Gain

The real system has a limited amount of voltage it can supply. The actual control voltage is limited to this supply voltage, even if the compensator demands more than this amount. To capture this effect a saturation block is used. The total gain of the controller can be decreased as one way to help this effect – in this case a gain LGain is added as an adjustment mechanism. In general saturation of the system should be avoided because the system no longer behaves “linearly” and the response then out of the “predicted” linear response envelope. In practice it may be impossible to avoid saturation so its behavior must at least be predicted.



Create the diagram above and save as a new file. Simulate the same system with an LGain of 1 and a saturation of 9v [adjust the upper and lower saturation limit to ‘+vref ‘/ ’-vref’]. Edit your m-file to plot the results as seen below:



Now even though the controller demands more than 10 volts, only a maximum of 9 volts is applied to the system – just as it would occur in the real system.

Next adjust LGain to a smaller value so that the controller demand does NOT exceed 9 volts.

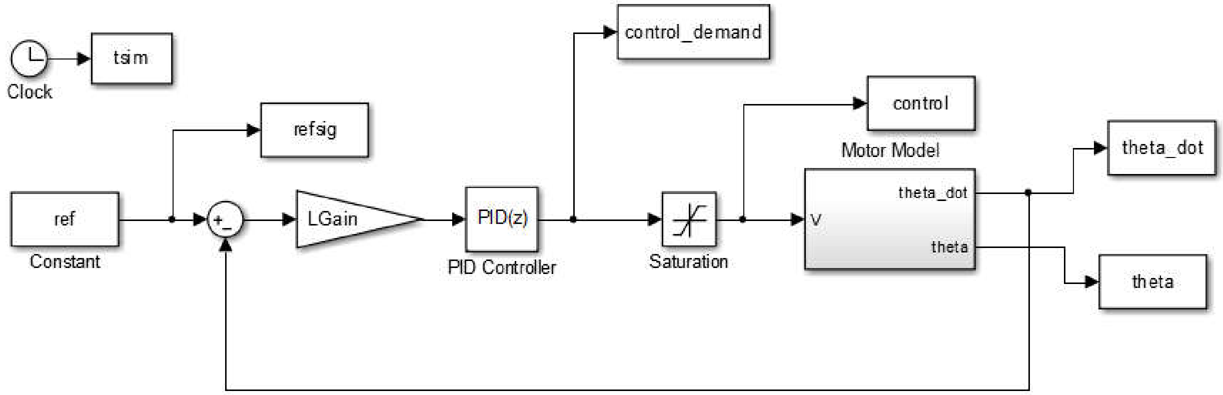
Questions:

- What value of loop gain LGain is needed to avoid saturation?

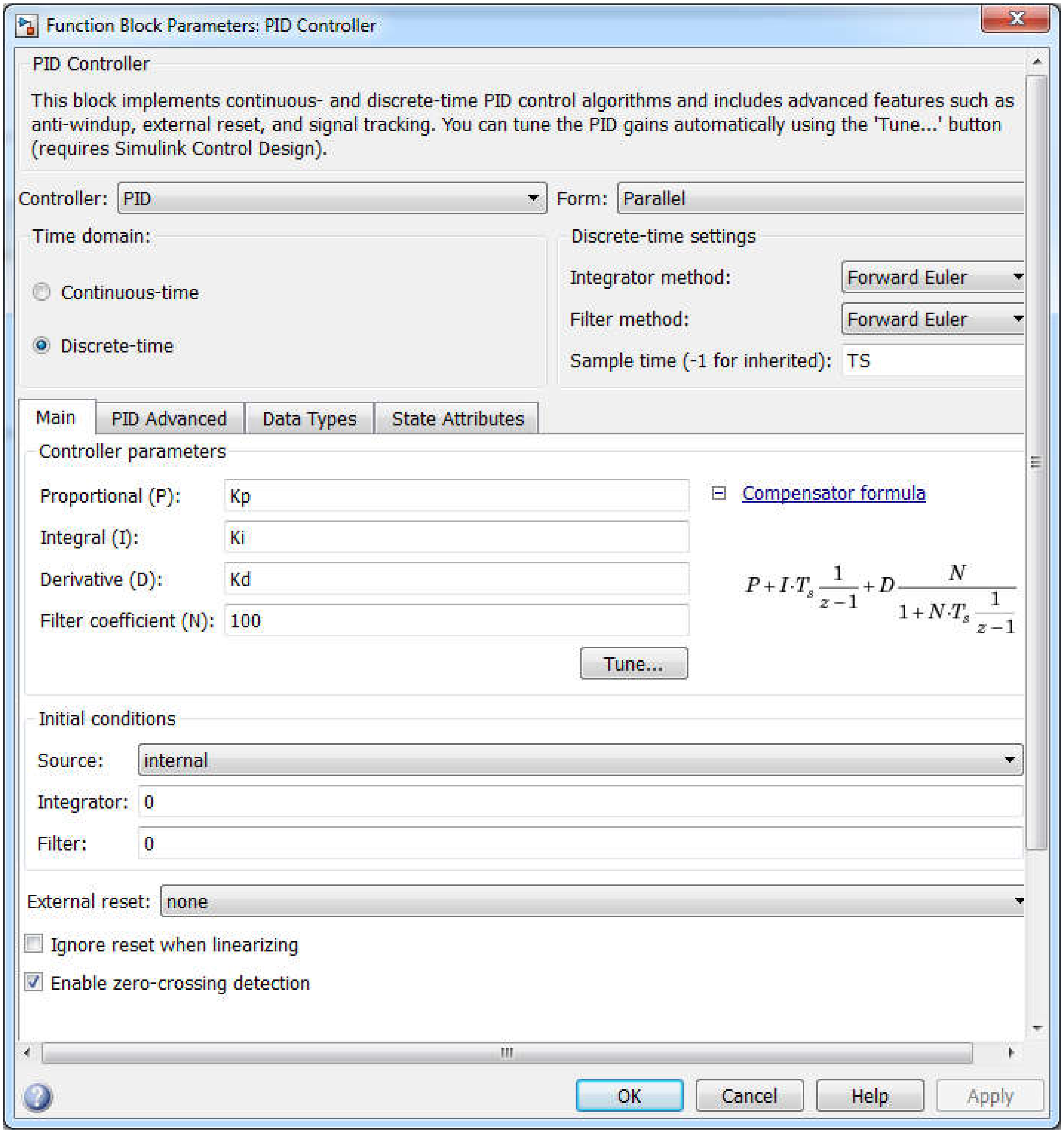
Part 4: Effect of Discrete Controller Sampling

So far the simulation assumes the measurement is obtained instantaneously and the control is applied instantaneously. The hardware is only able to read the sensors, compute the control, and update the control at a certain rate.

To observe this effect the controller is discretized. This way the control can only update at certain times just like in the real system. Use the Simulink diagram from the previous exercise, but save this as a new file to make adjustments.


Clock
tsim
control_demand
control
refsig
Motor Model
theta_dot
ref
Constant
+
_
LGain
PID(z)
PID Controller
Saturation
V
theta_dot
theta
theta

Double-click the PID controller block and change it to discrete time. A variable “TS” is used to indicate the sample time for the hardware controller:



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

Discrete-time settings

Integrator method: Forward Euler

Filter method: Forward Euler

Sample time (-1 for inherited): TS

Controller parameters

Proportional (P): Kp

Integral (I): Ki

Derivative (D): Kd

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

Integrator: 0

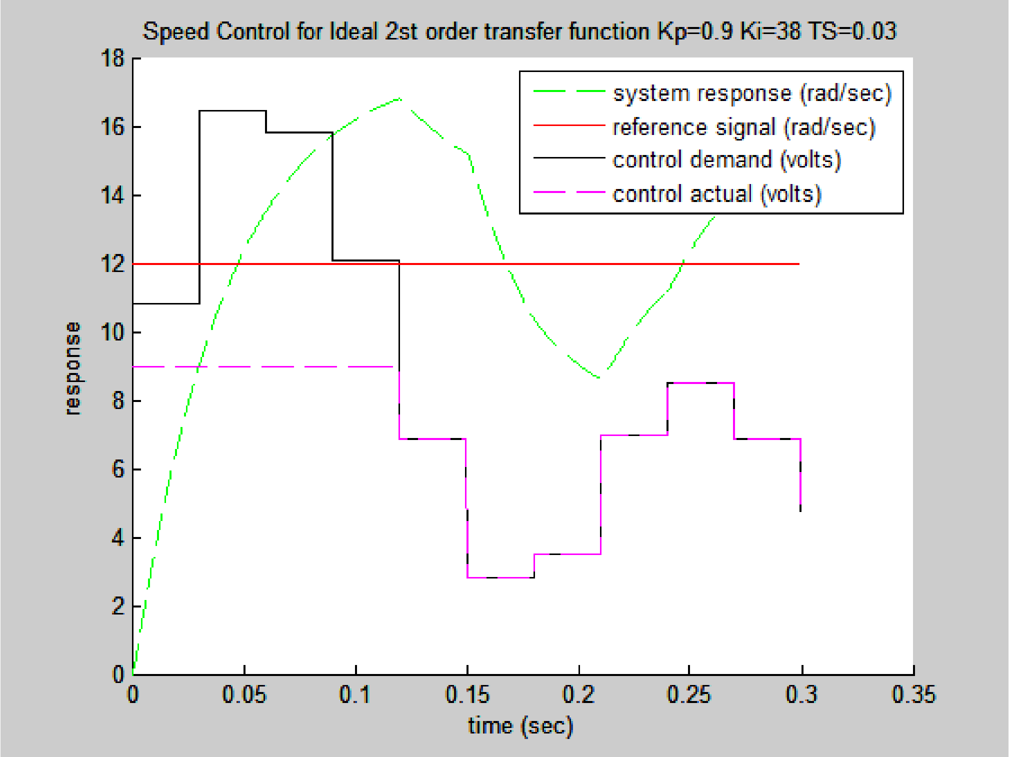
Filter: 0

External reset: none

☐ Ignore reset when linearizing

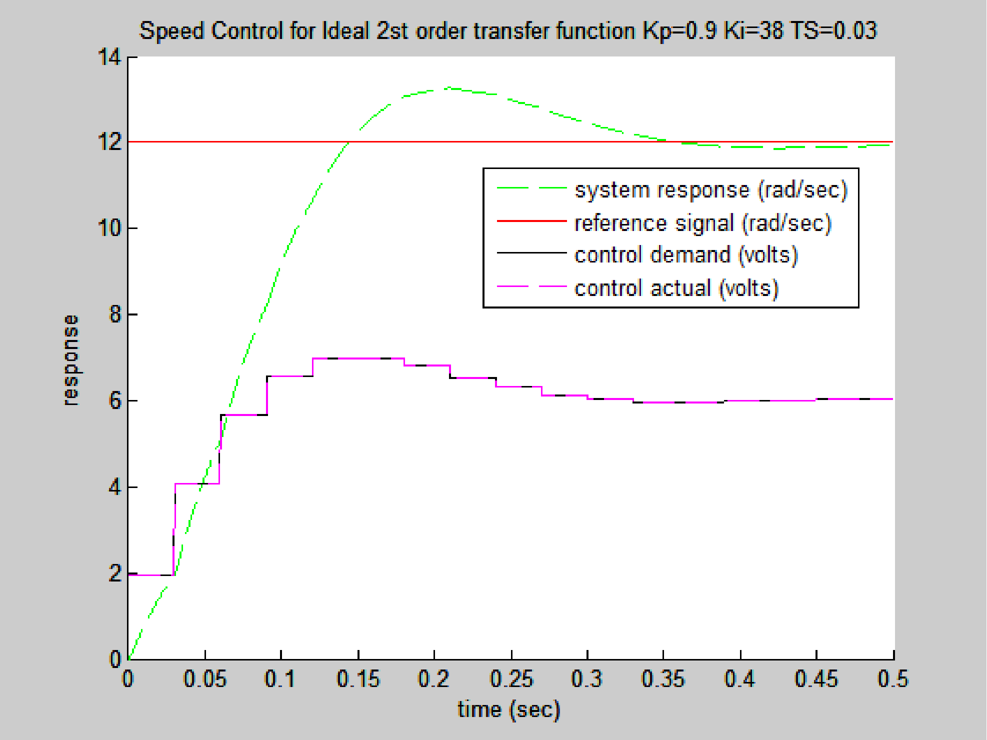
☑ Enable zero-crossing detection

Use a hardware sample time of 30 milliseconds by setting TS=.03. Also change LGain back to 1 to observe the results. Update your m-file and observe the results.



Notice how the system is now unstable with a total gain of 1.

Question: What loop gain is required so that the system is stable and does produces less than 10% overshoot like the figure below?

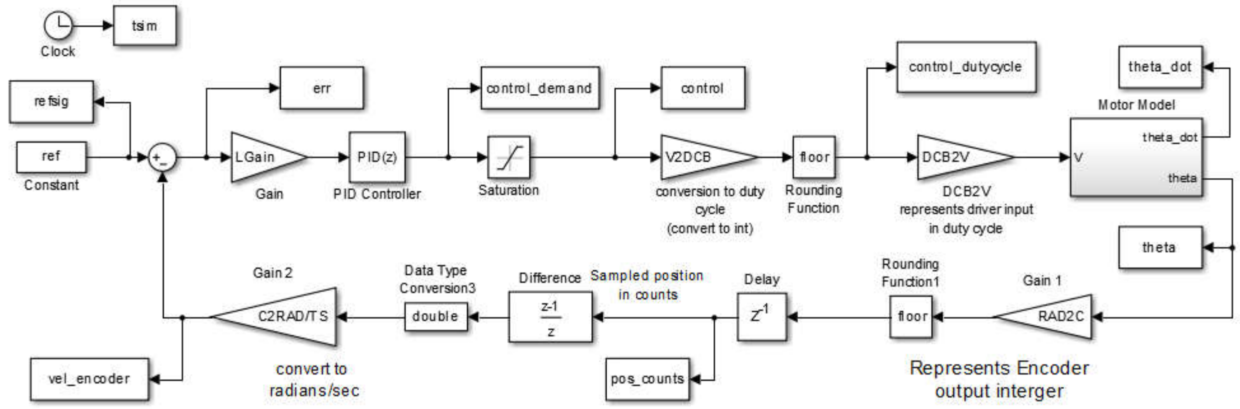


Part 5: Effect of Discrete Measurement

The simulation so far assumes:

- The measurement has infinite resolution and velocity can be obtained directly. In reality the velocity is computed from position which only has 720 counts each revolution.
- The controller is not able to output volts directly. The output is actually a PWM value with 255 discrete settings which will create an average voltage.

The Simulink diagram below captures these effects. Create the model shown:



Note the following items:

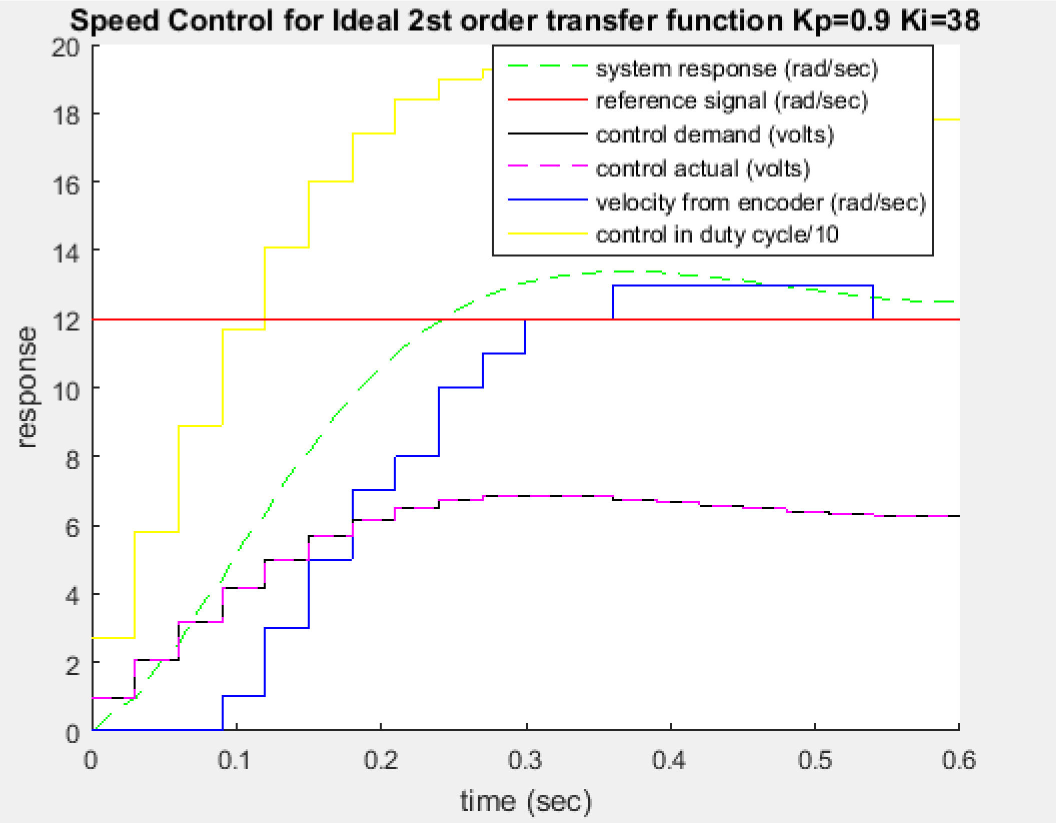
<u>Calculation of PWM value:</u>

- V2DCB (voltage to duty cycle bits) converts the controller output from volts to the equivalent PWM signal value (number from 0-255). This represents the signal value inside the microcontroller. *You need to determine this conversion factor.* This is necessary because it takes the ideal controller value and converts it to the correct PWM signal value.
- The “floor” block rounds the result to an integer since the PWM value has to be discreet values of 0-255
- The second gain DCB2V converts the signal (0-255) back to volts. This represents what happens to the signal after it leaves the microcontroller (it is volts). It is simply 1/V2DCB.

<u>Simulating a discrete encoder:</u>

- The velocity is actually computed from the position. The signal is now taken from theta to represent this.
- RAD2C represents the unit conversion to go from radians (in the ideal world) to discrete encoder counts. *You need to determine this conversion value*
- The "floor" block rounds the result to an integer since the encoder value has to be discreet values
- Z^{-1} is the delay in getting the measurement. It is how fast it takes the hardware to make a reading. In this case it is set to "TS" – the hardware sample time. Set the 'Delay Length' to 1, but make the 'sample time' to "TS".
- A difference block is used to calculate the difference between consecutive samples – the output of this is then a velocity in units of counts/(sample period). This needs to be converted back to the units the controller will be using which is radians per second.

Simulate this system with the correct gains using the LGain from the last exercise to ensure it is stable. Update your m-file to plot the following results:



Which is very similar to the previous response. When the sampling time is 30 milliseconds, this delay is the primary factor reducing the performance. As this delay is reduced, these other factors will become more important. This simulation tool can be used to investigate all of these variables.

Now change the sampling time to 3 milliseconds and plot the results.

Question: Provide a plot of the final simulation with a 3 milliseconds sample time. How does this response compare to the experimental response presented in the first section?

Part 6: Verification of Real World System Performance

You have now modeled the response of the system using a variety of factors, including PI controller sampling rate, ADC and DAC resolution, and many of the physical characteristics of the motor itself. It is now time to compare the actual performance of the system compared to its step response model.

Modifying the Simulink diagram so external mode can be used to obtain the response from the actual hardware system. The model from part 5 is modified to produce the following. You will have to modify the driver block and encoder block for your hardware system and library version.

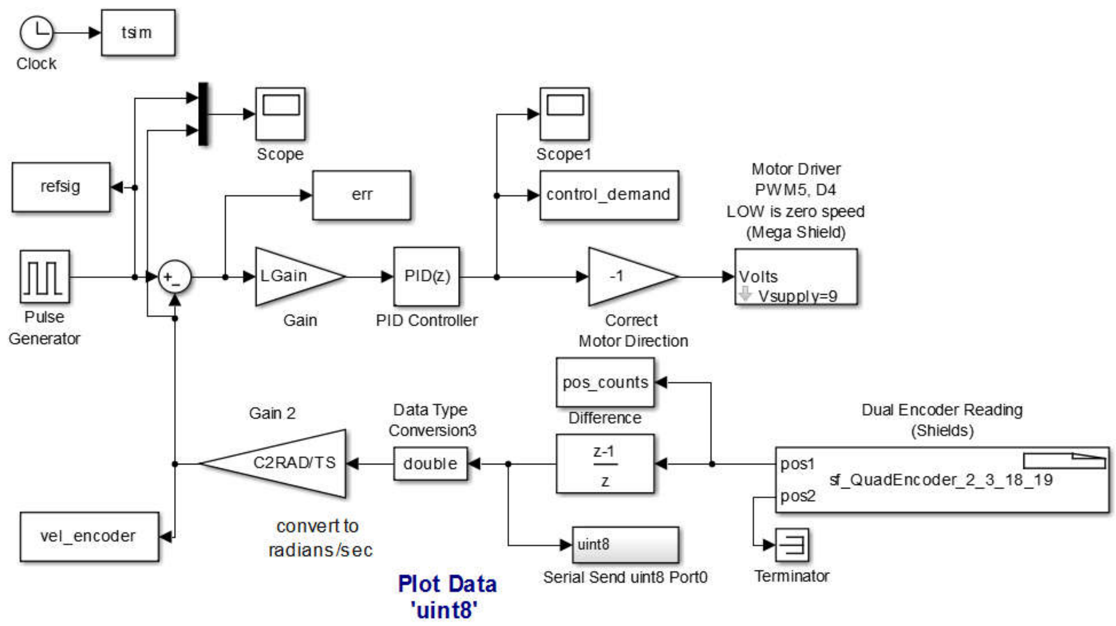
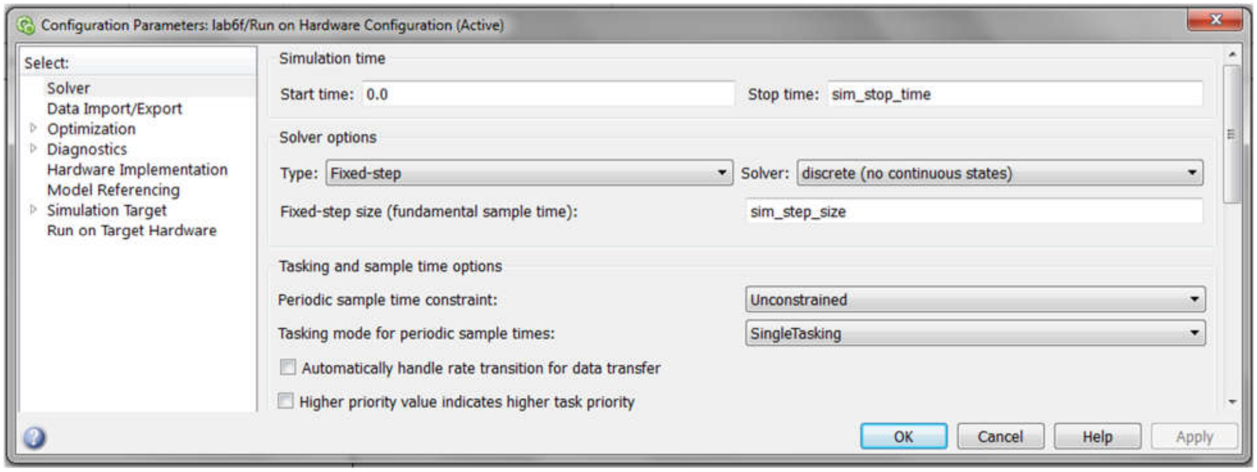


Figure 3: Hardware response - you will have to update the motor output blocks and encoder blocks for your hardware system

Compare this to the schematic for exercise 5. Which differences stand out?

- Depending on your hardware system you may have to change the motor direction to ensure it spins in the correct direction
- The pulse generator allows the system to cycle on and off during the chosen duration if external mode is not used
  - Newer verions of matlab (2015a) external mode should work okay at .003 seconds

    - Verify this using dierct serial commication
- Double click the ‘Pulse Generator’ icon, and set the amplitude to ‘ref’, the variable you will control in your script file. Set the pulse width to 50%, and a period of your choosing (6 seconds used in example).
- Make sure that the model is set to ‘fixed-step’ under the ‘solver’ pane, and set the step size and stop time to variables seen below. Now save the model as a new file.



Configuration Parameters: lab6f/Run on Hardware Configuration (Active)

Simulation time

Start time: 0.0

Stop time: sim_stop_time

Solver options

Type: Fixed-step

Solver: discrete (no continuous states)

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: SingleTasking

□ Automatically handle rate transition for data transfer

□ Higher priority value indicates higher task priority

- Plot the serial data, and find the start and end points of a pulse, and make note of the x-values. Because the serial time data is not synced to the ideal simulation, you will have to manually adjust the plot to compare them.

Questions: Provide a plot of the simulated response, and external, and the serial data. How do these results of external and direct serial compare?