RENSSELAER MECHATRONICS

Basic DC Motor Control with SISOtool, Full State Feedback

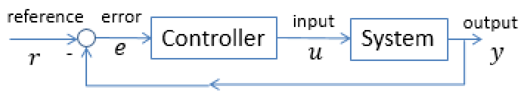
Objectives:

- Design a SISO compensator to meet transient specifications
- Use Full State Feedback (FSF) to meet the same transient specifications
- Address steady-state error with FSF
- Implement LQR FSF

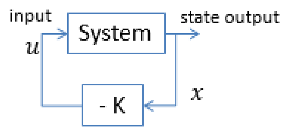
Background Information:

The standard SISO controller uses the output compared to a reference. A full state feedback compensator uses the complete state multiplied by a gain matrix as the control law $u = -Kx$. If the reference state is zero this compensator is referred to as a regulator.

Standard SISO Closed Loop

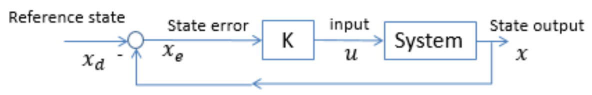


Full State Feedback – regulator (zero reference)

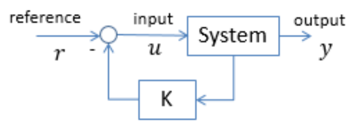


If the reference is not zero, the easiest way to add a reference is to add a desired reference state x_d. Another way is to add a reference signal the same size as the input, but this requires additional design considerations:

Full State Feedback – with state reference



Full State Feedback – with reference



All of these different architectures have the same basic control law $u = -Kx$.

SISO controller design:

Design a standard SISO controller (using SISOTOOL in MATLAB) for the position control of a DC motor such that the closed loop poles satisfies a settling time of 0.3 seconds and 5% overshoot for a unit step response. Use a DC motor model with $K = 2, \tau = 0.07$. Ignore potential saturation (assume infinite control effort available) and other hardware limitations (use a PD compensator – no extra pole)

Questions:

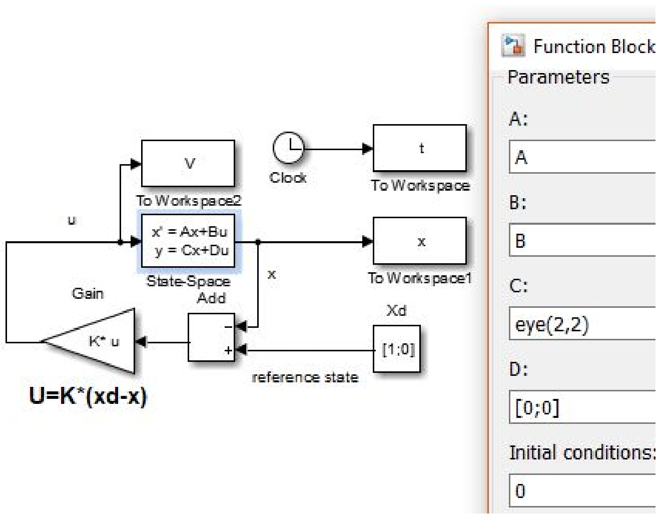
- What is the zeros, poles and gains of your compensator (in SISOTOOL choose 'Edit', 'SISOTOOL Preferences', 'Options', 'Zero/Pole/Gain')?
- Where are the closed loop poles and zeros located?
- Plot the step response for a unit step
- What is the actual percent overshoot and settling time? If it is different than the designed overshoot, explain why.
- Write out the control law of your resulting compensator (multiply everything out):
    - $u =$
- What can you say about the magnitude of the control effort initially (based off the above control law)?

Full State Feedback:

Given the model transfer function: $\frac{\theta}{V} = \frac{K}{s(\tau s+1)}$ and state $x = \left[\theta\ \dot{\theta}\right]^T$,

- What are the state space matrices A, B?
- Using the 'place' command in MATLAB what is the gain matrix K required to meet the same transient specifications as the previous section?

Implement this controller in Simulink:



A: A

B: B

C: eye(2,2)

D: [0;0]

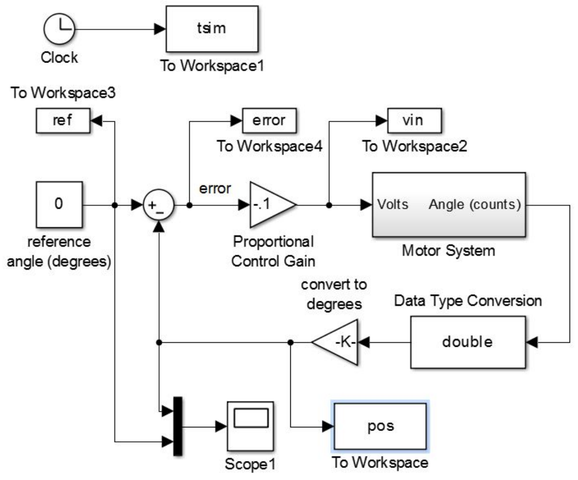
Initial conditions: 0

The step input (step in position θ) is implemented by adding a reference state $x_d = [1\ 0]^T$. The control effort is still of the form $u = -Kx$. Notice the C matrix is identity so that the State-Space block has the full state as the output $x = \left[\theta\ \dot{\theta}\right]^T$.

- Plot the step response for a unit step
- What is the actual percent overshoot? If it is different than the designed overshoot, explain why.
- Plot the control effort. What is the magnitude of the control effort initially?
- Write out the control law of your resulting compensator (multiply everything out):
    - $u =$
- What can you say about the magnitude of the control effort initially (based off the above control law)? Why is this different from the previous section?
- Compare the control laws from this section and the previous section – what can you say about what type of compensator a FSF is?
- If you have finite control effort of 7.8 volts (the SN754410 driver and a 9v battery) what is the largest step change in the reference position possible to avoid saturation (in degrees)? What is the max change in reference position if the 3.25v are available? (USB Power)

Hardware Verification:

- Modify the previous Full State feedback simulation so it can be deployed to the hardware with the USB as the voltage supply.
    - You will want to copy and paste the code to a demo file that is set up to work with the hardware. You are now running the controller on the hardware NOT solving an ODE, so the solver parameters need to be set up correctly. Use the fastest sampling time you can to reduce the delay and get a better match (use overrun detection to make sure you can run the code that fast). Be sure to specify the sample time you use.
    - It is recommended to make subsystem for the hardware system so it can easily replace your simulation plant as was done in the Basic DC Motor Control Lab:



- Plot the response to the max step change to avoid saturation
- What is the percent overshoot? What is the approximate settling time? What is the steady state error?
- Why are these different?

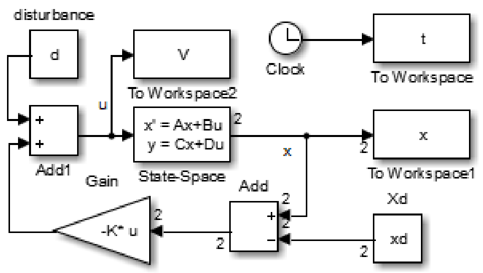
Eliminating Steady State Error:

1.) Repeat the design process for the SISO compensator to eliminate steady sate error and verify simulated performance:

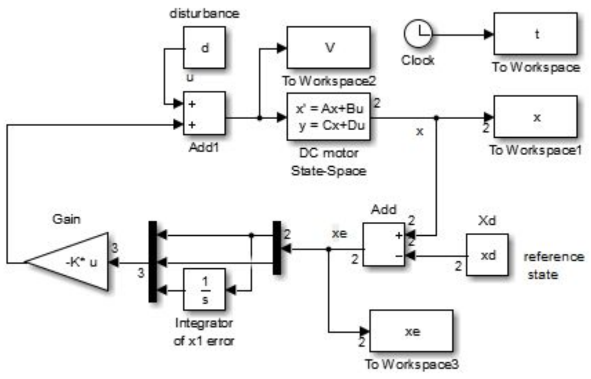
- Modify the SISO compensator from the first part with a pole and zero 'close' to the origin
- Write out your final compensator transfer function zeros, poles, and gain
- Plot the simulated step response with the PID compensator

2.) Repeat the design process for a full state feedback compensator to eliminate steady sate error and verify simulated performance:

- Modify the previous simulation diagram to include a disturbance (d=1)



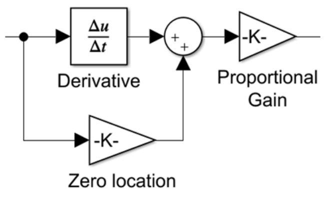
- Simulate this response with your previous controller and observe the steady state error. Plot the response – clearly label all axes.
- Design a full state feedback compensator to eliminate the steady sate error in the first state
- Simulate this response with your new controller to observe the steady state error going to zero:



- Plot the response and clearly label all axes

Hardware Verification:

- Implement the PID compensator on the hardware – Provide a plot of the step response and show the steady state error – clearly label all axes, and note all important locations and observations. A simple PD compensator $K_p(s + z_c)$ can be implemented with a discrete derivative block as follows:



- Implement the FSB with integrator on the hardware – Provide a plot of the step response and show the steady state error – clearly label all axes, and note all important locations and observations.