Control Challenges: Solutions

Iacopo Moles

Table of contents

# 1. Introduction

## 1.1 What is this?

This is a collection of write ups on how to solve the various problems presented by [Github user](https://janismac.github.io/ControlChallenges/) “Janismac”.

# 2. Block With Friction

Position Control with friction. Using Pole Placement + PD.

## 2.1 State Space representation

We can convert the set of ODE into a state space representation. The final bode plot of the block position is:

using DiscretePIDs, ControlSystems, Plots, LinearAlgebra  
  
# System parameters  
Ts = 0.02 # sampling time  
Tf = 2.5; #final simulation time  
g = 9.81 #gravity  
α = 0.0 # slope  
μ = 1.0 # friction coefficient  
x\_0 = -2.0 # starting position  
dx\_0 = 0.0 # starting velocity  
τ = 20.0 # torque constant   
  
# State Space Matrix  
A = [0 1 0  
 0 -μ 1  
 0 0 -τ];  
B = [0  
 0  
 τ];  
C = [1 0 0  
 0 1 0];  
  
sys = ss(A, B, C, 0.0) # Continuous  
  
plot!(bodeplot(tf(sys)),pzmap(tf(sys)))

|  |
| --- |
| Figure 2.1: Starting Bode Plot |

It has the shape we expect from a motor + friction. Slow pole for the mass + friction and a faster pole for the current & inductance.

Numerically they are:

display(eigvals(A))

|  |
| --- |
| 3-element Vector{Float64}:  -20.0  -1.0  0.0  Figure 2.2: Starting PZ map |

We see that we start with all the pole in the left-half plane, which is good.

## 2.2 Pole Placement

We can design a controller with pole placement.

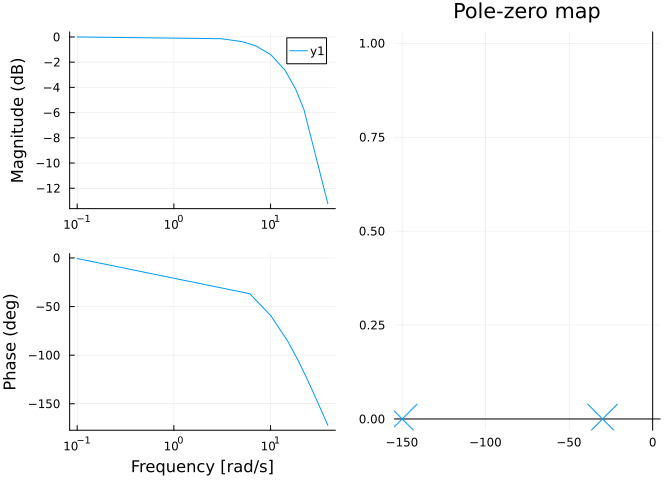
For some reason pole placement doesn’t work for the observer, I use a Kalman Filter with random fast values.

observability(A, C).isobservable & controllability(A, B).iscontrollable; #OK  
  
ε = 0.01;  
pp = 15.0;  
poles\_cont = -2.0 \* [pp + ε, pp - ε, pp];  
L = real(place(sys, poles\_cont, :c));  
  
poles\_obs = poles\_cont \* 5.0;  
K = place(1.0 \* A', 1.0 \* C', poles\_obs)'  
cont = observer\_controller(sys, L, K; direct=false);

We can check the effect of the new controller on the loop

closedLoop = feedback(sys \* cont)  
print(poles(closedLoop));  
setPlotScale("dB")  
plot!(bodeplot(closedLoop[1, 1], 0.1:40), pzmap(closedLoop))

ComplexF64[-29.980000105166976 + 0.0im, -29.999999789554334 + 0.0im, -30.020000105278555 + 0.0im, -150.00000000000009 + 0.0im, -150.09999999988327 + 0.0im, -149.90000000010167 + 0.0im]



We can compare this to the open-loop response in [Figure 2.1](#fig-bode). We can see that we achieve unitary gain throughout the whole low-frequency range.

We can convert the pole placement controller into the standard PD gain form.

K = L[1];  
Ti = 0;  
Td = L[2] / L[1];  
pid = DiscretePID(; K, Ts, Ti, Td);

## 2.3 Simulation

We can simulate this with a motor that only outputs the position:

sysreal = ss(A, B, [1 0 0], 0.0)  
ctrl = function (x, t)  
 y = (sysreal.C\*x)[] # measurement  
 d = 0 \* [1.0] # disturbance  
 r = 2.0 \* (t >= 1) # reference  
 # u = pid(r, y) # control signal  
 # u + d # Plant input is control signal + disturbance  
 # u =1  
 e = x - [r; 0.0; 0.0]  
 e[3] = 0.0 # torque not observable, just ignore it in the final feedback  
 u = -L \* e + d  
 u = [maximum([-20.0 minimum([20.0 u])])]  
end  
t = 0:Ts:Tf  
  
res = lsim(sysreal, ctrl, t)  
  
plot(res, plotu=true, plotx=true, ploty=false)  
ylabel!("u", sp=1);  
ylabel!("x", sp=2);  
ylabel!("v", sp=3);  
ylabel!("T", sp=4);

For more stats:

si = stepinfo(res);  
plot(si)  
title!("Step Response");

We can also simulate it in a SIMULINK-like environment:

using FMI, DifferentialEquations  
fmuPath = abspath(joinpath(@\_\_DIR\_\_,  
 "..",  
 "modelica",  
 "ControlChallenges",  
 "ControlChallenges.BlockOnSlope\_Challenges.Examples.WithFriction.fmu"))  
fmu = loadFMU(fmuPath);  
simData = simulateME(  
 fmu,  
 (0.0, 5.0);  
 recordValues=["blockOnSlope.x", "blockOnSlope.xd", "blockOnSlope.usat"],  
 showProgress=false);  
unloadFMU(fmu);  
plot(simData, states=false, timeEvents=false)



There is a slight difference between the lsim simulation and the FMU simulation. I need to recheck some stuff.