16-299 Assignment: Lab Assignment 1

Part 1

Make a state space model of the motor-wheel system, using any technique you like.

I would like to see a description of how the students used data, either the data given to them, or data they collected, to make a model. Even better would be evaluation of that model on a different set of data.

Continuous or discrete time models are okay.

Frequency domain or direct regression approaches are ok.

A minimal model would relate current wheel angle, angular velocity, and command to future wheel angle and angular velocity.

I used frequency domain approaches. I used the "Sinusoids" data to make a frequency response (Bode plot), and then manually adjusted model parameters until the frequency response of the model matched that of the data.

`http://www.cs.cmu.edu/~cga/elegoo/collecting.html` talks about estimating the frequency response.

`http://www.cs.cmu.edu/~cga/elegoo/modeling.html` talks about fitting a model to the frequency response.

My best model added the previous command to the state:

```
% State is (pos, vel, trq), command is (u)

Ts = 0.002;
c0 = 0.029;
c1 = 0.0027;
c2 = 0.75;

A = [ 1 Ts 0
0 (1-c0) c1*c2
0 0 0 ]

B = [ 0
      c1*(1-c2)
      1 ]

C = [ 1 0 0 ]

D = [ 0 ]

sys = ss( A, B, C, D, Ts );
```

```
w = logspace(-1,1.7,50);
H = freqresp( sys, w, 'Hz' );

figure(1)
hold off
loglog( w, abs( reshape( H(:,:,:), 1, [] ) ) )
% hold on
% loglog( freq_response(:,1), freq_response(:,2) )
% legend('model','reality')
title( "Magnitude ratio angle/torque" )
xlabel( "Log frequency (Hz)" )
ylabel( "Log ratio" )


figure(2)
a = angle( reshape( H(:,:,:), 1, [] ) );
for i = 1:length(a)
  if a(i) > 0
    a(i) = a(i) - 2*pi;
  end
end
hold off
semilogx( w, a )
% hold on
% loglog( freq_response(:,1), -freq_response(:,3) )
% legend('model','reality')
title( "Phase lag of angle wrt torque" )
xlabel( "Log frequency (Hz)" )
ylabel( "Phase lag" )
```

The plots with the frequency response of the actual setup are the last plots on `http://www.cs.cmu.edu/~cga/elegoo/modeling.html`.

Part 2

Part 2A: Design a feedback-only PID controller to move each wheel 180 degrees (1/2 revolution). Where does the velocity signal come from? How quickly can you complete the movement? The end of the movement is when the wheel remains within 5 encoder counts of the target.

Any PD gains are okay. The velocity signal should come from a differentiating filter, ideally with a low pass component.

I presented some data on `http://www.cs.cmu.edu/~cga/controls-intro/lab1/`. In the data I presented the fastest movement was about 100 samples or 200 milliseconds. The key point is that one had to introduce damping to go fast, otherwise the oscillations at the end would delay satisfying the movement end criterion. The command is almost always saturated during the movement. The fastest movement would be a bang-bang control signal, followed by a PD controller to wipe out any residual motion.

Part 2B: How well does your controller controlling the actual robot match the same controller controlling a model in simulation?

I am expecting a comparison of an actual movement with a simulation. I expect the matches to be pretty good.

Part 3

Design and implement a feedback only state space controller for the motor-wheel system based on your estimated model. Does the state space controller match the feedback controller you designed by hand?

I am expecting to see the use of dlqr(A,B,Q,R). I don't really care what Q and R the students pick.

How quickly can you complete a 180 degree movement? The end of the movement is when the wheel remains within 5 encoder counts of the target. Note that as long as the command is saturated what feedback controller is in use won't matter. The controller only matters for quickly slowing down the movement after the initial command saturation is over.

Is the state space controller "better" than a manually designed feedback controller? When would the difference be greater?

Because the command is saturated for fast movements, the details of the controller don't matter.

With the model I created, an LQR controller introduces too much damping. Why might this be the case, aside from my model is bad?

The model does not take into account static friction, so it adds unnecessary damping. Other answers are okay, don't weigh this part of the question very highly.

Part 4

Does adding an observer to your state space controller such as a Kalman Filter to filter noise and estimate missing states help? A Kalman Filter should introduce much less delay that a simple low pass filter, since it uses a model of the process dynamics to predict future states such as velocity.

I want to see an implementation of a Kalman filter: Define a C matrix. Compute dlqr(A',C',Qf,Rf).

Here are the results from my implementation, where the Kalman Filter has much less delay, except towards the end of the movement. Any ideas why that might be happending?

See the figure on `http://www.cs.cmu.edu/~cga/controls-intro/lab1/` that indicates the delay of the low pass differentiating filter that is avoided by the Kalman filter. Again, static friction is not modeled in the linear model used by the Kalman filter, so the real system slows down faster than the model at low speeds.

How quickly can you complete a 180 degree movement now? Can you attain higher velocity gains, and thus higher position gains? The end of the movement is when the wheel remains within 5 encoder counts of the target. Note that as long as the command is saturated what feedback controller or observer is in use won't matter. The observer only matters for quickly slowing down the movement after the initial command saturation is over.

Because the command is almost always saturated, no big improvements in movement duration are possible. Higher velocity gains should be achievable, which help damp out motion at the end of the movement.

Part 5

Add a feedforward command. Does this improve performance, in terms of how quickly can you complete a 180 degree movement? The end of the movement is when the wheel remains within 5 encoder counts of the target. Note that as long as the command is saturated feedforward commands won't make any difference. However, a bang-bang trajectory, where negative command saturation slows down the movement, is likely to be the way to make the fastest movement to a goal. The figure below shows that fast movements driven by a feedback controller begin with a saturated command, and going faster makes the negative part of the command almost saturated as well. In this example the velocity gain was limited by instability, so I couldn't make the negative pulse saturated with only feedback control.

Feedforward control enables true bang-bang control. That bang-bang control can be created by manually choosing the pulse timings.

Bonus questions

I did not do this yet.