Homework 3 – State Estimation and Project Proposal

Due Nov. 4, 2011

This homework involves two parts.

1) In part I, you are asked to use MATLAB to simulate a passive pendulum that measures angular position as an output. This measurement is corrupted by an unknown encoder offset, analogous to the offset our encoders in lab have (since the encoders are not indexed). Your system will have three states: measured angle (i.e., angle minus offset), angular velocity, and angle offset (which remains constant). The actual pendulum angle is the sum of the first and third states. The system will have one output: measured angle (state 1, only).

Parameters:

- Assume the **plant** has an undamped natural frequency of 10 rad/s, with damping ratio (zeta) of 0.02. Assume the actual encoder offset is a constant 10 degrees.
- Assume the **model** has an undamped natural frequency of 6 rad/s, with damping ratio (zeta) of 0.1. Assume the model of encoder offset is initially 0 degrees. (*This offset is a state you will estimate during simulation.*)
- Use a time step of T=0.001 seconds for simulation. You may use the c2d function in MATLAB to create a discrete-time state space system for simulation. (See attached pages for a suggested template, if desired…)
- Start with an initial condition of X0 = [20; 0; 10]*(pi/180)

a) Estimate all 3 states over time using a standard observer, L. Set the poles of the observer to be about 5 times faster than the poles of the model.

b) Now, estimate all 3 states using a Kalman filter. Set Q, R, and P0 such that results seem "reasonable". Note that only a few lines of code need to be modified to implement b), once you have completed part a).

For each part, a) and b), turn in a one-page figure with the following sub-plots:

- Actual and estimated pendulum angle over time on a single subplot (state 1 plus state 3)
- Actual and estimated angular velocity (state 2)
- Actual and estimated encoder offset (state 3)

2) For problem 2, you should submit a short project proposal (1/2 to 1 page in length). If you have a partner, only one of the two of you need submit this proposal. (Just list both names on the proposal, of course.) You should

1. specify your intended hardware and control goals,
2. list (briefly) basic steps you plan to use to get to your goal (e.g., system ID, modeling in MATLAB, designing an estimator or using other signal filtering strategies, control techniques are might try, etc…),
3. identify potential "stumbling block(s)" you think might be particularly challenging (if any), and
4. if there is significant "risk" you can identify in part 3, list a potential "back-up plan".

Parts 3 and 4 would mostly apply if you trying to control a system no one has ever controlled before (for example), such as hardware for research. They may also apply if you are trying to do something particularly unique, however, such as playing ping pong with the acrobat, etc.

```
clear all       % to be sure what you are doing...
format compact  % for better readability.

% ECE 238
%
% This version include ENCODER DISCRETIZATION
%
% A number of people have asked about "zeroing out" any
% unintended offset in encoder measurement. That is, since
% we have no "index" telling us the absolute angle of the
% encoder output, the initial angle when we begin real-time
% control is the angle the sensor defines as "zero" --
% and this MAY be off from "exactly up" or "exactly down"
% (or however you want to start the system).
%
% This modified version of Assignment 4 includes an
% additional "state", which is the (unintended) OFFSET ANGLE
% that must be added to the encoder to obtain the TRUE ANGLE
% (here, i.e., wrt exactly DOWN-pointing, for our passive
% pendulum...)
%
% This "state" is really just a constant, but its value
% affects the acceleration of the pendulum (because it
% contributes to the TRUE angle of the pendulum).
% This offset angle state will be updated only if we include
% some assumption about noise (in updating its true value)
% in our KALMAN FILTER.
%
% Figure 2 plots the actual and estimated offset angle (top)
% and the absolute angle (which is measurement plus offset, bottom).
%-------------------------------------------------------
% First, we define an ACTUAL plant and a MODEL of the plant.
% These SHOULD be the SAME, in general, but we also want to
% examine how sensitive our results are if they are DIFFERENT.
% In variable names:
%    "c" stands for "continuous" time. "d" for "discrete".
%    "p" stands for "plant", and "m" stands for "model".

% === Define the PLANT and MODEL dynamics:
wn_p = 5; % actual (p="plant") natural frequency of pendulum, rad/s
zeta_p = .01;  % damping ratio--> d2x + 2*zeta*wn*dx + wn^2*x
wn_m =   ;  % modeled (m="model") natural frequency
zeta_m =    ;
% ------ PLANT:
% include an "offset error" in (measurement of, via encoder) angle:
Acp = [0 1  ; -wn_p^2 -2*zeta_p*wn_p        ;      ];  % CONTINUOUS TIME PLANT dynamics
Bcp = [0; 1; 0];
Ccp = [1 0 0];
Dcp = [0];
sscp = ss(Acp,Bcp,Ccp,Dcp);  % CONTINUOUS TIME state-space PLANT system
% ------ MODEL:
Acm = [0 1  ; -wn_m^2 -2*zeta_m*wn_m        ;      ];  % CONTINUOUS TIME MODEL dynamics
Bcm = [0; 1; 0];
Ccm = [1 0 0];
Dcm = [0];
```

Oct. 25, 2011

P1

```
56 sscm = ss(Acm,Bcm,Ccm,Dcm);  % CONTINUOUS TIME state-space MODEL system
57
58 % Now, simulate PLANT and MODEL as a discrete-time systems:
59 T = 1e-3;  % Sample time (seconds)
60 ssdm = c2d(sscm,T);
61 ssdp = c2d(sscp,T);
62 Adm = ssdm.A; Bdm = ssdm.B; Cdm = ssdm.C; Ddm = ssdm.D; % MODEL
63 Adp = ssdp.A; Bdp = ssdp.B; Cdp = ssdp.C; Ddp = ssdp.D; % PLANT
64
65 % Create an estimator given a "no noise" assumption:
66 pc = eig(Acm) % poles of plant
67 pc_dominant = min(abs(pc))  % SLOWEST pole of dynamic system; should be wn
68 rel_est_speed = 5;    % make estimator 2-6x faster than plant dynamics
69 pe = -rel_est_speed*pc_dominant*[1 1 1];  %
70 % Below, calculate observer gains (NOT Kalman filter; no noise assumed)
71 Ld = acker(Adm',Cdm',exp(T*pe))'  % match poles, in discrete-time sys
72
73 % Initialize state and other variables for simulation loop:
74 encoder_error =            % offset in actual "zero" location...
75 x0 = [20*pi/180; 0; encoder_error];  % angle and velocity
76 y0 = 0;                  % start with false estimate of state
77 ndo = 1e4;             % Number of time steps in simulation
78 t = [0:ndo-1]*T;       % Corresponding time vector
79 x = 0*x0*t;            % pre-allocation (for MATLAB) of entire vector "x"
80 x(:,1) = x0;  % PLUG IN the INITIAL CONDITIONS (position and velocity)
81 y = 0*t;              % actual measurement (pre-allocation)
82 xe = 0*x;              % estimated state (pre-allocation)
83 ye = y;               % estimated measurement (pre-allocation)
84 u = 0*t;  % no torque; you can modify, if desired...
85 P = zeros(3,3); % initialize covariance matrix
86 Psave = zeros(9,ndo);  % save matrix coefficients over time....
87 H = Cdm;
88
89 % Below, try playing with v and w, vs Q and R...
90 w = .001;  % noise on process (forces and torques)
91 v = 2*pi/(2^12); % noise on MEASUREMENT...
92 Q = 1*w.*[                    ];  % try changing, so not equal to actual "w"
93 R =     v;                % try changing, so not equal to actual "v"
94 for n=2:ndo
95     % ACTUAL PLANT DYNAMICS:  Use Adp, Bdp, Cdp and Ddp here.
96     x(:,n) = Adp*x(:,n-1) + Bdp*u(n-1) + [0;randn;0]*w;
97     y(:,n) = Cdp*x(:,n) + Ddp*u(n) + randn*v*0;
98     %y(:,n) = round(y(:,n)*(2^12/(2*pi))) * (2*pi/(2^12));
99
100    % INTERNAL MODEL OF PLANT: Use Adm, Bdm, Cdm and Ddm below.
101
102    % --- Predictor / Time Update steps:
103    % (1) Predictor: a priori estimated state (time update)
104    xe(:,n) = Adm*xe(:,n-1) + Bdm*u(n-1); % + Ld*(y(:,n-1)-ye(:,n-1));
105    % (2) Predictor: a priori covariance (time update)
106    P = Adm*P*Adm' + Q;
107    Q(1,1) = .99*Q(1,1);
108    Q(3,3) = .99*Q(3,3);
109
110    % --- Corrector / Measurement Update steps:
111    % (3) Corrector: Kalman gain (or, replace with a constant estimator!)
112    % Lkalman = P*Cdm'*inv(Cdm*P*Cdm' + R);  % Line below does this calc better,
```

} ← Can increase confidence in "bias" state, over time...

p2

```
113     Lkalman = (P*Cdm') / (Cdm*P*Cdm' + R); % better for speed AND accuracy
114
115     %%%%%%%%%%%%% BELOW IS A "TRICK", TO COMPARE ESTIMATORS: %%%%%%
116     % CHEAT below: substitute the ESTIMATOR of your choice...
117     %Luse = Ld;
118     Luse = Lkalman;
119
120     % (4) Corrector: a posteriori estimated state (measurement update)
121     xe(:,n) = xe(:,n) + Luse*(y(:,n)-Cdm*xe(:,n));
122     % (5) Corrector: a posteriori estimated covariance (meas. update)
123     P = P - Lkalman*Cdm*P;
124
125     % I also saved ye and Psave, just to look at them later...
126     ye(:,n) = Cdm*xe(:,n);
127     Psave(:,n) = P(:);  % keep a running log of P, over time...
128 end
129
130 % Then, plot results:
131 figure(1); clf
132 subplot(211)
133 plot(t,x(1,:)); hold on; grid on
134 plot(t,xe(1,:),'r--'); xlabel('Time (s)')
135 plot(t,y,'k.','MarkerSize',4)
136 ylabel('Angle (rad)')
137 legend('Actual measured state','Estimated measured state','Encoder output')
138 subplot(212)
139 plot(t,x(2,:)); hold on; grid on
140 plot(t,xe(2,:),'r--'); xlabel('Time (s)')
141 ylabel('Angular velocity (rad/s)')
142 legend('Actual state','Estimated state')
143
144 figure(2); clf; subplot(211)
145 plot(t,x(3,:)); hold on
146 plot(t,xe(3,:),'r--'); grid on
147 xlabel('Time (s)')
148 ylabel('Offset angle (rad)')
149 legend('Actual state','Estimated state')
150 subplot(212);
151 plot(t,x(1,:)+x(3,:),'b-'); hold on
152 plot(t,xe(1,:)+xe(3,:),'r--'); grid on
153 xlabel('Time (s)')
154 ylabel('Actual Angle (rad)')
155 legend('Actual state','Estimated state')
156
```

(faster than inv() in MATLAB...)

p3