# EN.530.603 Applied Optimal Control HW #5 Solutions

## Gowtham Garimella

February 23, 2014

#### 1. Code for question 1b

```
1 function f = int_test
2 % Kalman filtering of the double integrator with position measurements
4 % timing
5 dt = 1;
           % time-step
6 N = 30;
           % total time-steps
  T = N*dt; % final time
  % noise terms
10 S.qu = (3e-9)^2;
                     % external disturbance variance thetadot
                    % external disturbance variance bias
11 S.qv = (3e-6)^2;
12 S.qn = (1.5e-5)^2; % measurement noise variance
14 % PHI matrix
15 S.Phi = [1 - dt;
        0 1];
17
18 % G matrix
19 S.G = [dt;
         01;
22 % Q matrix
S.Q = [S.qv*dt + S.qu*(dt^3/3), -S.qu*(dt^2/2);
             -S.qu*(dt^2/2), S.qu*dt];
26 % R matrix
S.R = S.qn;
29 % H matrix
30 \text{ S.H} = [1, 0];
32 % initial estimate of mean and covariance
33 \times = [0; 1.7e-7];
^{34} P = diag([1e-2; 1e-12]);
35 thetadot = 0.02; % given trajectory for true state theta
36
```

```
37 xts = zeros(2, N+1); % true states
xs = zeros(2, N+1); % estimated states
39 Ps = zeros(2, 2, N+1); % estimated covariances
40 nm = zeros(N+1,1);
42 zs = zeros(1, N); % estimated state
44 pms = zeros(1, N); % measured position
45
46 \text{ xts}(:,1) = x;
47 \times (:,1) = x;
48 Ps(:,:,1) = P;
49 \text{ nm}(1) = \text{norm}(P);
51 for k=1:N
   xts(:,k+1) = xts(:,k) + [thetadot*dt; sqrt(S.qu)*randn];
53
     %generate u based on true state
    u = thetadot + xts(2,k+1) + sqrt(S.qv)*randn;
55
     [x,P] = kf_predict(x,P,u,S); % prediction
57
     z = xts(1,k+1) + sqrt(S.qn) * randn; % generate random measurement
59
     [x,P] = kf_correct(x,P,z,S); % correction
61
62
    % record result
63
xs(:,k+1) = x;
   Ps(:,:,k+1) = P;
66
    zs(:,k) = z;
nm(k+1) = norm(P);
68 end
70 plot(xts(1,:), 'x-', 'LineWidth',2)
71 hold on
72 plot(xs(1,:), 'gx-', 'LineWidth',2)
73 plot(dt*(2:N+1),zs(1,:), 'ro-', 'LineWidth',2)
74
75 xlabel('time(sec)');
76 ylabel('\theta(rad)');
77 legend('true', 'estimated', 'measured')
78
79 % 95% confidence intervals of the estimated position
80 plot(xs(1,:) + 1.96*reshape(sqrt(Ps(1,1,:)),N+1,1)', '-g')
si plot(xs(1,:) - 1.96*reshape(sqrt(Ps(1,1,:)),N+1,1)', '-q')
82
83 figure;
84 plot(dt*(2:N+1), nm(2:N+1),'-');
85 legend('norm covariance');
86 xlabel('time(sec)');
87
88 figure, hold on;
89 error = xs - xts;
90 plot(dt*(1:N+1), error(1,:),'r');
```

```
91 plot(dt*(1:N+1), error(2,:),'b');
  ylabel('rad or rad/s');
   xlabel('time(s)');
  legend('etheta', 'ebias');
95
   function [x,P] = kf_predict(x, P, u, S)
96
97
   x = S.Phi*x + S.G*u;
98
   P = S.Phi*P*S.Phi' + S.Q;
99
100
   function [x,P] = kf\_correct(x, P, z, S)
101
102
103
   K = P*S.H'*inv(S.H*P*S.H' + S.R);
  P = (eye(length(x)) - K*S.H)*P;
105 x = x + K*(z - S.H*x);
```

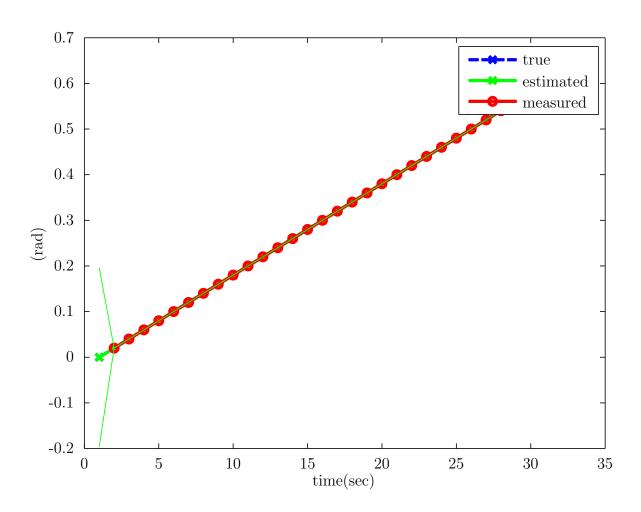


Figure 1: The result from kalman filter tracking of angle

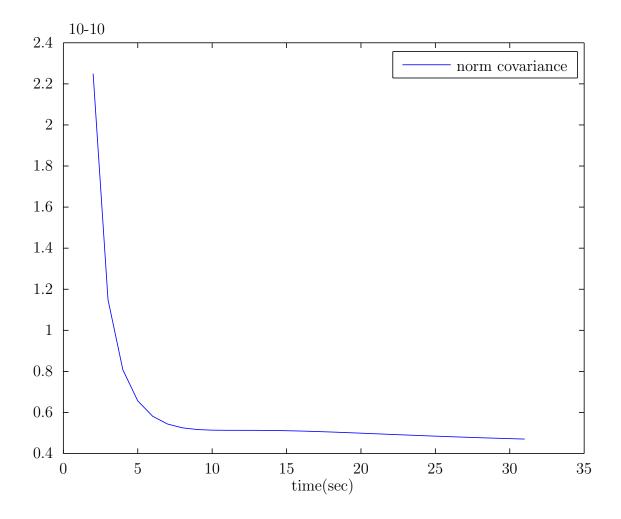


Figure 2: Norm of Covariance matrix (P) to show convergence

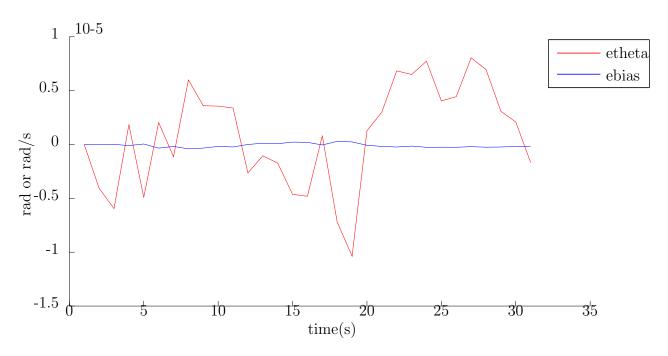


Figure 3: Error in angle and bias

### 2. Code for Question 3a

```
1 function f = uni_test1
  % Extended Kalman filtering of the unicycle with bearing and range ...
      measurements
  rng('default')
4
  S.f = @uni_f; % mobile-robot dynamics
  S.h = @br_h; % bearing—reange sensing
  S.n = 4;
                 % state dimension
  S.r = 2;
                % measurement dimension
  S.p0 = [0; 2]; % beacon position
11
12
  % timing
  dt = .1;
  N = 50;
  T = dt *N;
  S.dt = dt;
18
  % noise models
20 \text{ S.Q} = .1*dt*dt*diag([.1 .1 .1, .001]);
S.R = .01*diag([.5 1]); %??
  % initial mean and covariance
  xt = [0; 0; 0; 1]; % true state
P = 10 * .01 * diag([2 2 2 5]) % covariance
```

```
27 \times = xt + sqrt(P) * randn(S.n, 1); % initial estimate with added noise
28
29 xts = zeros(S.n, N+1); % true states
30 xs = zeros(S.n, N+1); % estimated states
31 Ps = zeros(S.n, S.n, N+1); % estimated covariances
33 zs = zeros(S.r, N); % measurements
34
35 \text{ xts}(:, 1) = xt;
36 \times (:, 1) = x;
37 Ps(:, :, 1) = P;
39 ds = zeros(S.n, N+1); % errors
40 ds(:,1) = x - xt;
41
42 for k=1:N,
   u = dt*[2; 1]; % known controls
43
   xts(:,k+1) = S.f(xts(:,k), u, S) + sqrt(S.Q)*randn(S.n,1); % true state
45
46
    [x,P] = ekf\_predict(x, P, u, S); % predict
47
   z = S.h(xts(:,k+1), S) + sqrt(S.R) * randn(S.r,1); % generate measurement
49
    z(1) = fangle(z(1));
50
    [x,P] = ekf\_correct(x, P, z, S); % correct
51
52
   xs(:,k+1) = x;
53
Ps(:,:,k+1) = P;
   zs(:,k) = z;
56
   ds(:,k+1) = x - xts(:,k+1); % actual estimate error
57
58 end
60 subplot (1, 2, 1)
62 plot(xts(1,:), xts(2,:), '-g', 'LineWidth',3)
63 hold on
64 plot(xs(1,:), xs(2,:), '-b', 'LineWidth', 3)
65 legend('true', 'estimated')
67 xlabel('x')
68 ylabel('y')
69 axis equal
70 axis xy
72 % beacon
73 plot(S.p0(1), S.p0(2), '*r');
74
75 for k=1:5:N
76 plotcov2(xs(1:2,k+1), Ps(1:2,1:2,k+1));
77 end
79 subplot (1, 2, 2)
80
```

```
81 plot(ds')
82
83 mean(sqrt(sum(ds.*ds, 1)))
84 xlabel('k')
85 ylabel('meters or radians')
86 legend('e_x','e_y','e_\theta','e_r')
88 figure, hold on;
89 plot(dt*(1:N+1),xs(4,:),'b');
90 plot(dt*(1:N+1),xts(4,:),'r');
91 xlabel('time(sec)');
92 ylabel('Radius(m)');
93 legend('Rest', 'Rtrue');
94
95 function [x, varargout] = uni_f(x, u, S)
96 % dynamical model of the unicycle modified
97 \% x = [x, y, theta, radius]
98 % u = [sigma(commanded wheel vel), rotational angl vel];
99 C = \cos(x(3));
100 s = sin(x(3));
101
102 \times = [x(1) + c*x(4)*u(1);
103
         x(2) + s*x(4)*u(1);
104
         x(3) + u(2);
        x(4)];
105
106
   if nargout > 1
107
108
     % F-matrix
     varargout{1} = [1, 0, -s*x(4)*u(1), c*u(1);
109
                       0, 1, c*x(4)*u(1), s*u(1);
110
                       0, 0, 1, 0;
111
                       0, 0, 0, 1];
112
113
   end
114
  function [y, varargout] = br_h(x, S)
116
117
p = x(1:2);
px = p(1);
py = p(2);
122 d = S.p0 - p;
123 r = norm(d);
124
125 th = fangle(atan2(d(2), d(1)) - x(3));
126
127 y = [th; r];
128
  if nargout > 1
129
   % H—matrix
130
     varargout\{1\} = [d(2)/r^2, -d(1)/r^2, -1, 0;
131
                      -d'/r, 0, 0];
132
133
   end
134
```

```
135
   function [x,P] = ekf_predict(x, P, u, S)
136
137
   [x, F] = S.f(x, u, S);
138
   P = F * P * F' + S.Q;
140
141
    function [x,P] = ekf_correct(x, P, z, S)
142
143
    [y, H] = S.h(x, S);
144
145
   K = P*H'*inv(H*P*H' + S.R);
146
147
   P = (eye(S.n) - K*H)*P;
148
   x = x + K*fangle(z-y);
149
150
151
   function a = fangle(a)
   % make sure angle is between -pi and pi
153
   if a < -pi
      a = a + 2*pi
155
   else
156
      if a > pi
157
158
        a = a - 2*pi
      end
159
   end
160
```

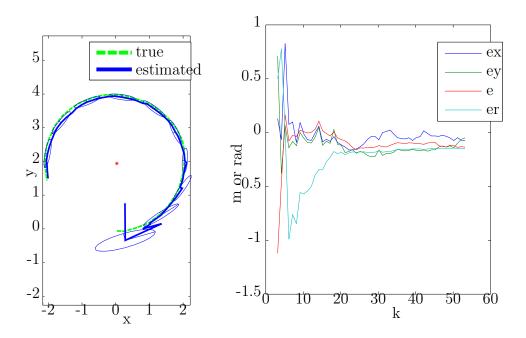


Figure 4: Estimated trajectory and true trajectory for EKF. (b) Errors of all the four states oscillate around 0

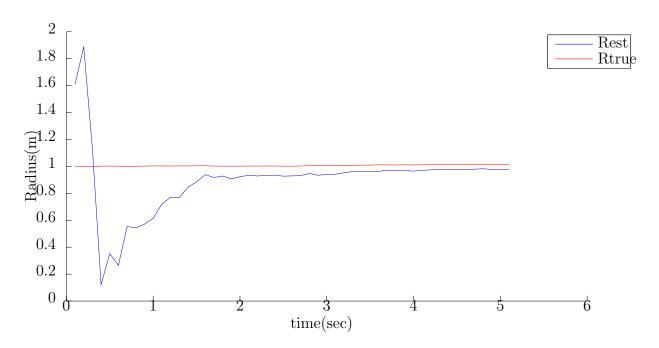


Figure 5: The radius of the wheel converges to it's true value

#### 3. Code for 3b

```
1 function f = uni_test2
  % Extended Kalman filtering of the unicycle with bearing and range ...
      measurements
  rng('default')
  S.f = @uni_f; % mobile-robot dynamics
  S.h = @br_h; % bearing—reange sensing
  S.n = 4;
                % state dimension
  S.r = 1;
               % measurement dimension
  S.p0 = [0; 2];
                   % beacon position
11
  % timing
13
  dt = .1;
  N = 80;
  T = dt *N;
  S.dt = dt;
17
18
  % noise models
S.Q = .1*dt*dt*diag([.1 .1 .1, .001]);
S.R = .01*diag([.5]);%no distance
  % initial mean and covariance
  xt = [0; 0; 0; 1]; % true state
25
```

```
P = 5*.01*diag([1 1 1 4]) % covariance
x = xt + sqrt(P) * randn(S.n, 1); % initial estimate with added noise
29 xts = zeros(S.n, N+1); % true states
30 xs = zeros(S.n, N+1); % estimated states
31 Ps = zeros(S.n, S.n, N+1); % estimated covariances
33 zs = zeros(S.r, N); % measurements
34
35 \text{ xts}(:, 1) = xt;
36 \times (:, 1) = x;
37 Ps(:, :, 1) = P;
39 ds = zeros(S.n, N+1); % errors
40 ds(:,1) = x - xt;
41
42 for k=1:N,
    u = dt * [2; 1]; % known controls
44
    xts(:,k+1) = S.f(xts(:,k), u, S) + sqrt(S.Q)*randn(S.n,1); % true state
46
     [x,P] = ekf\_predict(x, P, u, S); % predict
     48
     z = fangle(S.h(xts(:,k+1), S) + sqrt(S.R)*randn(S.r,1)); % generate ...
49
        measurement
50
     [x,P] = ekf\_correct(x, P, z, S); % correct
51
52
   xs(:,k+1) = x;
53
    Ps(:,:,k+1) = P;
54
55
   zs(:,k) = z;
56
    ds(:,k+1) = x - xts(:,k+1); % actual estimate error
58 end
60 subplot (1, 2, 1)
62 plot(xts(1,:), xts(2,:), '-g', 'LineWidth',3)
63 hold on
64 plot(xs(1,:), xs(2,:), '-b', 'LineWidth', 3)
65 legend('true', 'estimated')
66
67 xlabel('x')
68 ylabel('y')
69 axis equal
70 axis xy
71
72 % beacon
73 plot(S.p0(1), S.p0(2), '*r');
75 for k=1:5:N
   plotcov2(xs(1:2,k+1), Ps(1:2,1:2,k+1));
76
77 end
78
```

```
79 subplot (1, 2, 2)
80
   plot(ds')
81
82
83 mean(sqrt(sum(ds.*ds, 1)))
84 xlabel('k')
85 ylabel('meters or radians')
86 legend('e_x','e_y','e_\theta','e_r')
88 figure, hold on;
89 plot(dt*(1:N+1),xs(4,:),'b');
90 plot(dt*(1:N+1),xts(4,:),'r');
91 xlabel('time(sec)');
92 ylabel('Radius(m)');
93 legend('Rest', 'Rtrue');
94
95 function [x, varargout] = uni_f(x, u, S)
96 % dynamical model of the unicycle modified
97 \% x = [x, y, theta, radius]
98 % u = [sigma(commanded wheel vel), rotational angl vel];
99 \ C = \cos(x(3));
100 s = sin(x(3));
101
   x = [x(1) + c*x(4)*u(1);
        x(2) + s*x(4)*u(1);
103
         x(3) + u(2);
104
         x(4)];
105
106
   if nargout > 1
107
     % F-matrix
108
     varargout{1} = [1, 0, -s*x(4)*u(1), c*u(1);
109
                       0, 1, c*x(4)*u(1), s*u(1);
110
                       0, 0, 1, 0;
111
112
                       0, 0, 0, 1];
113
   end
114
  function [y, varargout] = br_h(x, S)
116
117
p = x(1:2);
   px = p(1);
119
_{120} %py = p(2);
121
122 d = S.p0 - p;
123 r = norm(d);
124
   th = fangle(atan2(d(2), d(1)) - x(3));
126
_{127} y = th;
128
129 if nargout > 1
130
    % H—matrix
131
     varargout\{1\} = [d(2)/r^2, -d(1)/r^2, -1, 0];
132 end
```

```
133
134
    function [x,P] = ekf_predict(x, P, u, S)
135
136
    [x, F] = S.f(x, u, S);
   P = F * P * F' + S.O;
138
139
140
    function [x,P] = ekf_correct(x, P, z, S)
141
142
    [y, H] = S.h(x, S);
143
144
145
   K = P*H'*inv(H*P*H' + S.R);
   P = (eye(S.n) - K*H)*P;
146
147
   x = x + K*fangle(z-y);
148
149
150
   function a = fangle(a)
151
   % make sure angle is between -pi and pi
   if a < -pi
153
      a = a + 2*pi
155
   else
      if a > pi
156
        a = a - 2*pi
157
      end
158
   end
159
```

**Remarks:** From the results in Fig[6, 7] the sensor is not able to reconstruct the whole state. The radius of the wheel goes converges to around 0.8 and the errors in position x and y look like sine waves without any drop in amplitude. This can be easily understood based on the nature of the sensor data. Since we are only getting angle information and no distance information, in the Fig[6], the estimated value converges to a circle of a smaller radius. Since both have same phase the angle with respect to the beacon will be the same but the distance from the beacon will be different. Hence it is not possible to reconstruct the full state information from only angle measurement.

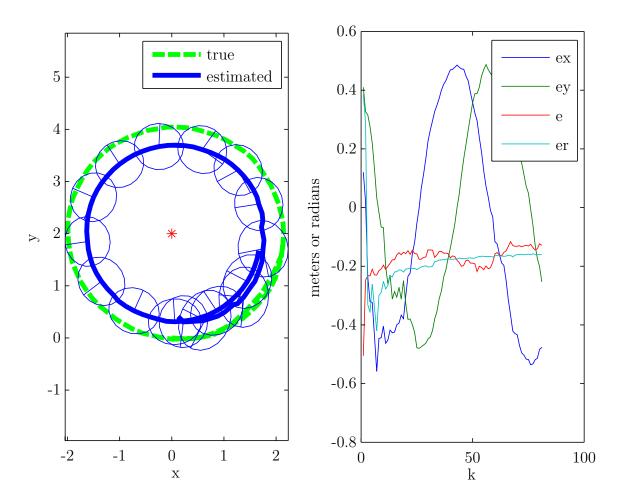


Figure 6: The result of tracking for EKF with only angle measurements

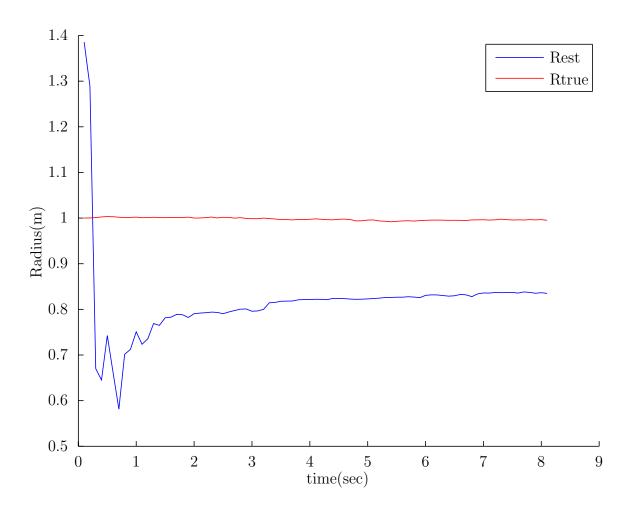


Figure 7: The radius does not converge to the true value and has an offset