EN.530.603 Applied Optimal Control
HW #5 Solutions

Gowtham Garimella

February 23, 2014

1. Code for question 1b

```
function f = int_test
% Kalman filtering of the double integrator with position measurements

% timing
dt = 1;    % time-step
N = 30;    % total time-steps
T = N*dt;  % final time

% noise terms
S.qu = (3e-9)^2;     % external disturbance variance thetadot
S.qv = (3e-6)^2;     % external disturbance variance bias
S.qn = (1.5e-5)^2;     % measurement noise variance

% PHI matrix
S.Phi = [1 -dt;
       0 1];

% G matrix
S.G = [dt;
       0];

% 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];

% R matrix
S.R = S.qn;

% H matrix
S.H = [1, 0];

% initial estimate of mean and covariance
x = [0; 1.7e-7];
P = diag([1e-2; 1e-12]);
thetadot = 0.02; % given trajectory for true state theta

```

```
xts = zeros(2, N+1); % true states
xs = zeros(2, N+1);  % estimated states
Ps = zeros(2, 2, N+1); % estimated covariances
nm = zeros(N+1,1);

zs = zeros(1, N);  % estimated state

pms = zeros(1, N); % measured position

xts(:,1) = x;
xs(:,1) = x;
Ps(:,:,1) = P;
nm(1) = norm(P);

for k=1:N
  xts(:,k+1) = xts(:,k) + [thetadot*dt;sqrt(S.qu)*randn];

  %generate u based on true state
  u = thetadot + xts(2,k+1) + sqrt(S.qv)*randn;

  [x,P] = kf_predict(x,P,u,S);  % prediction

  z = xts(1,k+1) + sqrt(S.qn)*randn;   % generate random measurement

  [x,P] = kf_correct(x,P,z,S);  % correction

  % record result
  xs(:,k+1) = x;
  Ps(:,:,k+1) = P;
  zs(:,k) = z;
  nm(k+1) = norm(P);
end

plot(xts(1,:), 'x--', 'LineWidth',2)
hold on
plot(xs(1,:), 'gx-', 'LineWidth',2)
plot(dt*(2:N+1),zs(1,:), 'ro-', 'LineWidth',2)

xlabel('time(sec)');
ylabel('\theta(rad)');
legend('true', 'estimated','measured')

% 95% confidence intervals of the estimated position
plot(xs(1,:) + 1.96*reshape(sqrt(Ps(1,1,:)),N+1,1)', '-g')
plot(xs(1,:) - 1.96*reshape(sqrt(Ps(1,1,:)),N+1,1)', '-g')

figure;
plot(dt*(2:N+1),nm(2:N+1),'-');
legend('norm covariance');
xlabel('time(sec)');

figure, hold on;
error = xs - xts;
plot(dt*(1:N+1), error(1,:),'r');
```

```
plot(dt*(1:N+1), error(2,:),'b');
ylabel('rad or rad/s');
xlabel('time(s)');
legend('etheta','ebias');

function [x,P] = kf_predict(x, P, u, S)

x = S.Phi*x + S.G*u;
P = S.Phi*P*S.Phi' + S.Q;

function [x,P] = kf_correct(x, P, z, S)

K = P*S.H'*inv(S.H*P*S.H' + S.R);
P = (eye(length(x)) - K*S.H)*P;
x = x + K*(z - S.H*x);
```

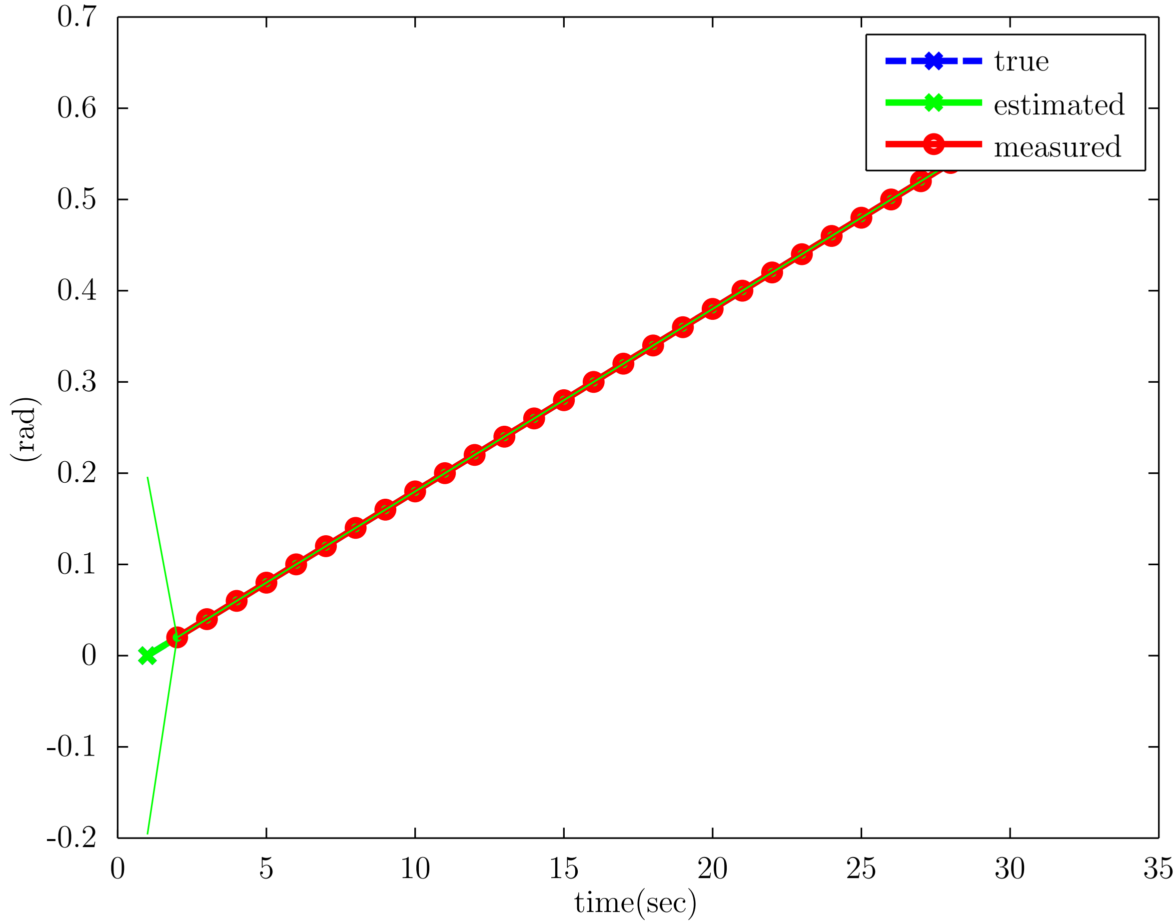

Figure 1: The result from kalman filter tracking of angle

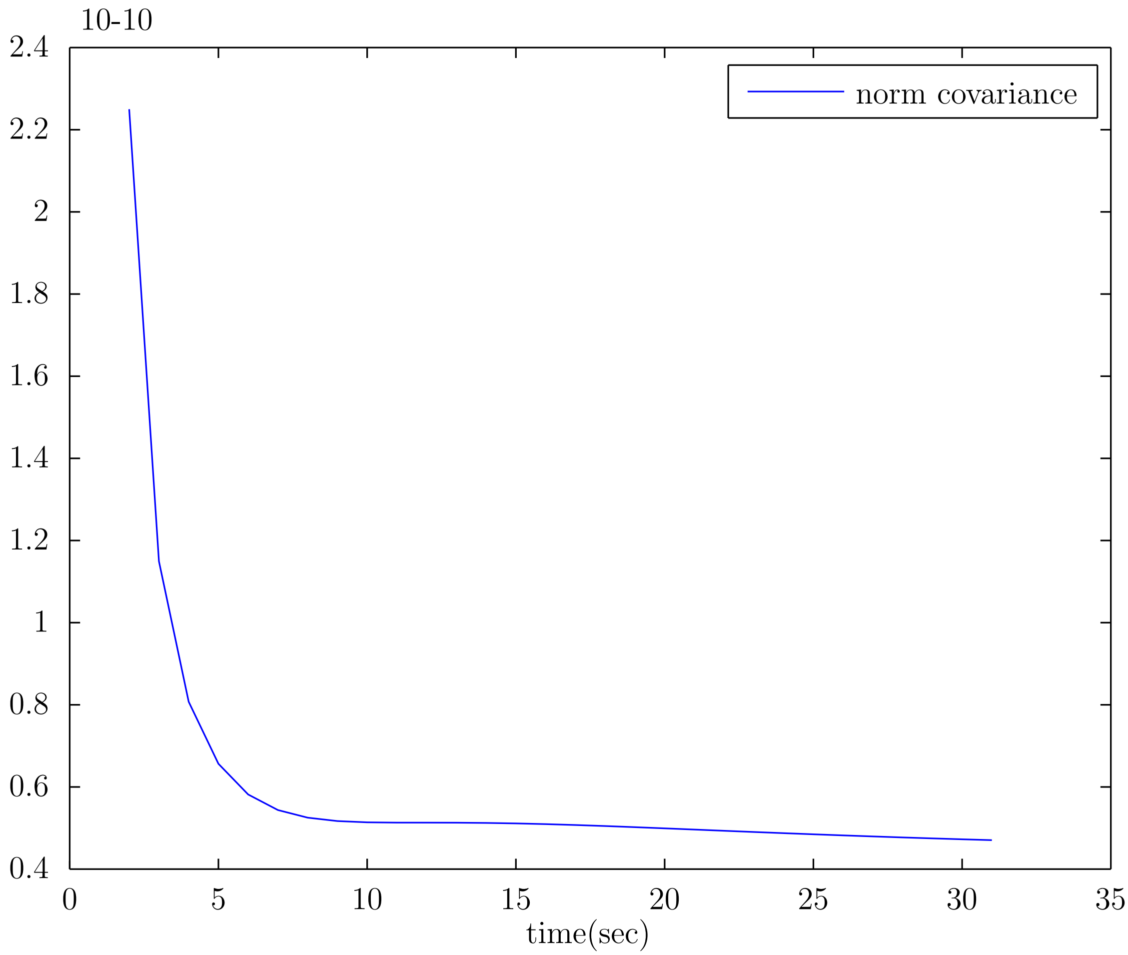

Figure 2: Norm of Covariance matrix (P) to show convergence

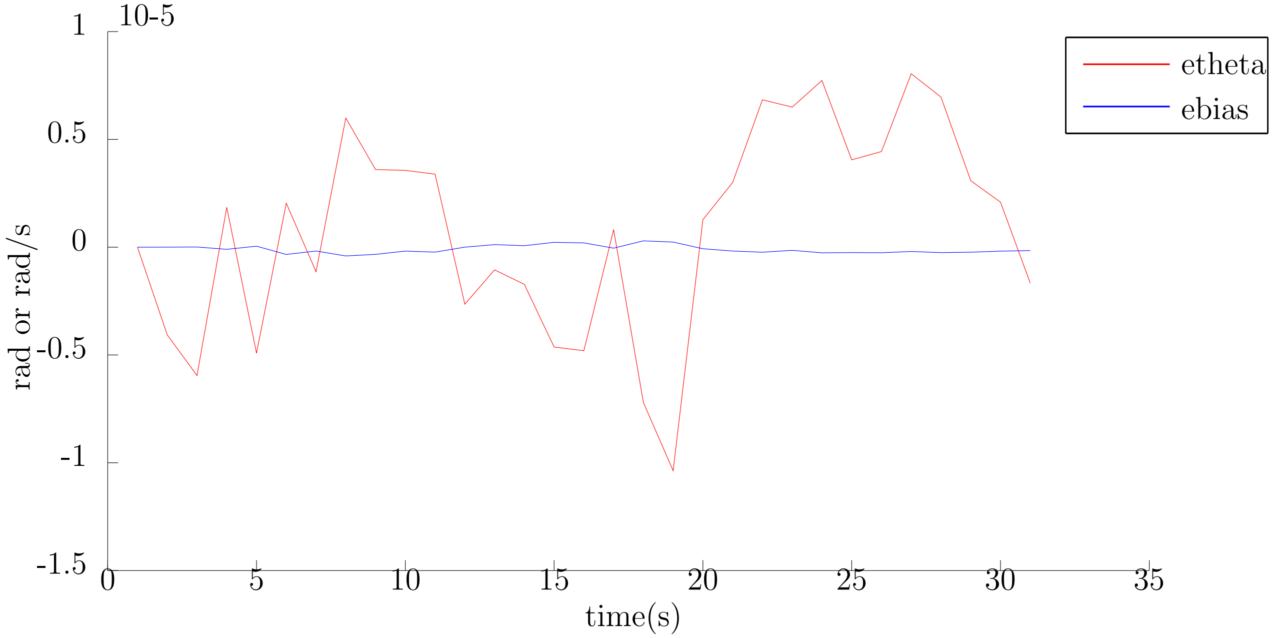

Figure 3: Error in angle and bias

2. Code for Question 3a

```
function f = uni_test1
% 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 = 2;      % measurement dimension

S.p0 = [0; 2];    % beacon position

% timing
dt = .1;
N = 50;
T = dt*N;
S.dt = dt;

% noise models
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
```

```
x = xt + sqrt(P)*randn(S.n, 1); % initial estimate with added noise

xts = zeros(S.n, N+1); % true states
xs = zeros(S.n, N+1);  % estimated states
Ps = zeros(S.n, S.n, N+1); % estimated covariances

zs = zeros(S.r, N);  % measurements

xts(:, 1) = xt;
xs(:, 1) = x;
Ps(:, :, 1) = P;

ds = zeros(S.n, N+1);  % errors
ds(:,1) = x - xt;

for k=1:N,
  u = dt*[2; 1];  % known controls

  xts(:,k+1) = S.f(xts(:,k), u, S) + sqrt(S.Q)*randn(S.n,1);  % true state

  [x,P] = ekf_predict(x, P, u, S);  % predict

  z = S.h(xts(:,k+1), S) + sqrt(S.R)*randn(S.r,1);  % generate measurement
  z(1) = fangle(z(1));
  [x,P] = ekf_correct(x, P, z, S);  % correct

  xs(:,k+1) = x;
  Ps(:,:,k+1) = P;

  zs(:,k) = z;
  ds(:,k+1) = x - xts(:,k+1);  % actual estimate error
end

subplot(1, 2, 1)

plot(xts(1,:), xts(2,:), '--g','LineWidth',3)
hold on
plot(xs(1,:), xs(2,:), '-b','LineWidth',3)
legend('true', 'estimated')

xlabel('x')
ylabel('y')
axis equal
axis xy

% beacon
plot(S.p0(1), S.p0(2), '*r');

for k=1:5:N
  plotcov2(xs(1:2,k+1), Ps(1:2,1:2,k+1));
end

subplot(1,2,2)

```

```
plot(ds')

mean(sqrt(sum(ds.*ds, 1)))
xlabel('k')
ylabel('meters or radians')
legend('e_x','e_y','e_\theta','e_r')

figure, hold on;
plot(dt*(1:N+1),xs(4,:),'b');
plot(dt*(1:N+1),xts(4,:),'r');
xlabel('time(sec)');
ylabel('Radius(m)');
legend('Rest','Rtrue');

function [x, varargout] = uni_f(x, u, S)
% dynamical model of the unicycle modified
% x = [x,y,theta,radius]
% u = [sigma(commanded wheel vel), rotational angl vel];
c = cos(x(3));
s = sin(x(3));

x = [x(1) + c*x(4)*u(1);
     x(2) + s*x(4)*u(1);
     x(3) + u(2);
     x(4)];

if nargout > 1
  % F-matrix
  varargout{1} = [1, 0, -s*x(4)*u(1), c*u(1);
                  0, 1, c*x(4)*u(1), s*u(1);
                  0, 0, 1, 0;
                  0, 0, 0, 1];
end


function [y, varargout] = br_h(x, S)

p = x(1:2);
px = p(1);
py = p(2);

d = S.p0 - p;
r = norm(d);

th = fangle(atan2(d(2), d(1)) - x(3));

y = [th; r];

if nargout > 1
  % H-matrix
  varargout{1} = [d(2)/r^2, -d(1)/r^2, -1, 0;
                  -d'/r, 0, 0];
end

```

```
function [x,P] = ekf_predict(x, P, u, S)

[x, F] = S.f(x, u, S);
P = F*P*F' + S.Q;


function [x,P] = ekf_correct(x, P, z, S)

[y, H] = S.h(x, S);

K = P*H'*inv(H*P*H' + S.R);
P = (eye(S.n) - K*H)*P;

x = x + K*fangle(z-y);


function a = fangle(a)
% make sure angle is between -pi and pi
if a < -pi
  a = a + 2*pi
else
  if a > pi
    a = a - 2*pi
  end
end
```

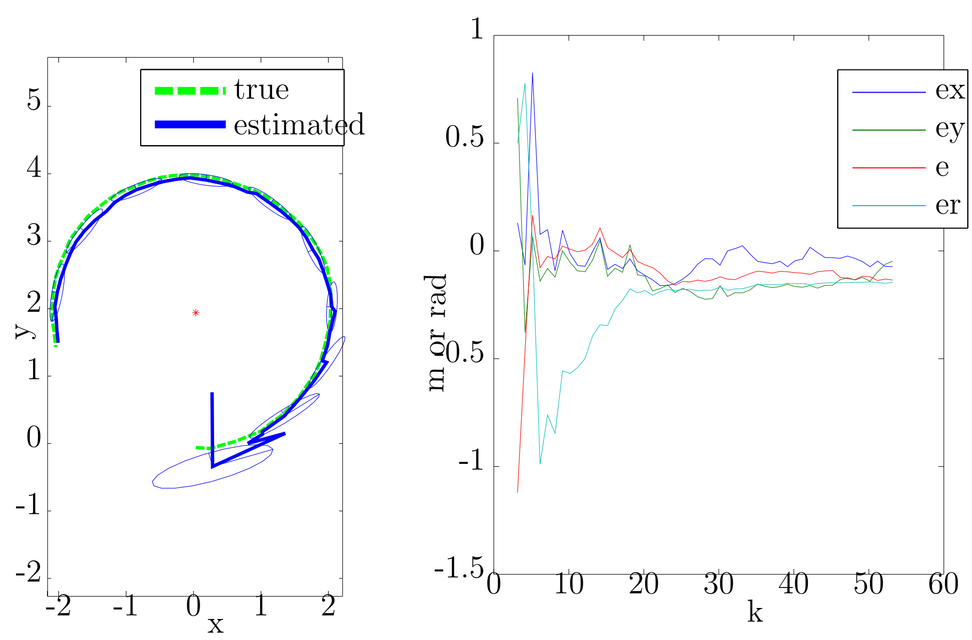

Figure 4: Estimated trajectory and true trajectory for EKF. (b) Errors of all the four states oscillate around 0

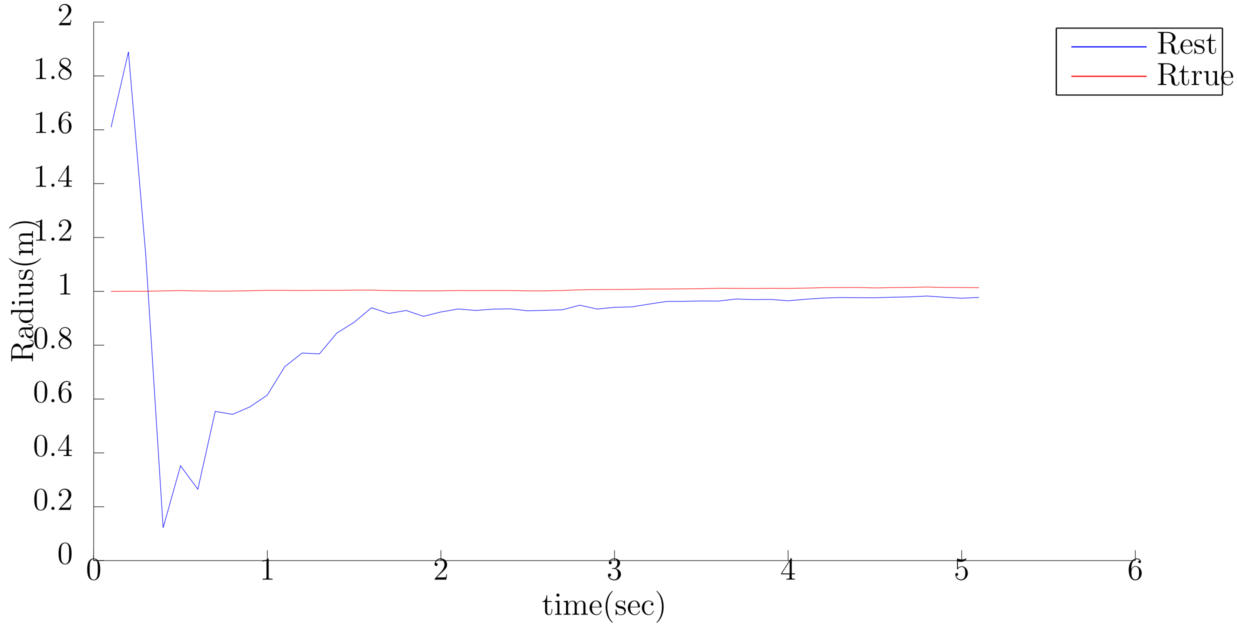

Figure 5: The radius of the wheel converges to it's true value

3. Code for 3b

```
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

% timing
dt = .1;
N = 80;
T = dt*N;
S.dt = dt;

% 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

```

```
P = 5*.01*diag([1 1 1 4]) % covariance
x = xt + sqrt(P)*randn(S.n, 1); % initial estimate with added noise

xts = zeros(S.n, N+1); % true states
xs = zeros(S.n, N+1);  % estimated states
Ps = zeros(S.n, S.n, N+1); % estimated covariances

zs = zeros(S.r, N);  % measurements

xts(:, 1) = xt;
xs(:, 1) = x;
Ps(:, :, 1) = P;

ds = zeros(S.n, N+1);  % errors
ds(:,1) = x - xt;

for k=1:N,
  u = dt*[2; 1];  % known controls

  xts(:,k+1) = S.f(xts(:,k), u, S) + sqrt(S.Q)*randn(S.n,1);  % true state

  [x,P] = ekf_predict(x, P, u, S);  % predict
  %%%%%%%%%%%%%%%%%%%%%%%%
  z = fangle(S.h(xts(:,k+1), S) + sqrt(S.R)*randn(S.r,1));  % generate ...
      measurement

  [x,P] = ekf_correct(x, P, z, S);  % correct

  xs(:,k+1) = x;
  Ps(:,:,k+1) = P;

  zs(:,k) = z;
  ds(:,k+1) = x - xts(:,k+1);  % actual estimate error
end

subplot(1, 2, 1)

plot(xts(1,:), xts(2,:), '--g','LineWidth',3)
hold on
plot(xs(1,:), xs(2,:), '-b','LineWidth',3)
legend('true', 'estimated')

xlabel('x')
ylabel('y')
axis equal
axis xy

% beacon
plot(S.p0(1), S.p0(2), '*r');

for k=1:5:N
  plotcov2(xs(1:2,k+1), Ps(1:2,1:2,k+1));
end

```

```
subplot(1,2,2)

plot(ds')

mean(sqrt(sum(ds.*ds, 1)))
xlabel('k')
ylabel('meters or radians')
legend('e_x','e_y','e_\theta','e_r')

figure, hold on;
plot(dt*(1:N+1),xs(4,:),'b');
plot(dt*(1:N+1),xts(4,:),'r');
xlabel('time(sec)');
ylabel('Radius(m)');
legend('Rest','Rtrue');

function [x, varargout] = uni_f(x, u, S)
% dynamical model of the unicycle modified
% x = [x,y,theta,radius]
% u = [sigma(commanded wheel vel), rotational angl vel];
c = cos(x(3));
s = sin(x(3));

x = [x(1) + c*x(4)*u(1);
     x(2) + s*x(4)*u(1);
     x(3) + u(2);
     x(4)];

if nargout > 1
  % F-matrix
  varargout{1} = [1, 0, -s*x(4)*u(1), c*u(1);
                  0, 1, c*x(4)*u(1), s*u(1);
                  0, 0, 1, 0;
                  0, 0, 0, 1];
end


function [y, varargout] = br_h(x, S)

p = x(1:2);
%px = p(1);
%py = p(2);

d = S.p0 - p;
r = norm(d);

th = fangle(atan2(d(2), d(1)) - x(3));

y = th;

if nargout > 1
  % H-matrix
  varargout{1} = [d(2)/r^2, -d(1)/r^2, -1, 0];
end
```

```
function [x,P] = ekf_predict(x, P, u, S)

[x, F] = S.f(x, u, S);
P = F*P*F' + S.Q;


function [x,P] = ekf_correct(x, P, z, S)

[y, H] = S.h(x, S);

K = P*H'*inv(H*P*H' + S.R);
P = (eye(S.n) - K*H)*P;

x = x + K*fangle(z-y);


function a = fangle(a)
% make sure angle is between -pi and pi
if a < -pi
  a = a + 2*pi
else
  if a > pi
    a = a - 2*pi
  end
end
```

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.

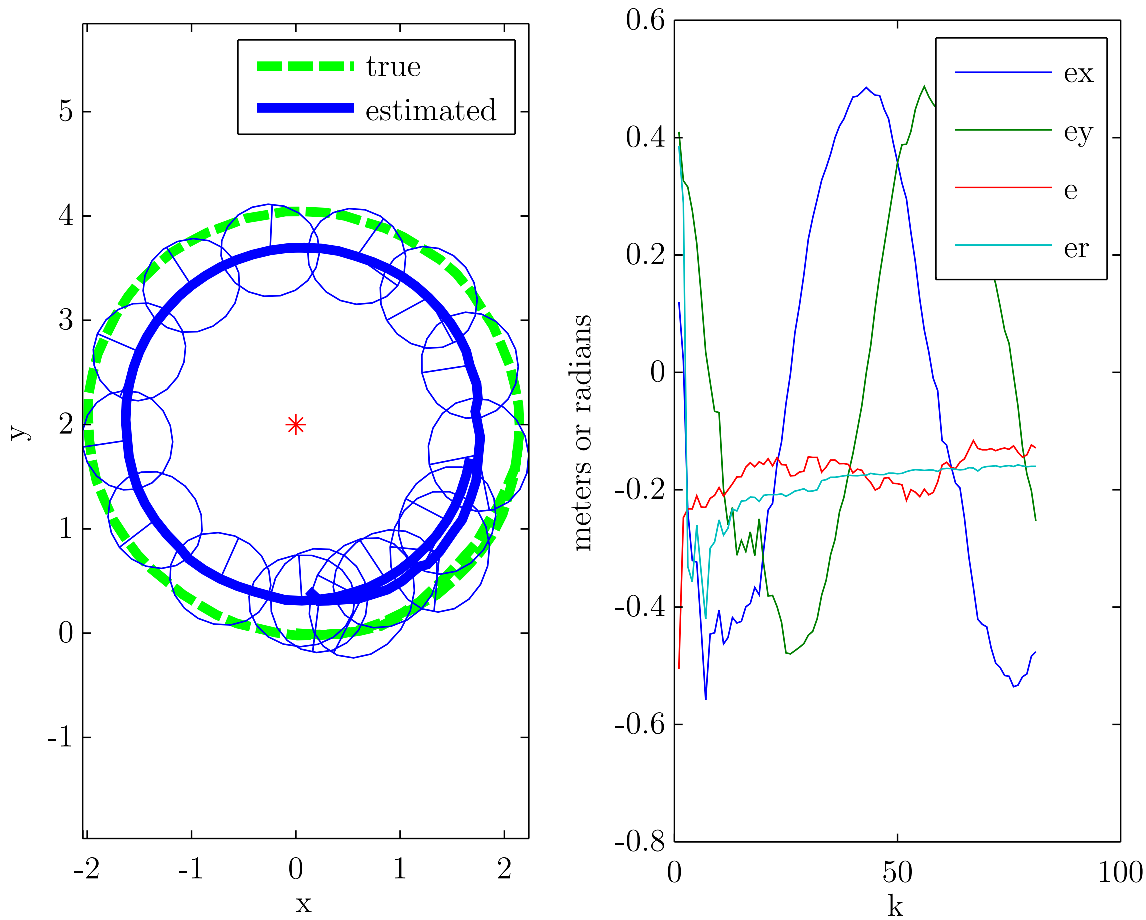

Figure 6: The result of tracking for EKF with only angle measurements

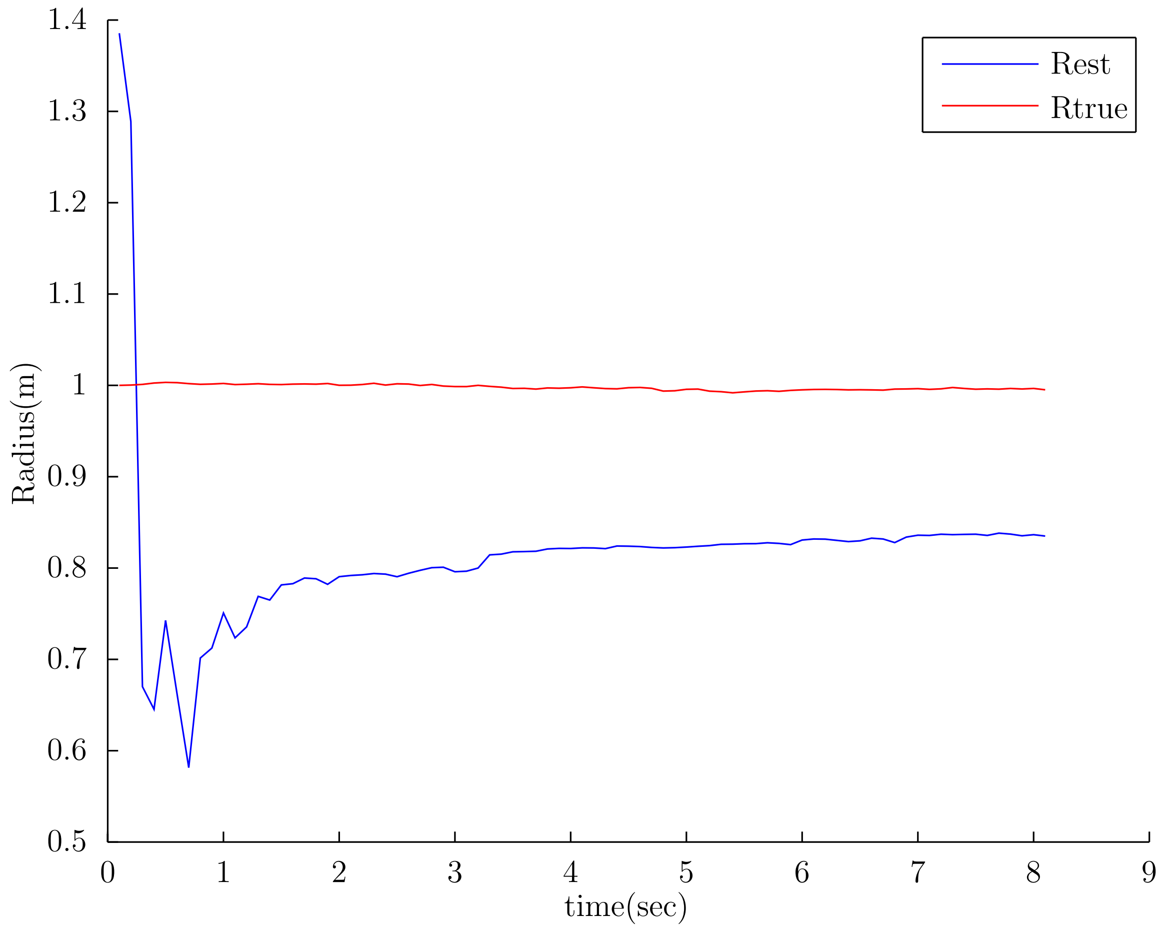

Figure 7: The radius does not converge to the true value and has an offset