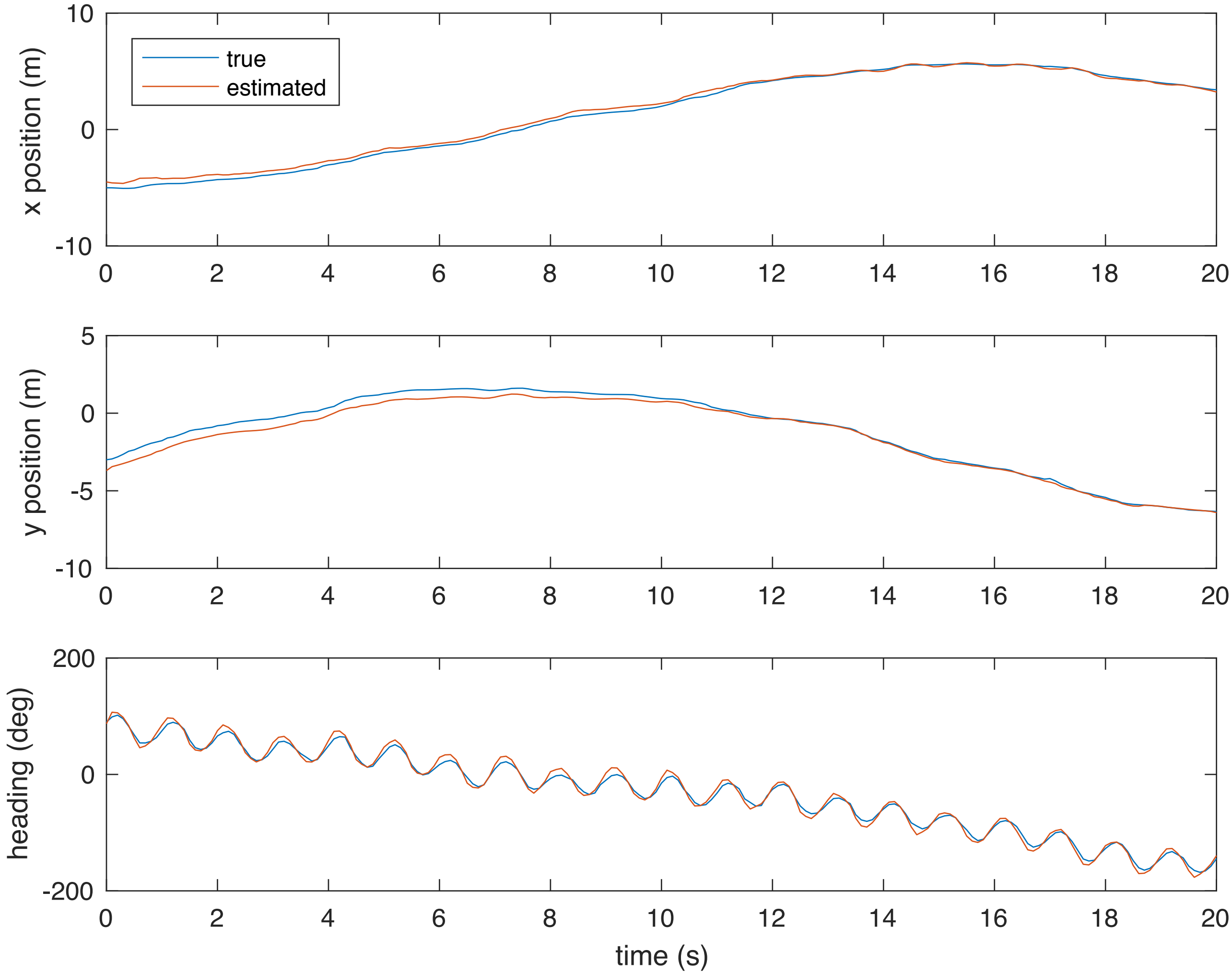
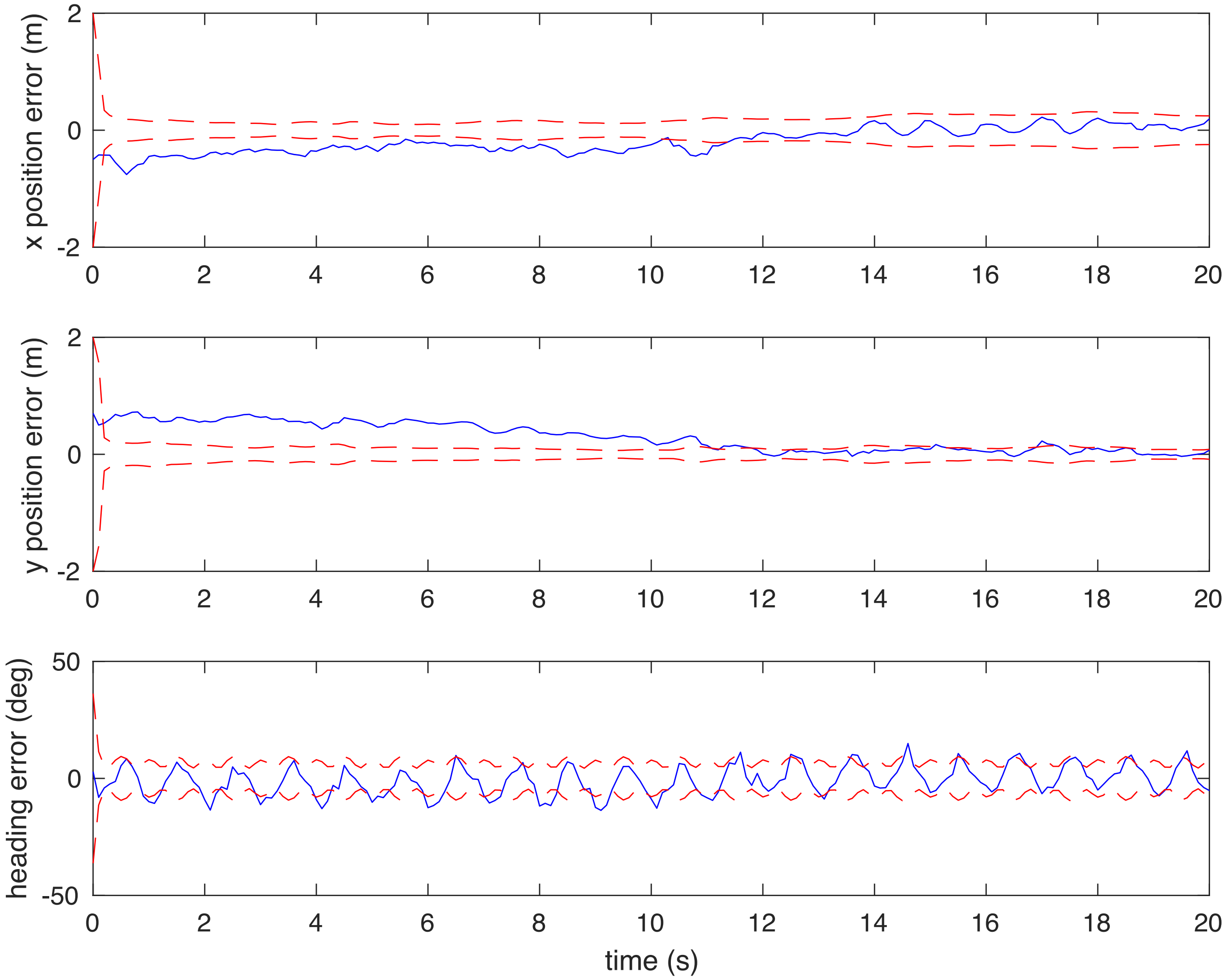
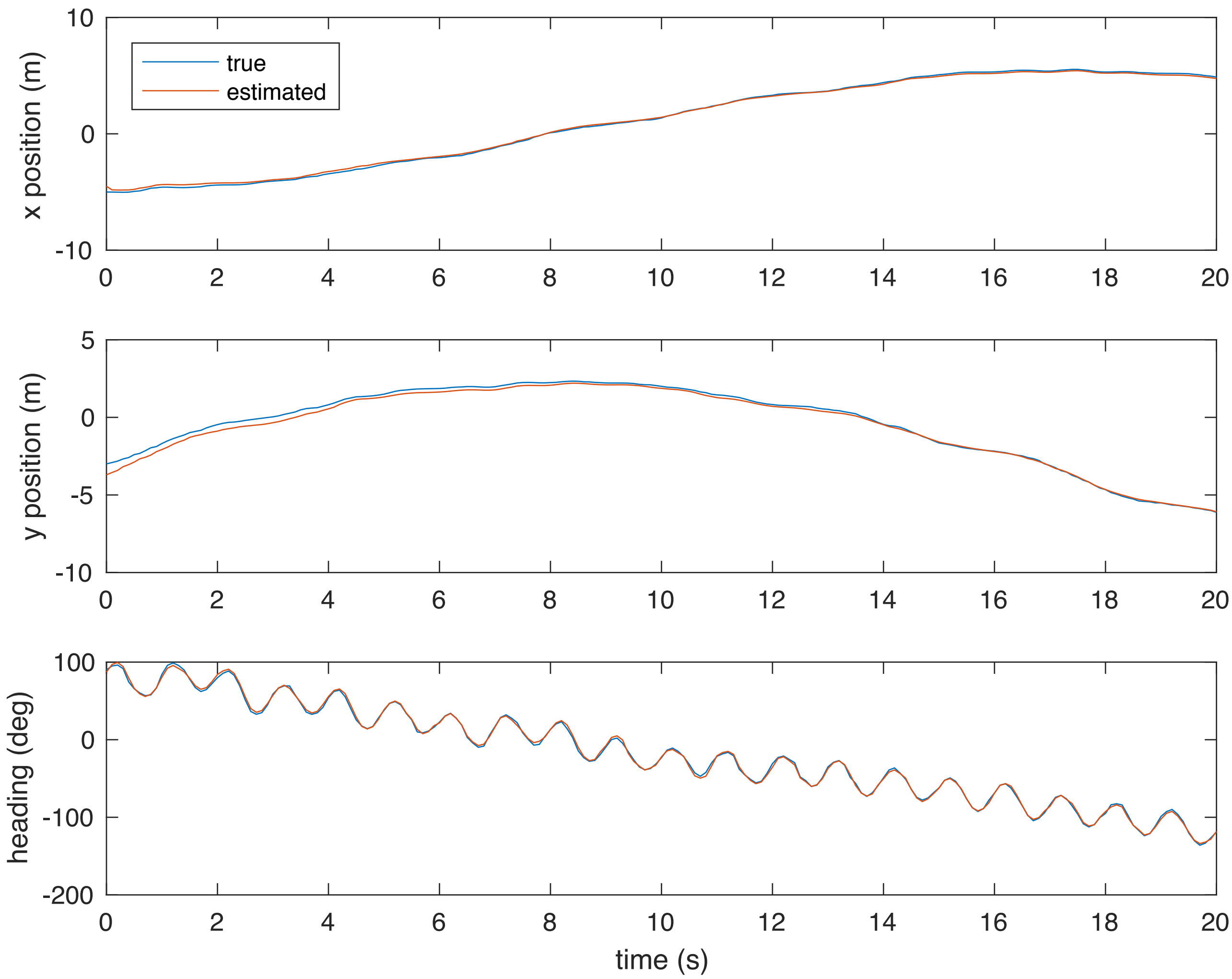
Single Landmark



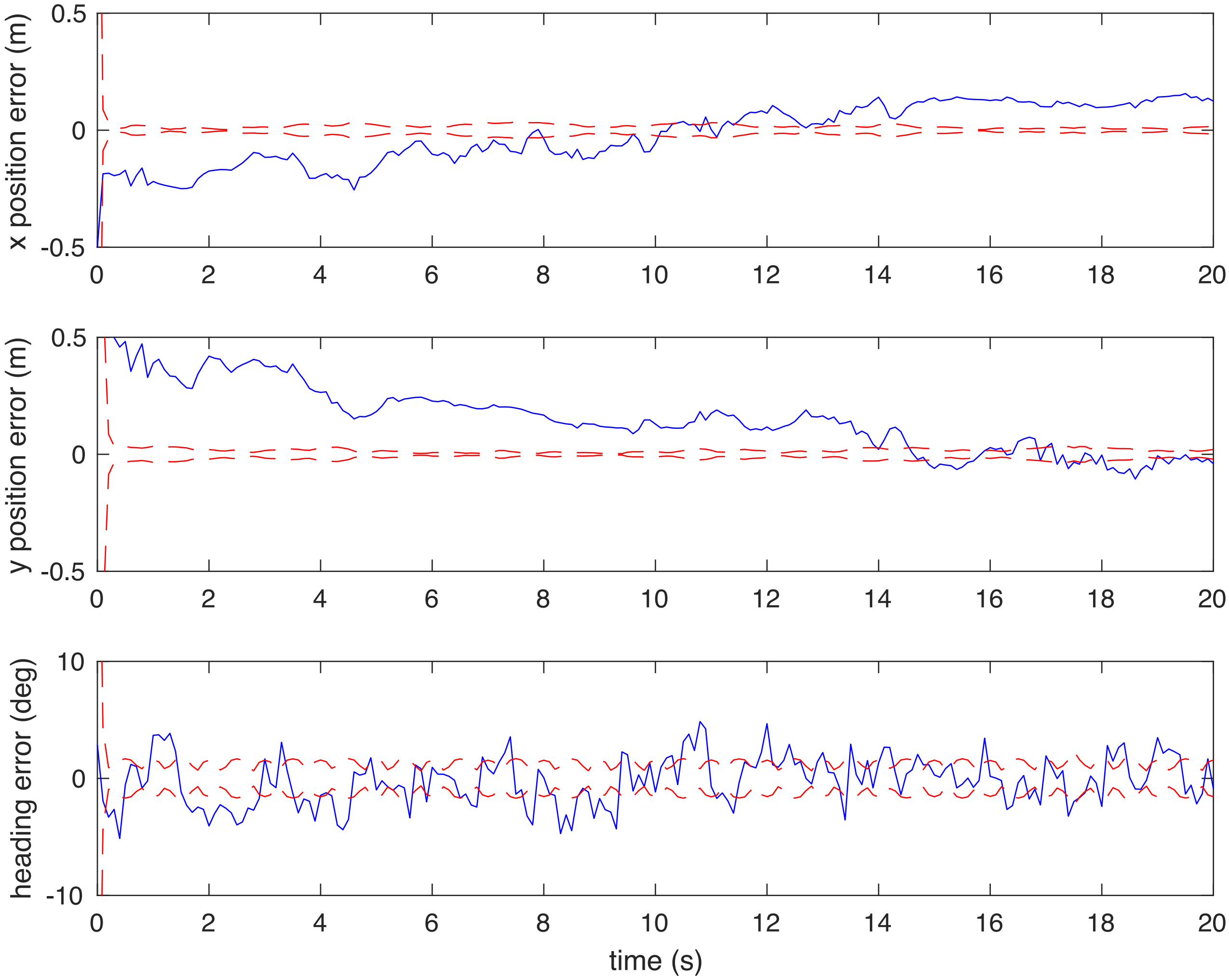
Single Landmark: Covariance bounds do not seem accurate, but error doesn't appear Gaussian either.



Three Landmarks



Three Landmarks: Covariance bounds do not seem accurate, but error doesn't appear Gaussian either. Errors are significantly smaller than one-landmark case (note change of scale on axes).



```
% Ground robot localization from Probabilistic Robotics, Thrun et al.,
% Table 7.4
%
% This version implements multiple landmarks with the UKF, but only allows
% one landmark measurement update per time update
%
% State variables are (x,y,th)
% State estimates are (mu_x,mu_y,mu_th)
% Covariance of estimation error is Sig

clear all;

dt = 0.1;
tfinal = 20;
t = 0:dt:tfinal;

N = length(t);

% Initial conditions
x0 = -5;
y0 = -3;
th0 = pi/2;

% Landmark (feature) locations
mx = [6 -7 6];      % x-coordinate of landmarks
my = [4 8 -4];      % y-coordinate of landmarks
m = [mx; my];
MM = 3;             % number of landmarks

% Motion input plus noise model
v_c = 1 + 0.5*sin(2*pi*0.3*t);
om_c = -0.2 + 2*cos(2*pi*1*t);
u_c = [v_c; om_c];
alph1 = 0.1;
alph2 = 0.01;
alph3 = 0.01;
alph4 = 0.1;

v = v_c + sqrt(alph1*v_c.^2+alph2*om_c.^2).*randn(1,N);
om = om_c + sqrt(alph3*v_c.^2+alph4*om_c.^2).*randn(1,N);

x(1) = x0;
y(1) = y0;
th(1) = th0;

% Draw robot at time step 1
drawRobot(x(1),y(1),th(1),m,t(1));

for i = 2:N,
   x(i) = x(i-1) + (-v(i)/om(i)*sin(th(i-1)) + v(i)/om(i)*sin(th(i-1)+om(i)*dt));
   y(i) = y(i-1) + (v(i)/om(i)*cos(th(i-1)) - v(i)/om(i)*cos(th(i-1)+om(i)*dt));
   th(i) = th(i-1) + om(i)*dt;
   drawRobot(x(i),y(i),th(i),m,t(i));
   % pause(0.05);
end
X = [x; y; th];     % matrix of true state vectors at all times

% Localize robot using UKF from Table 7.4
% UKF parameters
kap = 4;
alph = 0.4;
beta = 2;
n = 7;
lam = alph^2*(n+kap)-n;
gam = sqrt(n+lam);

% Sigma point weights
wm(1) = lam/(n+lam);
wc(1) = wm(1) + (1-alph^2+beta);
wm(2:15) = 1/(2*(n+lam));
wc(2:15) = wm(2:15);

% Sigma point scaling to improve robustness
ALF = 2.8;
wm(1) = wm(1)/ALF^2 + (1-1/ALF^2);
wm(2:15) = wm(2:15)/ALF^2;

wc = wm;
```

```
wc(1) = wc(1)/ALF^2 + (1-1/ALF^2);

% Measurement noise level
sig_r = 0.1;
sig_ph = 0.05;
sig = [sig_r; sig_ph];

% Initial conditions of state estimates at time zero
mu_x = x0+0.5;
mu_y = y0-0.7;
mu_th = th0-0.05;
mu = [mu_x; mu_y; mu_th];

Sig = diag([1,1,0.1]);

% Store estimates at each time step
mu_sv = zeros(3,201);
mu_sv(:,1) = mu;
cov_sv = zeros(3,N);
cov_sv(:,1) = [1; 1; 0.1];

% UKF localization - loop through data
for i=2:N
    % Prediction step
    % Control noise covariance
    M = diag([alph1*v_c(i)^2 + alph2*om_c(i)^2, ...
              alph3*v_c(i)^2 + alph4*om_c(i)^2]);

    % Measurement noise covariance
    Q = diag([sig_r^2, sig_ph^2]);

    % Augmented state estimate: state + control noise + measurement noise
    mu_a = [mu' 0 0 0 0]';
    Sig_a = [  Sig      zeros(3,2) zeros(3,2); ...
             zeros(2,3)   M        zeros(2,2); ...
             zeros(2,3) zeros(2,2)   Q];

    % Generate sigma points
    Chi_a = [mu_a (mu_a(:,ones(n,1))+gam*chol(Sig_a)) (mu_a(:,ones(n,1))-gam*chol(Sig_a))];
    Chi_x = Chi_a(1:3,:);
    Chi_u = Chi_a(4:5,:);
    Chi_z = Chi_a(6:7,:);

    % Propogate state sigma points from prior time to current time
    Chi_x_bar = zeros(size(Chi_x));
    for k=1:15,
        Chi_x_bar(:,k) = prop_state_sig_pts(u_c(:,i),Chi_u(:,k),Chi_x(:,k),dt);
    end

    % Calculate weighted mean and covariance of state sigma points
    mu_bar = Chi_x_bar*wm';
    Sig_bar = ((wc(ones(1,3),:))).*(Chi_x_bar-mu_bar(:,ones(15,1)))*(Chi_x_bar-mu_bar(:,ones(15,1)))';

    mu = mu_bar;
    Sig = Sig_bar;

    % Measurement update step
%    for j=1:MM,
%        [mu,Sig] = meas_up_UKF(X(:,i),Chi_x_bar,Chi_z,mu_bar,Sig_bar,sig,m(:,j),wm,wc);
        j = 1 + mod(i,MM);
        [mu,Sig] = meas_up_UKF(X(:,i),Chi_x_bar,Chi_z,mu,Sig,sig,m(:,j),wm,wc);
%    end

    mu_sv(:,i) = mu;
    cov_sv(:,i) = [Sig(1,1); Sig(2,2); Sig(3,3)];

end

mu_x = mu_sv(1,:);
mu_y = mu_sv(2,:);
mu_th = mu_sv(3,:);

err_bnd_x = 2*sqrt(cov_sv(1,:));
err_bnd_y = 2*sqrt(cov_sv(2,:));
err_bnd_th = 2*sqrt(cov_sv(3,:));

figure(2); clf;
subplot(311);
```

```
plot(t,x,t,mu_x);
ylabel('x position (m)');
legend('true','estimated','Location','NorthWest');
subplot(312);
plot(t,y,t,mu_y);
ylabel('y position (m)')
subplot(313);
plot(t,180/pi*th,t,180/pi*mu_th);
xlabel('time (s)');
ylabel('heading (deg)');

figure(3); clf;
subplot(311);
plot(t,x-mu_x,'b-',t,err_bnd_x,'r--',t,-err_bnd_x,'r--');
ylabel('x position error (m)');
axis([0 20 -0.5 0.5]);
subplot(312);
plot(t,y-mu_y,'b-',t,err_bnd_y,'r--',t,-err_bnd_y,'r--');
ylabel('y position error (m)')
axis([0 20 -0.5 0.5]);
subplot(313);
plot(t,180/pi*(th-mu_th),'b-',t,180/pi*err_bnd_th,'r--',t,-180/pi*err_bnd_th,'r--');
xlabel('time (s)');
ylabel('heading error (deg)');
axis([0 20 -10 10]);
```

```matlab
% This function propogates the sigma points corresponding to the state
% through the dynamics of the system

function Chi_x = prop_state_sig_pts(u_c,Chi_u,Chi_x,dt)

    % Commanded velocities with added noise from sigma point model of noise
    v = u_c(1) + Chi_u(1);
    om = u_c(2) + Chi_u(2);

    % States from sigma point model of state
    x = Chi_x(1);
    y = Chi_x(2);
    th = Chi_x(3);

    % Propogate state forward in time
    x = x + (-v/om*sin(th) + v/om*sin(th+om*dt));
    y = y + (v/om*cos(th) - v/om*cos(th+om*dt));
    th = th + om*dt;

    Chi_x = [x; y; th];

end
```

```matlab
function [mu,Sig] = meas_up_UKF(X,Chi_x_bar,Chi_z,mu,Sig,sig,m,wm,wc)

% This function performs the measurement update for a UKF corresponding
% to a specific landmark m. See lines 10-16 of Table 7.4 in Probabilistic
% Robotics by Thrun, et al.

    x = X(1);           % true states used to create measurements
    y = X(2);
    th = X(3);

    mx = m(1);          % known land mark location
    my = m(2);

    sig_r = sig(1);     % noise levels on sensor measurments
    sig_ph = sig(2);

    % Predict measurements at sigma points
    Z_bar = zeros(size(Chi_z));
    for k=1:15,
        Z_bar(:,k) = pred_meas_sig_pts(Chi_x_bar(:,k),Chi_z(:,k),m);
    end

    % Compute statistics for measurement update
    z_hat = Z_bar*wm';
    S = ((wc(ones(1,2),:))).*(Z_bar-z_hat(:,ones(15,1)))*(Z_bar-z_hat(:,ones(15,1)))';
    Sig_xz = ((wc(ones(1,3),:))).*(Chi_x_bar-mu(:,ones(15,1)))*(Z_bar-z_hat(:,ones(15,1)))';

    % Update mean and covariance of estimate with measurements
    % Kalman gain
    K = Sig_xz/S;

    % Measurements: truth + noise
    range = sqrt((mx-x).^2 + (my-y).^2) + sig_r*randn;
    bearing = atan2(my-y,mx-x) - th + sig_ph*randn;
    z = [range; bearing];

    % Measurment update
    mu = mu + K*(z-z_hat);
    Sig = Sig - K*S*K';

end
```

```matlab
% This function predicts measurements corresponding to the state and sensor
% noise sigma points

function Zbar = pred_meas_sig_pts(Chi_x,Chi_z,m)

    % States from sigma point model
    x = Chi_x(1);
    y = Chi_x(2);
    th = Chi_x(3);

    % Sensor noise from sigma point model
    r_noise = Chi_z(1);
    ph_noise = Chi_z(2);

    % Landmark coordinates
    mx = m(1);
    my = m(2);

    % Generate measurments from sigma points (state and noise)
    range = sqrt((mx-x)^2 + (my-y)^2) + r_noise;
    bearing = atan2(my-y,mx-x) - th + ph_noise;
    Zbar = [range; bearing];

end
```