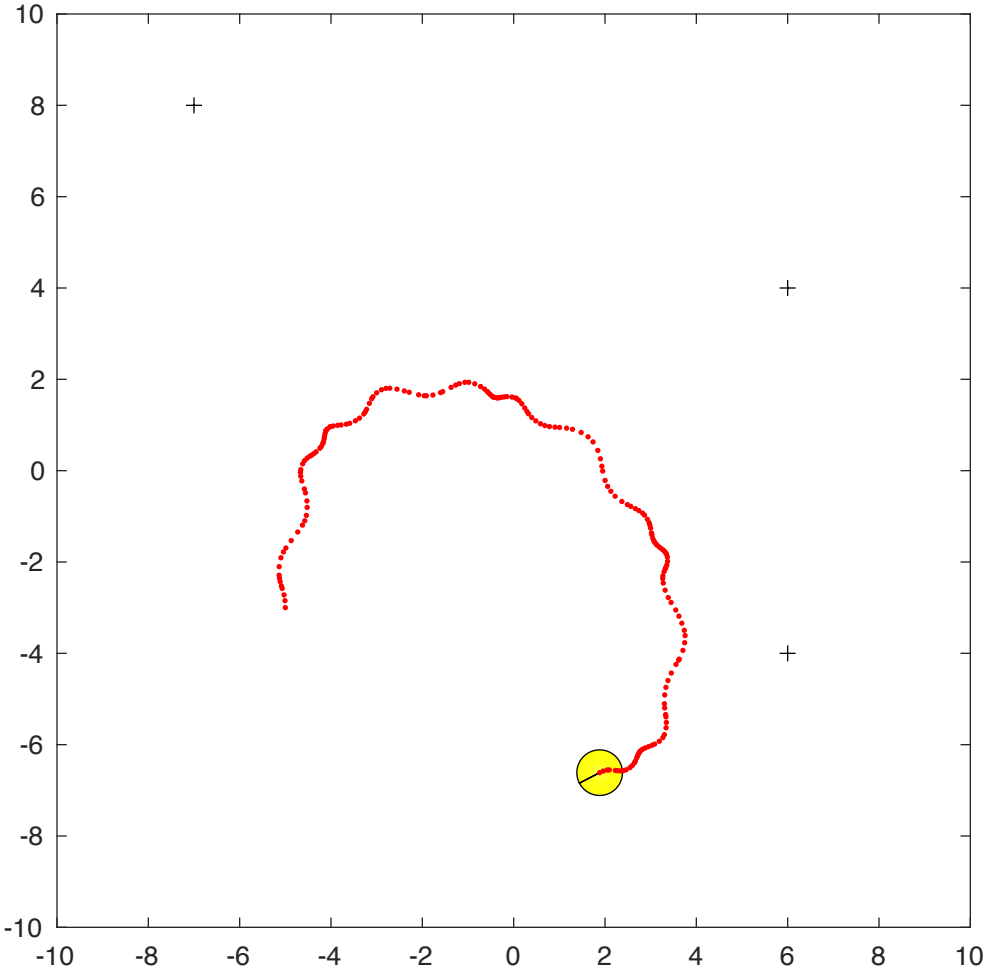
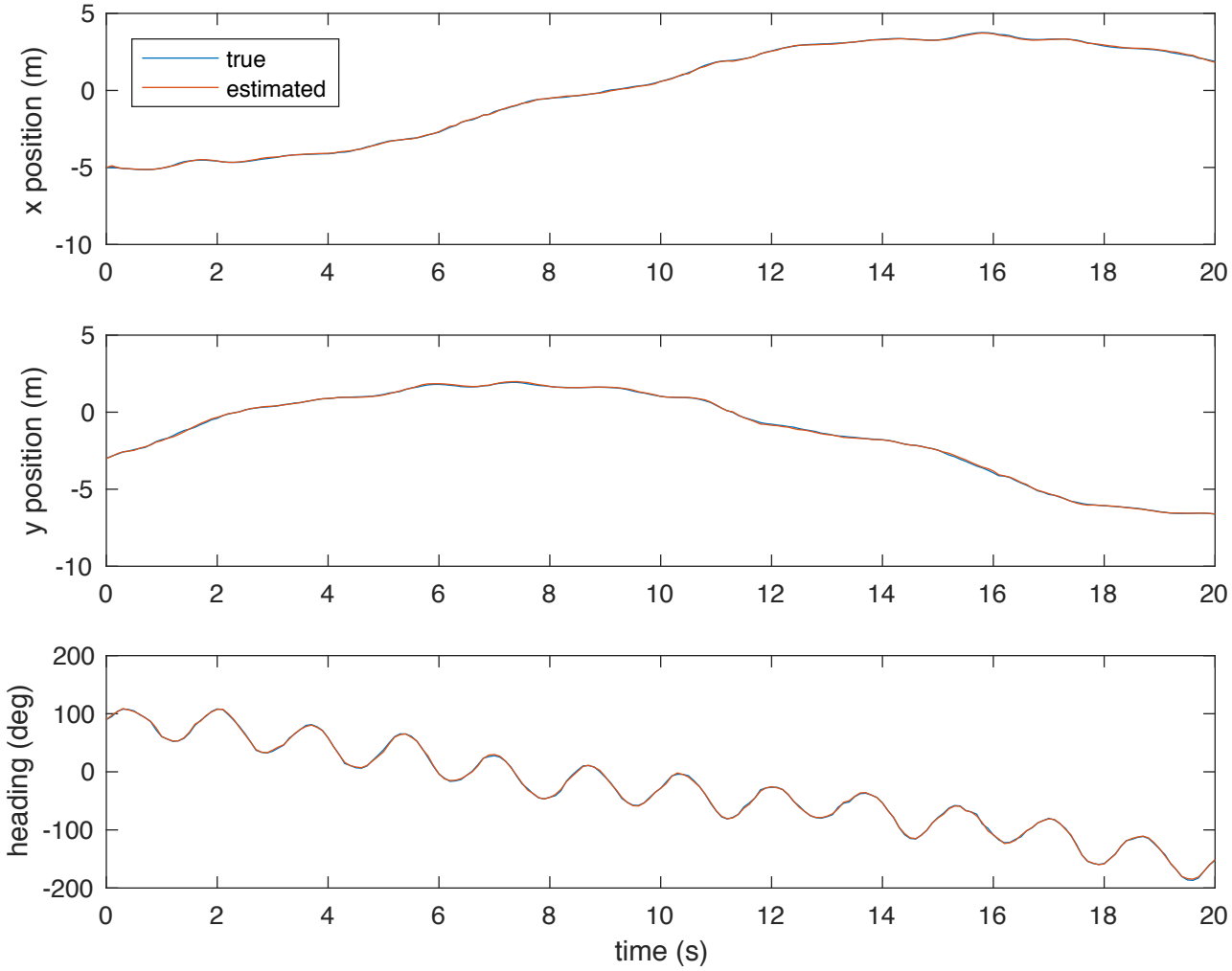
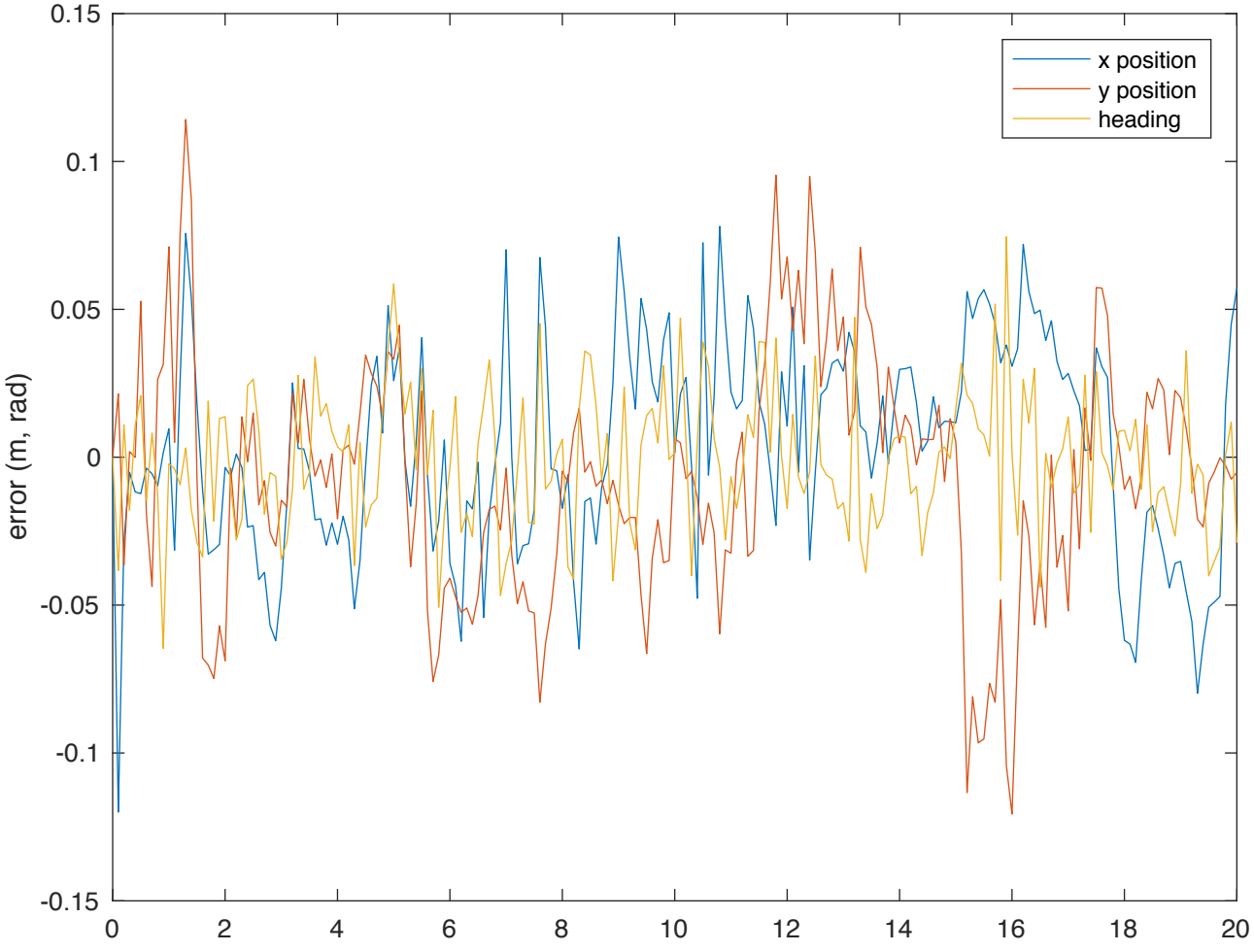

10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-8
-6
-4
-2
0
2
4
6
8
10


true
estimated
x position (m)
y position (m)
heading (deg)
time (s)
5
0
-5
-10
200
100
-100
-200
0
2
4
6
8
10
12
14
16
18
20


x position
y position
heading
error (m, rad)
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
0
2
4
6
8
10
12
14
16
18
20

```
% robot_EKF_mult_lm.m
%
% EKF localization implementation from Probabilistic Robotics, Thrun et
% al., Table 7.2
%
% This implementation handles multiple landmarks
%
% State variables are (x,y,th)
% State estimates are (mu_x,mu_y,mu_th)
% Estimation error covariance matrix is Sig

clear all;
% load('randvars.mat');     % load a data file with set random noise
                            % for debugging
dt = 0.1;
tfinal = 20;
t = 0:dt:tfinal;

N = length(t);

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

% Motion input plus noise model
v_c = 1.0 + 0.5*sin(2*pi*0.2*t);
om_c = -0.2 + 2*cos(2*pi*0.6*t);
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;

% Landmark (feature) locations
mx = [6 -7 6];      % x-coordinate of landmarks
my = [4 8 -4];      % y-coordinate of landmarks
ms = [1 2 3];       % signature of landmarks
m = [mx; my; ms];
MM = 3;             % number of landmarks

% 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 EKF from Table 7.2
% EKF parameters
sig_r = 0.1;
sig_ph = 0.05;
sig_s = 0.00001;
sig = [sig_r sig_ph sig_s];

% Initial conditions of state estimates at time zero
mu_x = x0+0.5;
mu_y = y0-0.7;
mu_th = th0-0.05;
Sig = diag([1,1,0.1]);

mu_x_sv = zeros(1,N);
mu_y_sv = zeros(1,N);
mu_th_sv = zeros(1,N);

mu_x_sv(1) = x0;
mu_y_sv(1) = y0;
mu_th_sv(1) = th0;

% EKF implementation -- loop through data
for i=2:N
    % Prediction step
    Th = mu_th;     % Use prior theta to predict current states

    % Jacobian of g(u(t),x(t-1)
    G = eye(3);
    G(1,3) = -v_c(i)/om_c(i)*cos(Th) + v_c(i)/om_c(i)*cos(Th+om_c(i)*dt);
    G(2,3) = -v_c(i)/om_c(i)*sin(Th) + v_c(i)/om_c(i)*sin(Th+om_c(i)*dt);

    % Jacobian to map noise from control space to state space
    V = zeros(3,2);
    V(1,1) = (-sin(Th)+sin(Th+om_c(i)*dt))/om_c(i);
    V(1,2) = (v_c(i)*(sin(Th)-sin(Th+om_c(i)*dt)))/(om_c(i)^2) + (v_c(i)*cos(Th+om_c(i)
*dt)*dt)/om_c(i);
    V(2,1) = (cos(Th)-cos(Th+om_c(i)*dt))/om_c(i);
    V(2,2) = -(v_c(i)*(cos(Th)-cos(Th+om_c(i)*dt)))/(om_c(i)^2) + (v_c(i)*sin(Th+om_c(i)
*dt)*dt)/om_c(i);
    V(3,2) = dt;

    % Control noise covariance
    M = eye(2);
    M(1,1) = alph1*v_c(i)^2 + alph2*om_c(i)^2;
    M(2,2) = alph3*v_c(i)^2 + alph4*om_c(i)^2;

    % State estimate - prediction step
    mu_x = mu_x + (-v_c(i)/om_c(i)*sin(Th) + v_c(i)/om_c(i)*sin(Th+om_c(i)*dt));
    mu_y = mu_y + (v_c(i)/om_c(i)*cos(Th) - v_c(i)/om_c(i)*cos(Th+om_c(i)*dt));
    mu_th = mu_th + om_c(i)*dt;
    mu = [mu_x; mu_y; mu_th];

    % State covariance - prediction step
    Sig = G*Sig*G' + V*M*V';

    % Measurement update step
    for j=1:MM,
```

```
        [mu,Sig] = meas_up_EKF(X(:,i),mu,Sig,m(:,j),sig);
    end

    mu_x = mu(1);
    mu_y = mu(2);
    mu_th = mu(3);

    mu_x_sv(i) = mu_x;
    mu_y_sv(i) = mu_y;
    mu_th_sv(i) = mu_th;

end

figure(2); clf;
subplot(311);
plot(t,x,t,mu_x_sv);
ylabel('x position (m)');
legend('true','estimated','Location','NorthWest');
subplot(312);
plot(t,y,t,mu_y_sv);
ylabel('y position (m)')
subplot(313);
plot(t,180/pi*th,t,180/pi*mu_th_sv);
xlabel('time (s)');
ylabel('heading (deg)');
```

```
function [mu,Sig] = meas_up_EKF(X,mu,Sig,m,sig)

% This function performs the measurement update corresponding to a
% specific landmark m. See lines 9-20 of Table 7.2 in Probabilistic
% Robotics by Thrun, et al.

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

mu_x = mu(1);       % estimated states to be updated by measurement
mu_y = mu(2);
mu_th = mu(3);

mx = m(1);          % known land mark location and signature
my = m(2);
ms = m(3);

sig_r = sig(1);     % s.d. of noise levels on measurements
sig_ph = sig(2);
sig_s = sig(3);

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

% Calculate predicted measurement based on state estimate
q = (mx-mu_x)^2 + (my-mu_y)^2;
zhat = zeros(3,1);
zhat(1) = sqrt(q);
zhat(2) = atan2((my-mu_y),(mx-mu_x)) - mu_th;
zhat(3) = ms;

% Jacobian of measurement function wrt state
H = zeros(3,3);
H(1,1) = -(mx-mu_x)/sqrt(q);
H(1,2) = -(my-mu_y)/sqrt(q);
H(2,1) = (my-mu_y)/q;
H(2,2) = -(mx-mu_x)/q;
H(2,3) = -1;

% Total uncertainty in predicted measurement
Q = diag([sig_r^2, sig_ph^2, sig_s^2]);
S = H*Sig*H' + Q;

% Kalman gain
K = (Sig*H')/S;

% Measurment update
mu = mu + K*(z-zhat);
Sig = (eye(3) - K*H)*Sig;

end
```