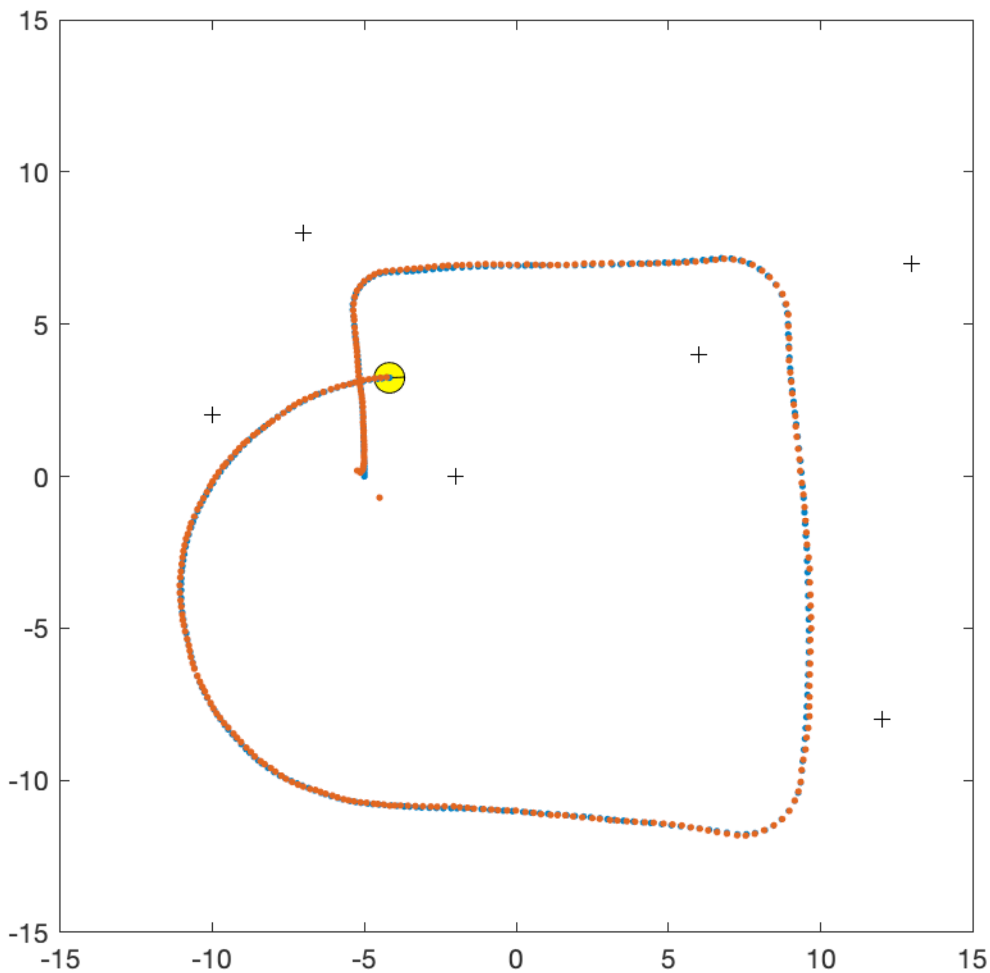
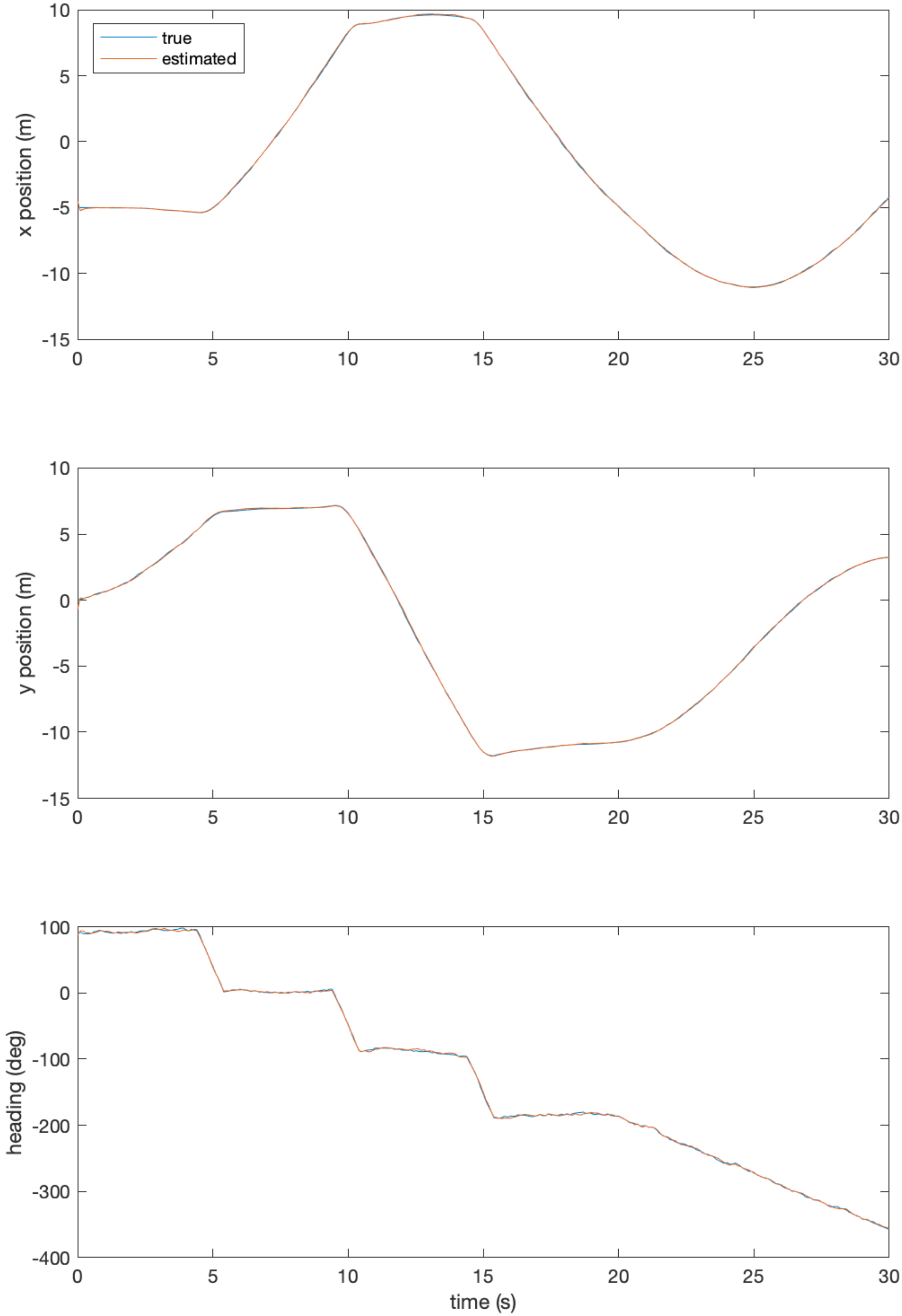
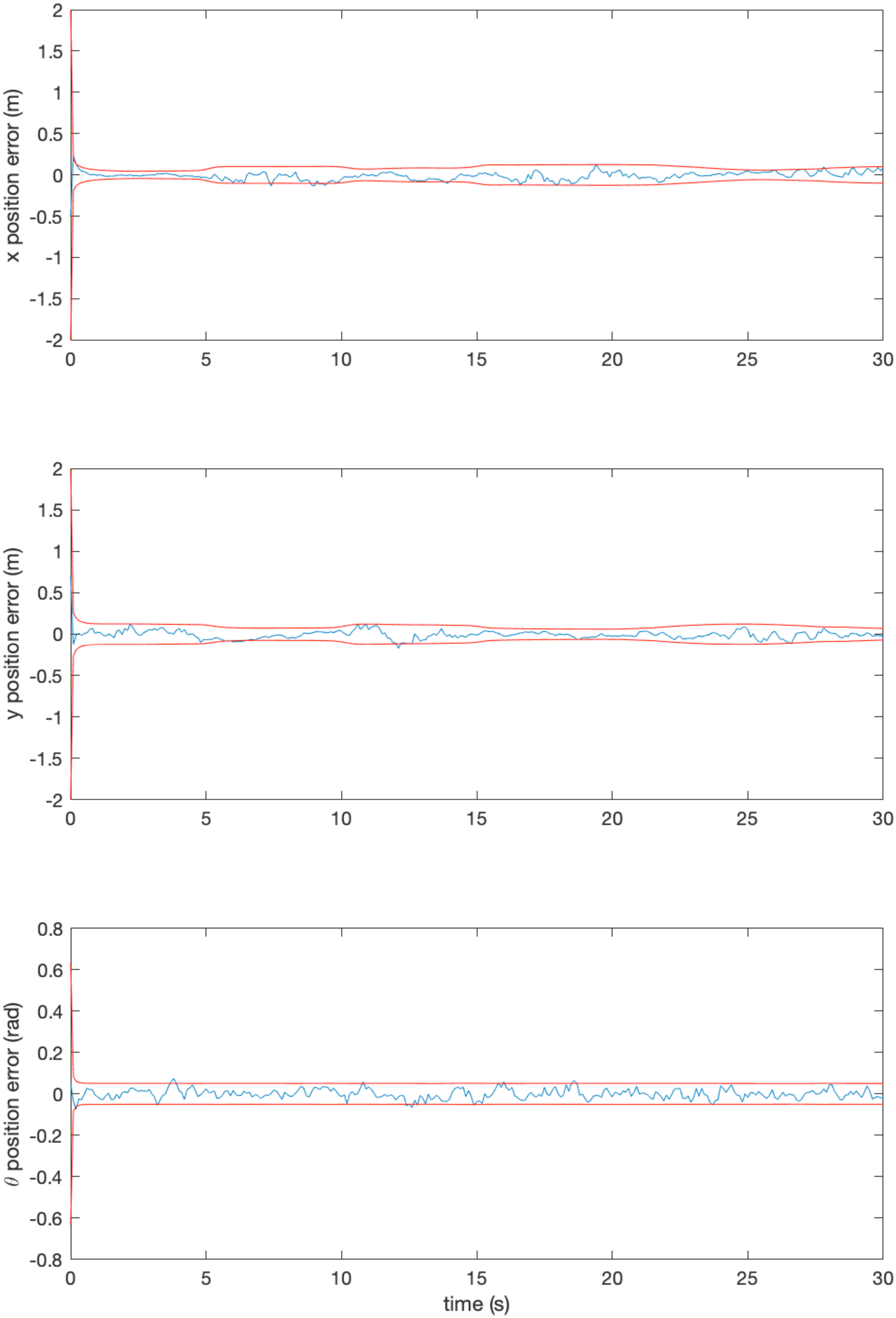
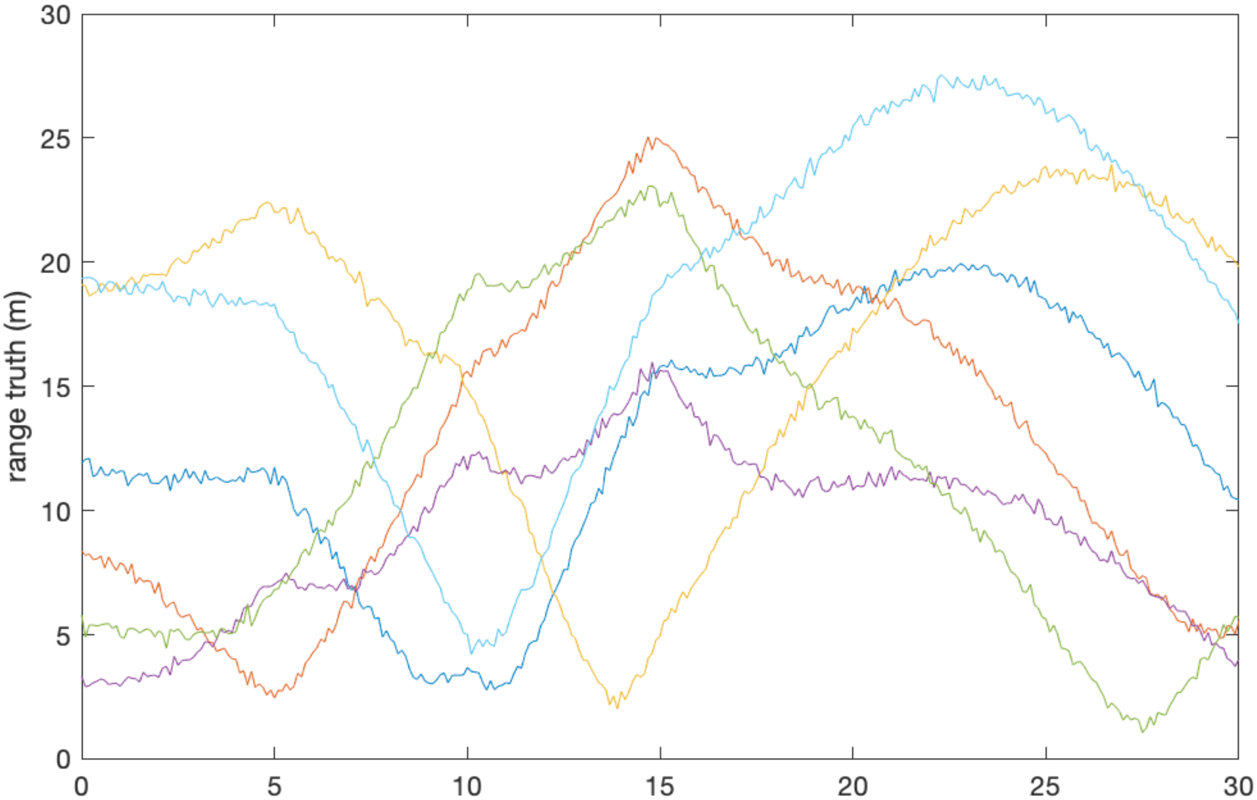


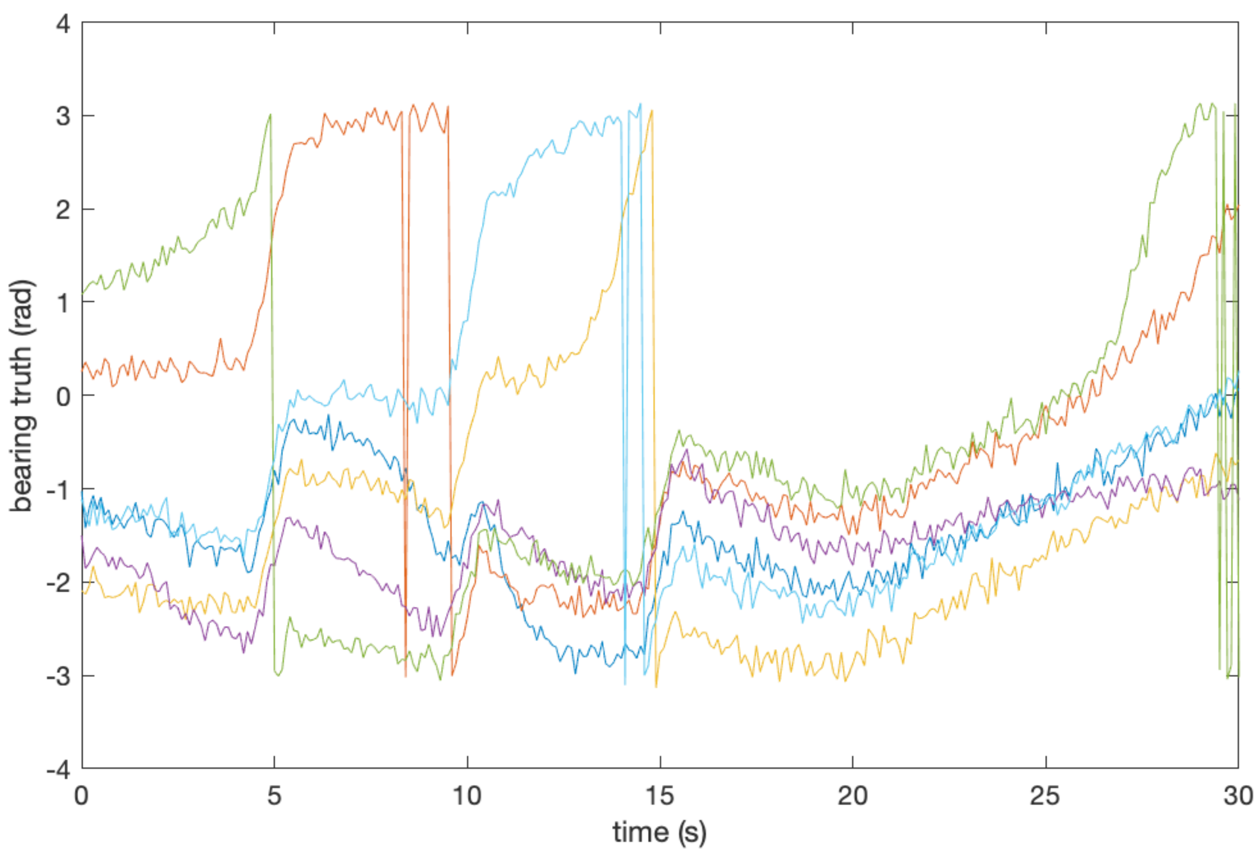
15
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15


true
estimated
x position (m)
y position (m)
heading (deg)
time (s)
10
5
0
-5
-10
-15
100
-100
-200
-300
-400
15
20
25
30

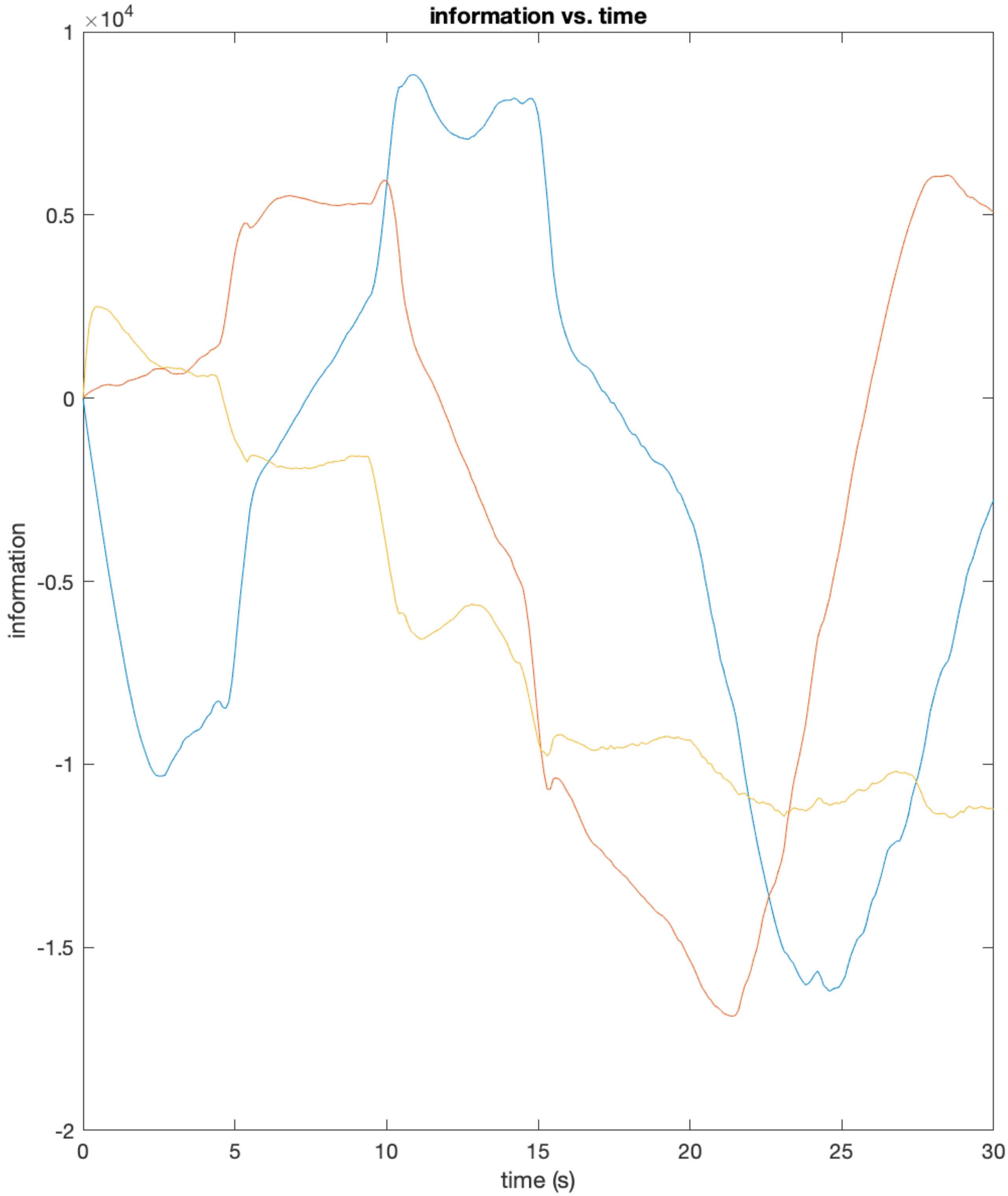

x position error (m)
y position error (m)
θ position error (rad)
time (s)



range truth (m)



bearing truth (rad)
time (s)


information vs. time
×10^4
1
0.5
0
-0.5
-1
-1.5
-2
information
0
5
10
15
20
25
30
time (s)

```matlab
% robot_EIF.m
%
% EIF localization for multirotor using "unicycle dynamics"
%
% State variables are (x,y,th)
% Information vector is xi
% state belief = mu = (mu_x,mu_y,mu_th)'
% Information matrix is Om

clear all;
load('midterm_data.mat');     % load data file

dt = 0.1;
% tfinal = 30;
% t = 0:dt:tfinal;

N = length(t);

% Initial conditions
x0 = -5;
y0 = 0;
th0 = pi/2;

% Motion input plus noise model
% v_c = 3*ones(1,N);
% v_c = [1*ones(1,20) 2*ones(1,40) 3*ones(1,60) 4*ones(1,50) 3*ones(1,31)];
% v_c = [(0.5+0.3*t(1:120)) (0.5+0.3*t(120)-0.25*t(1:81)) 2.07*ones(1,100)] ;
% om_c = -pi/2*[zeros(1,45) ones(1,10) zeros(1,40) ones(1,10) zeros(1,40) ones(1,10) zeros(1,46) 0.2*ones(1,100)];

m_v = 0.3;
m_om = 0.2;
M = diag([m_v^2 m_om^2]);

% v = v_c + m_v*randn(1,N);
% om = om_c + m_om*randn(1,N);

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

% Landmark (feature) locations
mx = [6 -7 12 -2 -10 13];      % x-coordinate of landmarks
my = [4 8 -8 0 2 7];      % y-coordinate of landmarks
m = [mx; my];
MM = 6;             % number of landmarks

% measurement noise parameters
sig_r = 0.2;
sig_ph = 0.1;
sig = [sig_r sig_ph];

% range_tr = zeros(N,MM);
% bearing_tr = zeros(N,MM);

% use truth data loaded from data file
x = X_tr(1,:);
y = X_tr(2,:);
th = X_tr(3,:);

% Draw robot at time step 1
drawRobot(x(1),y(1),th(1),m,t(1));
% Calculate measurement truth data at time step 1
% for j=1:MM
%     z_tr = meas_truth([x(1);y(1);th(1)],m(:,j),sig);
%     range_tr(1,j) = z_tr(1);
%     bearing_tr(1,j) = z_tr(2);
% end


% Create pose truth data
for i = 2:N
%    x(i) = x(i-1) + v(i)*cos(th(i-1))*dt;
%    y(i) = y(i-1) + v(i)*sin(th(i-1))*dt;
%    th(i) = th(i-1) + om(i)*dt;
   drawRobot(x(i),y(i),th(i),m,t(i));
   X_i = [x(i); y(i); th(i)];
   pause(0.01);
   % Calculate measurement truth data at time step i
%     for j=1:MM
%         z_tr = meas_truth(X_i,m(:,j),sig);
%         range_tr(i,j) = z_tr(1);
%         bearing_tr(i,j) = z_tr(2);
```

```
%     end
end

X = [x; y; th];     % matrix of true state vectors at all times

% Initial conditions of state estimates at time zero
mu = zeros(3,1);
mubar = zeros(3,1);
mu(1) = x0+0.5;
mu(2) = y0-0.7;
mu(3) = th0-0.05;
mu_0 = mu;

Om = diag([1,1,10]);
xi = Om*mu;

xi_sv = zeros(3,N);
xi_sv(:,1) = xi;

mu_sv = zeros(3,N);
mu_sv(:,1) = mu;

cov_sv = zeros(3,N);
cov_sv(:,1) = [1; 1; 0.1];


% EKF implementation -- loop through data
for i=2:N
    mu = Om\xi;

    % Prediction step
    Th = mu(3);     % Use prior theta to predict current states

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

    % Jacobian to map noise from control space to state space
    V = zeros(3,2);
    V(1,1) = dt*cos(Th);
    V(1,2) = 0;
    V(2,1) = dt*sin(Th);
    V(2,2) = 0;
    V(3,2) = dt;

    % State estimate - prediction step
    mubar(1) = mu(1) + v_c(i)*cos(Th)*dt;
    mubar(2) = mu(2) + v_c(i)*sin(Th)*dt;
    mubar(3) = mu(3) + om_c(i)*dt;

    % Information matrix - prediction step
    Ombar = inv(G/Om*G' + V*M*V');

    % Information vector - prediction step
    xibar = Ombar*mubar;

    % Measurement update step
    for j=1:MM
        [xibar,Ombar] = meas_up_EIF(range_tr(i,j),bearing_tr(i,j),mubar,xibar,Ombar,m(:,j),sig);
    end

    xi = xibar;
    Om = Ombar;

    xi_sv(:,i) = xi;

    Sig = inv(Om);
    mu = Sig*xi;

    mu_sv(:,i) = mu;

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

end

X_tr = X;

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(1); hold on;
plot(x,y,'.',mu_sv(1,:),mu_sv(2,:),'.'); hold off;

figure(2); clf;
subplot(311);
plot(t,x,t,mu_sv(1,:));
ylabel('x position (m)');
legend('true','estimated','Location','NorthWest');
subplot(312);
plot(t,y,t,mu_sv(2,:));
ylabel('y position (m)')
subplot(313);
plot(t,180/pi*th,t,180/pi*mu_sv(3,:));
xlabel('time (s)');
ylabel('heading (deg)');

figure(3); clf;
subplot(311);
plot(t,x-mu_sv(1,:),t,err_bnd_x,'r',t,-err_bnd_x,'r');
ylabel('x position error (m)');
subplot(312);
plot(t,y-mu_sv(2,:),t,err_bnd_y,'r',t,-err_bnd_y,'r');
ylabel('y position error (m)');
subplot(313);
plot(t,th-mu_sv(3,:),t,err_bnd_th,'r',t,-err_bnd_th,'r');
ylabel('\theta position error (rad)');
xlabel('time (s)');

figure(4); clf;
subplot(211);
plot(t,range_tr);
ylabel('range truth (m)');
subplot(212);
plot(t,bearing_tr);
ylabel('bearing truth (rad)');
xlabel('time (s)');

figure(5); clf;
plot(t,xi_sv);
xlabel('time (s)');
ylabel('information');
title('information vs. time');
```

```
function [xi,Om] = meas_up_EIF(range,bearing,mubar,xibar,Ombar,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.

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

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

% Measurements
z = [range; bearing];

% Calculate predicted measurement based on state estimate
q = (mx-mubar(1))^2 + (my-mubar(2))^2;
zbar = zeros(2,1);
zbar(1) = sqrt(q);
zbar(2) = wrap_ang(atan2((my-mubar(2)),(mx-mubar(1))) - mubar(3));

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

% Total uncertainty in predicted measurement
Q = diag([sig_r^2, sig_ph^2]);

res = z - zbar;
res(2) = wrap_ang(res(2));

Om = Ombar + H'/Q*H;
xi = xibar + H'/Q*(res + H*mubar);

end
```