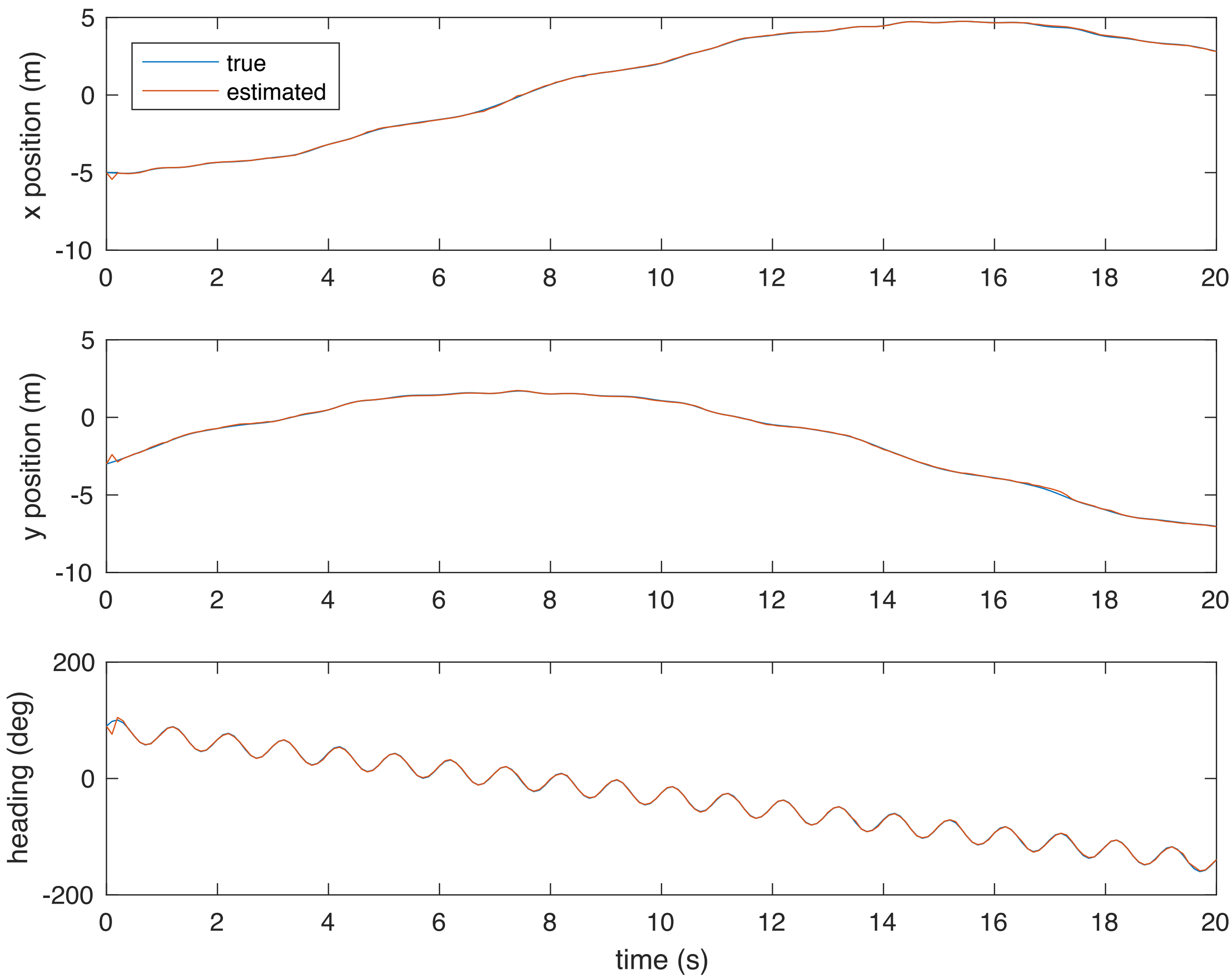
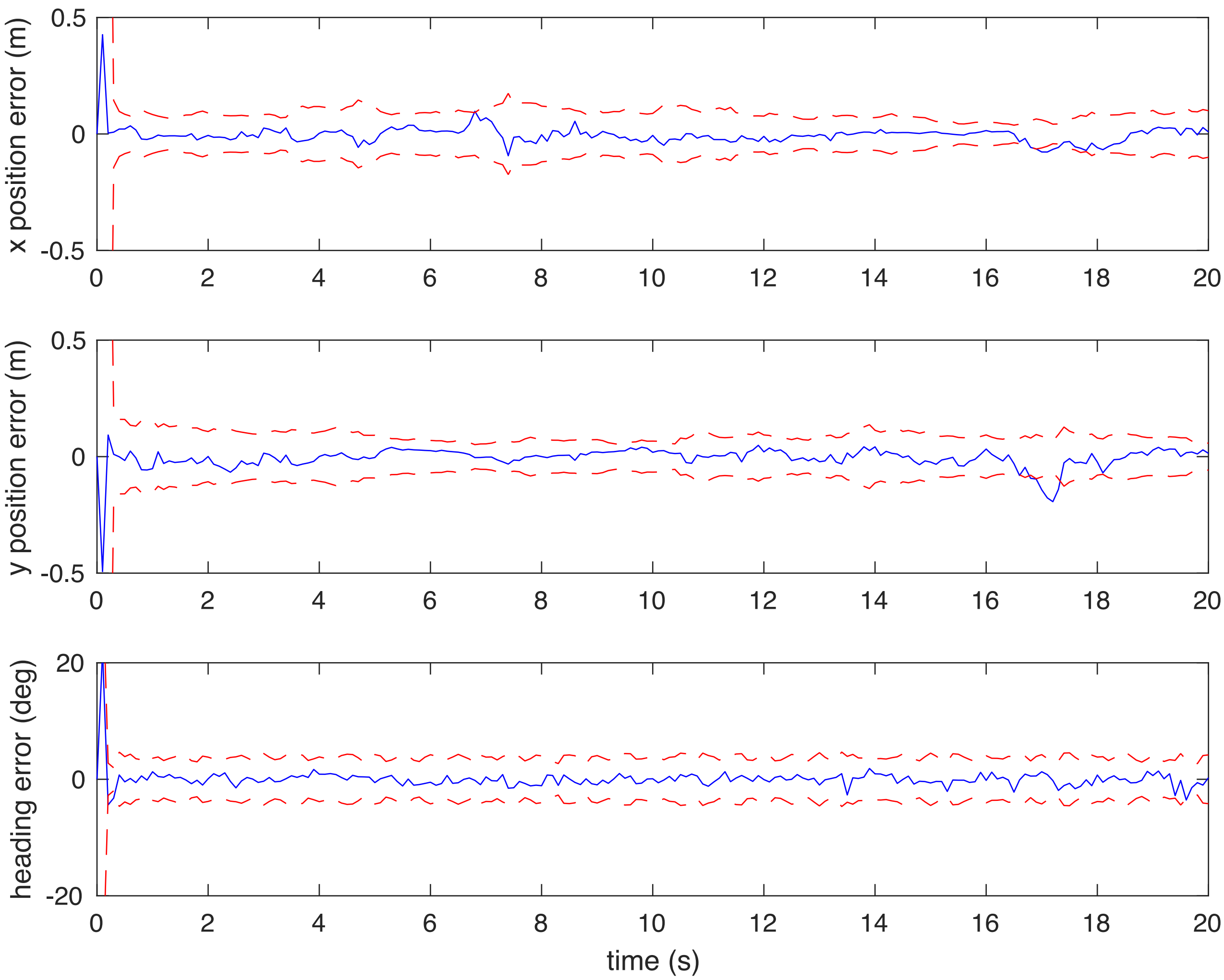

true
estimated
x position (m)
y position (m)
heading (deg)
time (s)



x position error (m)
0.5
0
-0.5
0 2 4 6 8 10 12 14 16 18 20
y position error (m)
0.5
0
-0.5
0 2 4 6 8 10 12 14 16 18 20
heading error (deg)
20
0
-20
0 2 4 6 8 10 12 14 16 18 20
time (s)

```
% Ground robot localization using Monte Carlo Localization from Probabilistic
% Robotics, Thrun et al., Table 8.2
%
% State variables true values are (x_tr,y_tr,th_tr)

clear all;

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

N = length(t);

% Initial conditions
x_tr0 = -5;
y_tr0 = -3;
th_tr0 = 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_tr = 1 + 0.5*sin(2*pi*0.3*t);     % defining tr = true = noise free for the inputs
om_tr = -0.2 + 2*cos(2*pi*1*t);
u_tr = [v_tr; om_tr];
alph1 = 0.1;
alph2 = 0.01;
alph3 = 0.01;
alph4 = 0.1;
alph5 = 0.01;
alph6 = 0.01;
alpha = [alph1 alph2 alph3 alph4 alph5 alph6];

v = v_tr + sqrt(alph1*v_tr.^2+alph2*om_tr.^2).*randn(1,N);
om = om_tr + sqrt(alph3*v_tr.^2+alph4*om_tr.^2).*randn(1,N);
u = [v; om];

x_tr(1) = x_tr0;
y_tr(1) = y_tr0;
th_tr(1) = th_tr0;

% Draw robot at time step 1
drawRobot(x_tr(1),y_tr(1),th_tr(1),m,t(1));

for i = 2:N,
   x_tr(i) = x_tr(i-1) + (-v_tr(i)/om_tr(i)*sin(th_tr(i-1)) + v_tr(i)/om_tr(i)*sin(th_tr(i-
1)+om_tr(i)*dt));
   y_tr(i) = y_tr(i-1) + (v_tr(i)/om_tr(i)*cos(th_tr(i-1)) - v_tr(i)/om_tr(i)*cos(th_tr(i-
1)+om_tr(i)*dt));
   th_tr(i) = th_tr(i-1) + om_tr(i)*dt;
   drawRobot(x_tr(i),y_tr(i),th_tr(i),m,t(i));
   % pause(0.05);
end
X_tr = [x_tr; y_tr; th_tr];     % matrix of true state vectors at all times

% initialize particle set to be uniformly spread over state space
M = 1000;    % number of particles
% Chi0 = diag([20 20 pi/16])*(rand(3,M)-0.5) + [0*ones(1,M); 0*ones(1,M); pi/2*ones(1,M)];
Chi0 = diag([20 20 2*pi])*(rand(3,M)-0.5);

% create measurements: truth + noise
sig_r = 0.1;
sig_ph = 0.05;
sig = [sig_r; sig_ph];

% particles at initial time (i=1) with importance weights appended and set to zero
Chi_t_1 = [Chi0; zeros(1,M)];

% create vectors to hold estimate values
mu_x = zeros(1,N);
mu_y = zeros(1,N);
mu_th = zeros(1,N);

mu_x(1) = x_tr0;
mu_y(1) = y_tr0;
```

```
mu_th(1) = th_tr0;

P_x = zeros(1,N);
P_y = zeros(1,N);
P_th = zeros(1,N);

P_x(1) = cov(Chi0(1,:));
P_y(1) = cov(Chi0(2,:));
P_th(1) = cov(Chi0(3,:));

px = Chi0(1,:);
py = Chi0(2,:);
drawRobotParticles(mu_x(1),mu_y(1),mu_th(1),m,px,py,t(1));

% Monte Carlo localization : loop through the data
for i=2:N

    u_t = u(:,i);
    X_t = X_tr(:,i);    % true state at the current time

    % Call MCL algorithm
    Chi_t = MCL_alg_LV(Chi_t_1,u_t,X_t,alpha,sig,m,dt,i);

    mu_x(i) = mean(Chi_t(1,:));
    mu_y(i) = mean(Chi_t(2,:));
    mu_th(i) = mean(Chi_t(3,:));

    P_x(i) = cov(Chi_t(1,:));
    P_y(i) = cov(Chi_t(2,:));
    P_th(i) = cov(Chi_t(3,:));

%     px = Chi_t(1,:);
%     py = Chi_t(2,:);
%
%     drawRobotParticles(mu_x(i),mu_y(i),mu_th(i),m,px,py,t);
%     pause(0.1);

    Chi_t_1 = Chi_t;
end

err_bnd_x = 2*sqrt(P_x);
err_bnd_y = 2*sqrt(P_y);
err_bnd_th = 2*sqrt(P_th);

figure(2); clf;
subplot(311);
plot(t,x_tr,t,mu_x);
ylabel('x position (m)');
legend('true','estimated','Location','NorthWest');
subplot(312);
plot(t,y_tr,t,mu_y);
ylabel('y position (m)')
subplot(313);
plot(t,180/pi*th_tr,t,180/pi*mu_th);
xlabel('time (s)');
ylabel('heading (deg)');

figure(3); clf;
subplot(311);
plot(t,x_tr-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_tr-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_tr-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 -20 20]);
```

```matlab
function Chi_t = MCL_alg_LV(Chi_t_1,u_t,X_t,alpha,sig,map,dt,idx);
    % MCL_alg_LV.m
    % Monte Carlo localization algorithm
    % Uses low variance sampler from Table 4.4

    % Initialize vectors, matrix
    [N,M] = size(Chi_t_1);
    [~,L] = size(map);
    x_t = zeros(N-1,M);
    w_t = zeros(1,M);

    % propagate particles x_t and calculate weights w_t
    for m = 1:M
        x_t_1 = Chi_t_1(1:3,m);
        x_t(:,m) = samp_motion_model(u_t,x_t_1,alpha,dt);

        % calculate weight for each landmark
        for l=1:L
            w(l) = meas_model(X_t,x_t(:,m),map(:,l),sig);
        end
        w_t(m) = prod(w);  % particle weight
    end

    % Plot particles after sampling motion model
    mu_x = mean(x_t(1,:));
    mu_y = mean(x_t(2,:));
    mu_th = mean(x_t(3,:));

    x_tr = X_t(1,:);
    y_tr = X_t(2,:);
    th_tr = X_t(3,:);

    px = x_t(1,:);
    py = x_t(2,:);

%     drawRobotParticles(mu_x,mu_y,mu_th,m,px,py,1);
%     drawRobotParticles(x_tr,y_tr,th_tr,m,px,py,1);
%     pause(0.1);

    % Normalize weights
    w_t = w_t/sum(w_t);

    % Assemble particles
    Chi_bar_t = [x_t; w_t];

    Chi_t = LVsamp(Chi_bar_t,M);

    % Plot particles after resampling based on measurement
    mu_x = mean(Chi_t(1,:));
    mu_y = mean(Chi_t(2,:));
    mu_th = mean(Chi_t(3,:));

    px = Chi_t(1,:);
    py = Chi_t(2,:);

%     drawRobotParticles(x_tr,y_tr,th_tr,m,px,py,1);
%     pause(0.1);

end
```

```
function x_t = samp_motion_model(u_t,x_t_1,alpha,dt)
    % Sample the motion model utilizing the algorithm in Table 5.3

    % inputs at current time
    v = u_t(1);
    om = u_t(2);

    % standard deviation of input noise
    sd_v = sqrt(alpha(1)*v^2 + alpha(2)*om^2);
    sd_om = sqrt(alpha(3)*v^2 + alpha(4)*om^2);
    sd_gam = sqrt(alpha(5)*v^2 + alpha(6)*om^2);

    % sampled inputs
    vhat = v + sd_v*randn(1);
    omhat = om + sd_om*randn(1);
    gamhat = sd_gam*randn(1);

    % particle state at prior time
    x = x_t_1(1);
    y = x_t_1(2);
    th = x_t_1(3);

    % particle state at current time
    xpr = x + (-vhat/omhat*sin(th) + vhat/omhat*sin(th+omhat*dt));
    ypr = y + (vhat/omhat*cos(th) - vhat/omhat*cos(th+omhat*dt));
    thpr = th + omhat*dt + gamhat*dt;

    x_t = [xpr; ypr; thpr];

end
```

```matlab
function w_t = meas_model(X_t,x_t,map,sig)
    % likelihood model to calculate particle weights
    % See Table 6.4

    % actual states from robot (no sensor noise)
    x_tr = X_t(1);
    y_tr = X_t(2);
    th_tr = X_t(3);

    % landmark location
    mx = map(1);
    my = map(2);

    % measurements from robot (with sensor noise)
    sig_r = sig(1);
    sig_ph = sig(2);
    r = sqrt((mx-x_tr)^2 + (my-y_tr)^2) + sig_r*randn(1);
    ph = atan2(my-y_tr,mx-x_tr) - th_tr + sig_ph*randn(1);

    % particle state
    xp = x_t(1);
    yp = x_t(2);
    thp = x_t(3);

    % range and bearing from particle to landmark
    rp = sqrt((mx-xp)^2 + (my-yp)^2);
    php = wrapToPi(atan2(my-yp,mx-xp) - thp);

    rerr = rp-r;
    pherr = wrapToPi(php-ph);

    p_r = prob_normal_dist(rerr,sig_r^2);
    p_ph = prob_normal_dist(pherr,sig_ph^2);

    w_t = p_r*p_ph;

end
```

```matlab
function [Chi] = LVsamp(Chi_bar,M)
    % LVsamp.m
    %
    % Low-variance sampler according to Table 4.4 in Probabilistic Robotics text

    n = 3;  % number of states

    x_bar = Chi_bar(1:n,:);
    w_bar = Chi_bar(n+1,:);

    x = [];
    w = [];
    ind = [];

    r = rand/M;

    c = w_bar(1);
    i = 1;

    for m = 1:M
       U = r+(m-1)/M;
       while U > c
           i = i + 1;
           c = c + w_bar(i);
       end
       x = [x x_bar(:,i)];
       w = [w w_bar(i)];
       ind = [ind i];
    end

    % Combating particle deprivation
    P = cov(x_bar');                % covariance of prior

    uniq = length(unique(ind));     % number of unique particle in resampled cloud
    if uniq/M < 0.5                 % if there is a lot of duplication
        Q = P/((M*uniq)^(1/n));     % add noise to the samples
        x = x + Q*randn(size(x));
    end

%     wtmp = w;

    % reset weights to make them uniform
    w = ones(1,M)/M;

    % particles after resampling
    Chi = [x; w];

%     figure(5);
%     plot(x_bar(1,:),w_bar,'o',x(1,:),w,'o');
%     axis([-10 10 0 1]);
%     legend('weights','resampled');
%     plot(wtmp);
%     pause(0.1);

end
```