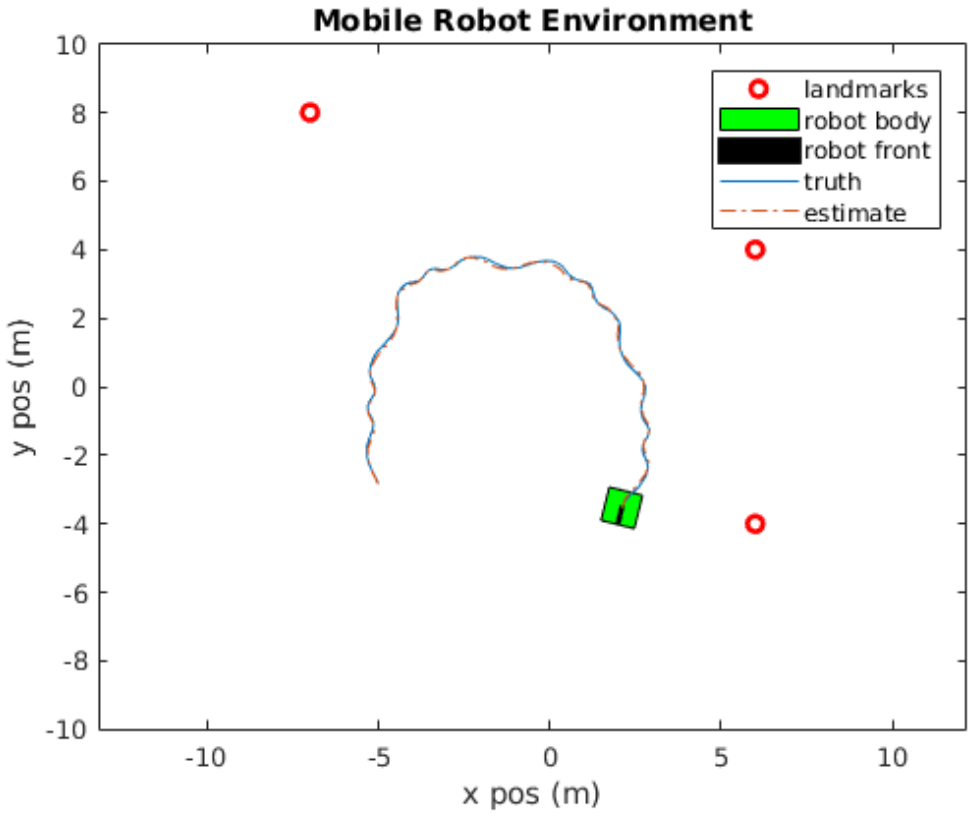
```
% Homework 2 Autonomous Two-wheele robot EKF
% Jesse Wynn
% September 25, 2017

clc;
clear all;
close all;

% time params
Ts = 0.1;   % sec
t = 0:Ts:20;

% commanded velocity
v_c = 1 + 0.5 * cos(2 * pi * (0.2) * t);
w_c = -0.2 + 2 * cos(2 * pi * (0.6) * t);

% noise params
alpha_1 = 0.1;
alpha_4 = 0.1;
alpha_2 = 0.01;
alpha_3 = 0.01;
alpha_5 = 0;
alpha_6 = 0;

sigma_r = 0.1;
sigma_phi = 0.05;

% landmark locations
landmarks = [6, -7 6;   % [x_positions; y_positions]
             4, 8, -4];

num_landmarks = size(landmarks);
num_landmarks = num_landmarks(2);

% initital conditions
x = -5;
y = -3;
theta = pi/2;
first = 0;
a = 0;

% initial state
mu_t = [x; y; theta];
Sigma_t = [1, 0, 0;
           0, 1, 0;
           0, 0, 1];

% plotting / storing stuff
x_true = zeros(1,length(t));
y_true = zeros(1,length(t));
theta_true = zeros(1,length(t));
```

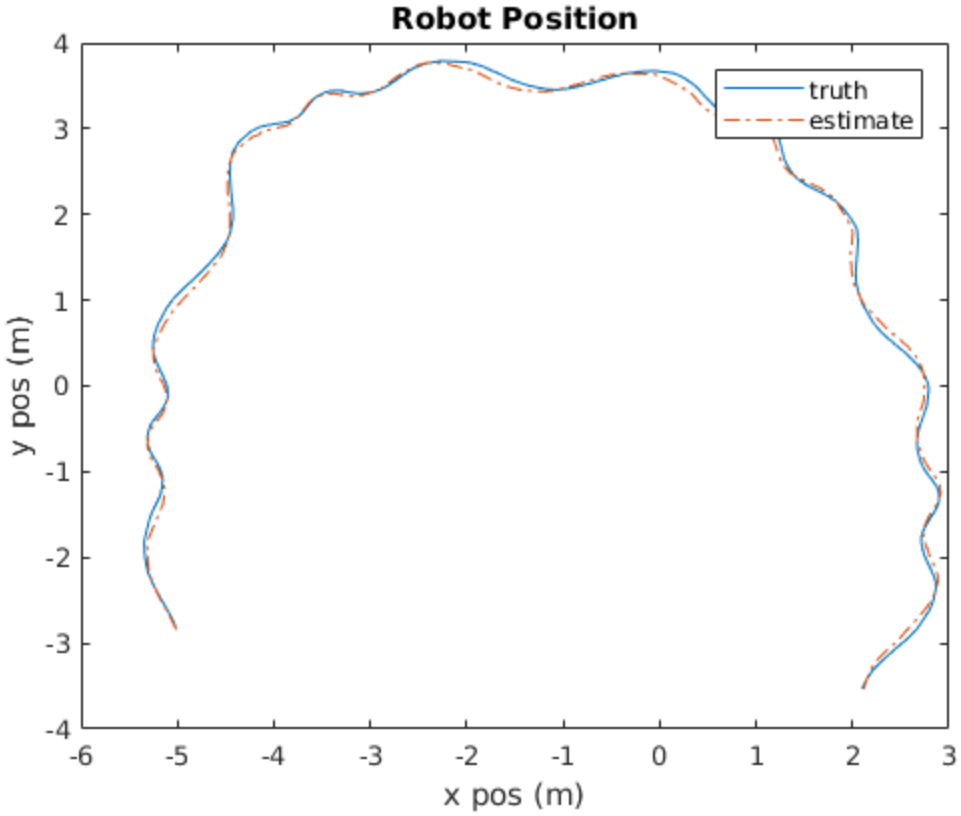
```
x_est = zeros(1,length(t));
y_est = zeros(1,length(t));
theta_estimated = zeros(1,length(t));

Sigma_x = zeros(1,length(t));
Sigma_y = zeros(1,length(t));
Sigma_theta = zeros(1,length(t));

range = zeros(length(t),3);
bearing = zeros(length(t),3);

% plot the first time
drawRobot(x,y,theta,landmarks,first);
first = 1;
for i=1:length(t)
    pause(0.01)
    % Task 1: Implement velocity motion model (Table 5.3)
    v_hat = v_c(i) + randn*sqrt(alpha_1*(v_c(i))^2 +
 alpha_2*(w_c(i))^2);
    w_hat = w_c(i) + randn*sqrt(alpha_3*(v_c(i))^2 +
 alpha_4*(w_c(i))^2);
    gamma_hat = randn*sqrt(alpha_5*(v_c(i))^2 + alpha_6*(w_c(i))^2);
    x = x - (v_hat/w_hat)*sin(theta) + (v_hat/w_hat)*sin(theta +
 w_hat*Ts);
    y = y + (v_hat/w_hat)*cos(theta) - (v_hat/w_hat)*cos(theta +
 w_hat*Ts);
    theta = theta + w_hat*Ts + gamma_hat*Ts;

    % update the plot
    drawRobot(x,y,theta,landmarks,first)

    % save some data for plotting
    x_true(i) = x;
    y_true(i) = y;
    theta_true(i) = theta;

    % Task 2: Simulate range and bearing measurements to landmarks
    range(i,1) = norm([x; y] - landmarks(:,1)) + randn*sigma_r^2;   %
 measurement + noise
    range(i,2) = norm([x; y] - landmarks(:,2)) + randn*sigma_r^2;
    range(i,3) = norm([x; y] - landmarks(:,3)) + randn*sigma_r^2;

    bearing(i,1) = atan2(landmarks(2,1) - y, landmarks(1,1) - x) -
 theta + randn*sigma_phi;  % measurement + noise
    bearing(i,2) = atan2(landmarks(2,2) - y, landmarks(1,2) - x) -
 theta + randn*sigma_phi;   % maybe sqrt
    bearing(i,3) = atan2(landmarks(2,3) - y, landmarks(1,3) - x) -
 theta + randn*sigma_phi;

    % Task 3: Implement the full EKF algorithm found in Table 7.2 (p.
 204)

    % input velocities
    v_t = v_hat;
```

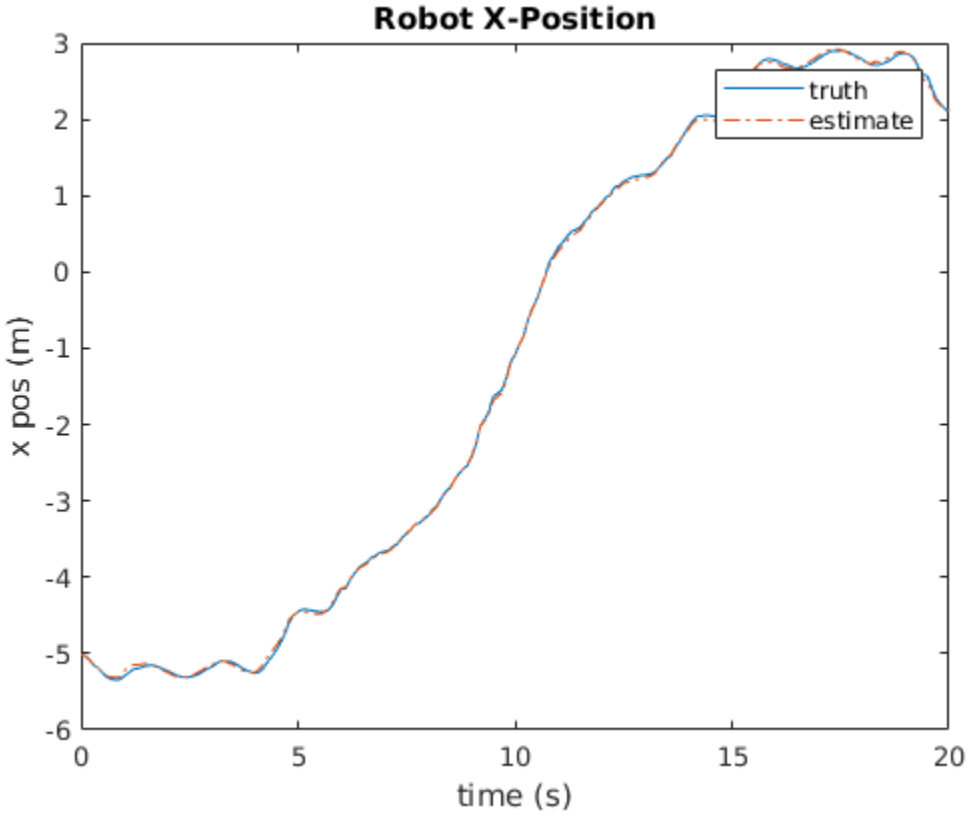
```
   w_t = w_hat;

   % line 2:
   theta_est = mu_t(3);
   % line 3:
   G_t = [1, 0, -(v_t/w_t)*cos(theta_est) + (v_t/w_t)*cos(theta_est +
w_t*Ts);
          0, 1, -(v_t/w_t)*sin(theta_est) + (v_t/w_t)*sin(theta_est +
w_t*Ts);
          0, 0, 1];
   % line 4:
   V_t = [(-sin(theta_est)+sin(theta_est + w_t*Ts))/w_t,
v_t*(sin(theta_est)-sin(theta_est + w_t*Ts))/w_t^2 +
v_t*cos(theta_est + w_t*Ts)*Ts/w_t;
          (cos(theta_est) - cos(theta_est + w_t*Ts))/w_t,
-v_t*(cos(theta_est) - cos(theta_est + w_t*Ts))/w_t^2 +
v_t*sin(theta_est + w_t*Ts)*Ts/w_t;
          0, Ts];
   % line 5:
   M_t = [alpha_1*v_t^2 + alpha_2*w_t^2, 0;
          0, alpha_3*v_t^2 + alpha_4*w_t^2];
   % line 6:
   mu_t = mu_t + [-(v_t/w_t)*sin(theta_est) + (v_t/w_t)*sin(theta_est
+ w_t*Ts);
                  (v_t/w_t)*cos(theta_est) - (v_t/w_t)*cos(theta_est
+ w_t*Ts);
                  w_t*Ts];
   % line 7:
   Sigma_t = G_t*Sigma_t*G_t' + V_t*M_t*V_t';
   % line 8:
   Q_t = [sigma_r^2, 0;
          0, sigma_phi^2];
   % line 9 (begin measurement updates):
   for j = 1:num_landmarks
       % line 11 (skip line 10):
       q = (landmarks(1,j) - mu_t(1))^2 + (landmarks(2,j) -
mu_t(2))^2;
       % line 12 (predicted measurement):
       zhat_t = [sqrt(q); atan2(landmarks(2,j) - mu_t(2),
landmarks(1,j) - mu_t(1)) - theta_est];
       % line 13:
       H_t = [-(landmarks(1,j) - mu_t(1))/sqrt(q), -(landmarks(2,j) -
mu_t(2))/sqrt(q), 0;
              (landmarks(2,j) - mu_t(2))/q, -(landmarks(1,j) -
mu_t(1))/q, -1];
       % line 14:
       S_t = H_t*Sigma_t*H_t' + Q_t;
       % line 15:
       K_t = Sigma_t*H_t'/(S_t);
       % line 16:
       mu_t = mu_t + K_t*([range(i,j); bearing(i,j)] - zhat_t);
       % line 17:
       Sigma_t = (eye(3) - K_t*H_t)*Sigma_t;
```

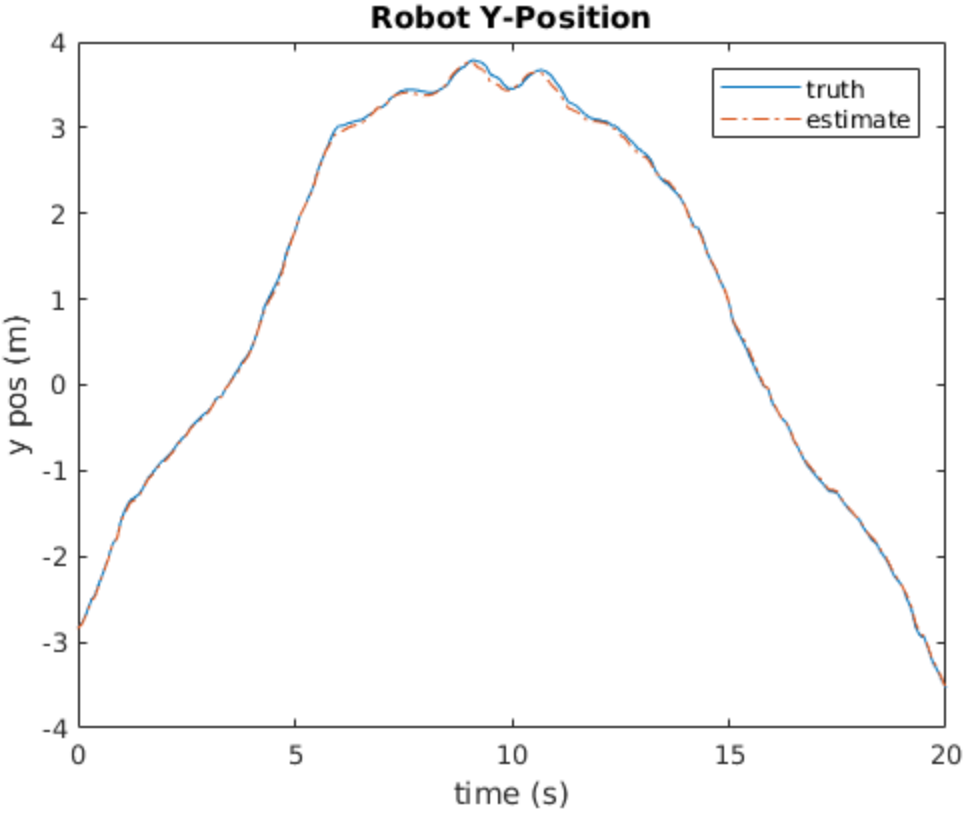
```
        end

        % save the estimate for plotting
        x_est(i) = mu_t(1);
        y_est(i) = mu_t(2);
        theta_estimated(i) = mu_t(3);

        Sigma_x(i) = Sigma_t(1,1);
        Sigma_y(i) = Sigma_t(2,2);
        Sigma_theta(i) = Sigma_t(3,3);

    end
    plot(x_true, y_true, x_est, y_est,'-.')
    legend('landmarks','robot body','robot front','truth','estimate')

    % state plots
    figure(2), clf
    plot(x_true, y_true, x_est, y_est,'-.')
    title('Robot Position')
    xlabel('x pos (m)')
    ylabel('y pos (m)')
    legend('truth','estimate')

    figure(3), clf
    plot(t, x_true, t, x_est,'-.')
    title('Robot X-Position')
    xlabel('time (s)')
    ylabel('x pos (m)')
    legend('truth','estimate')

    figure(4), clf
    plot(t, y_true, t, y_est,'-.')
    title('Robot Y-Position')
    xlabel('time (s)')
    ylabel('y pos (m)')
    legend('truth','estimate')

    figure(5), clf
    plot(t,theta_true,t,theta_estimated)
    title('Robot Heading vs Time')
    xlabel('time (s)')
    ylabel('heading (rad)')
    legend('truth','estimate')

    % error plots
    figure(6), clf
    plot(t,x_true - x_est,t,2*sqrt(Sigma_x),'r',t,-2*sqrt(Sigma_x),'r')
    title('Robot X-Position Error')
    xlabel('time (s)')
    ylabel('error (m)')
    legend('error','2-sigma bound')

    figure(7), clf
    plot(t,y_true - y_est,t,2*sqrt(Sigma_y),'r',t,-2*sqrt(Sigma_y),'r')
```

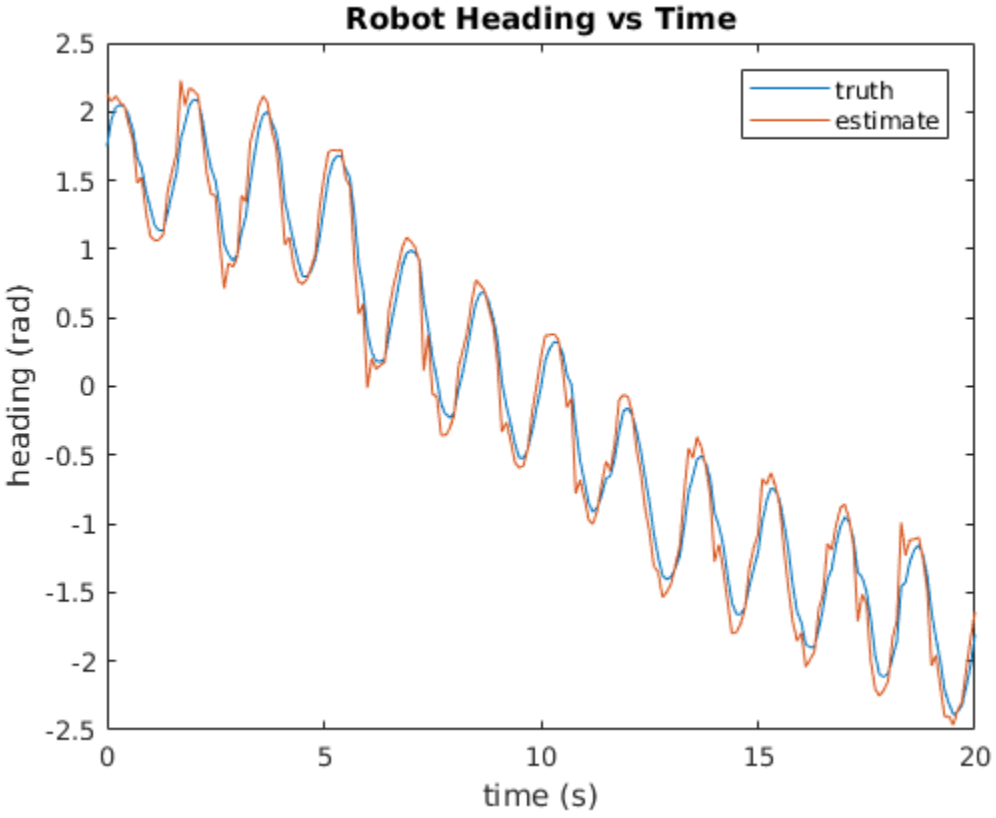
```
title('Robot Y-Position Error')
xlabel('time (s)')
ylabel('error (m)')
legend('error','2-sigma bound')

figure(8), clf
plot(t,theta_true -
 theta_estimated,t,2*sqrt(Sigma_theta),'r',t,-2*sqrt(Sigma_theta),'r')
title('Robot Heading Error')
xlabel('time (s)')
ylabel('error (rad)')
legend('error','2-sigma bound')
```

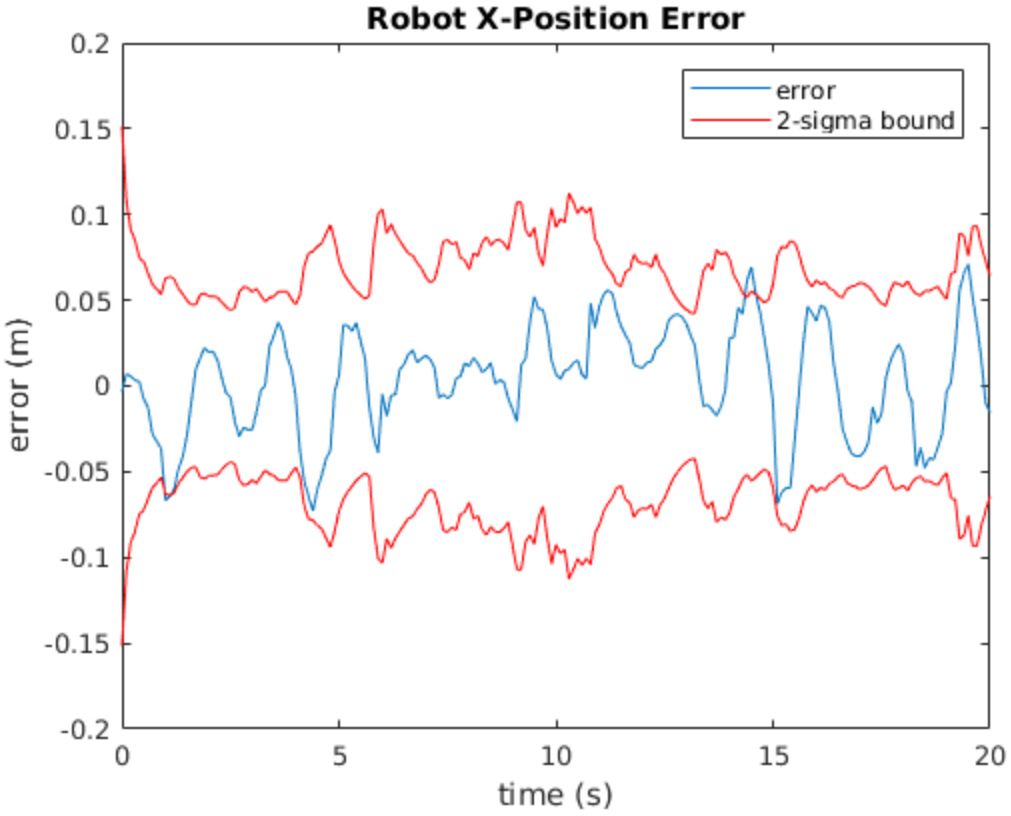


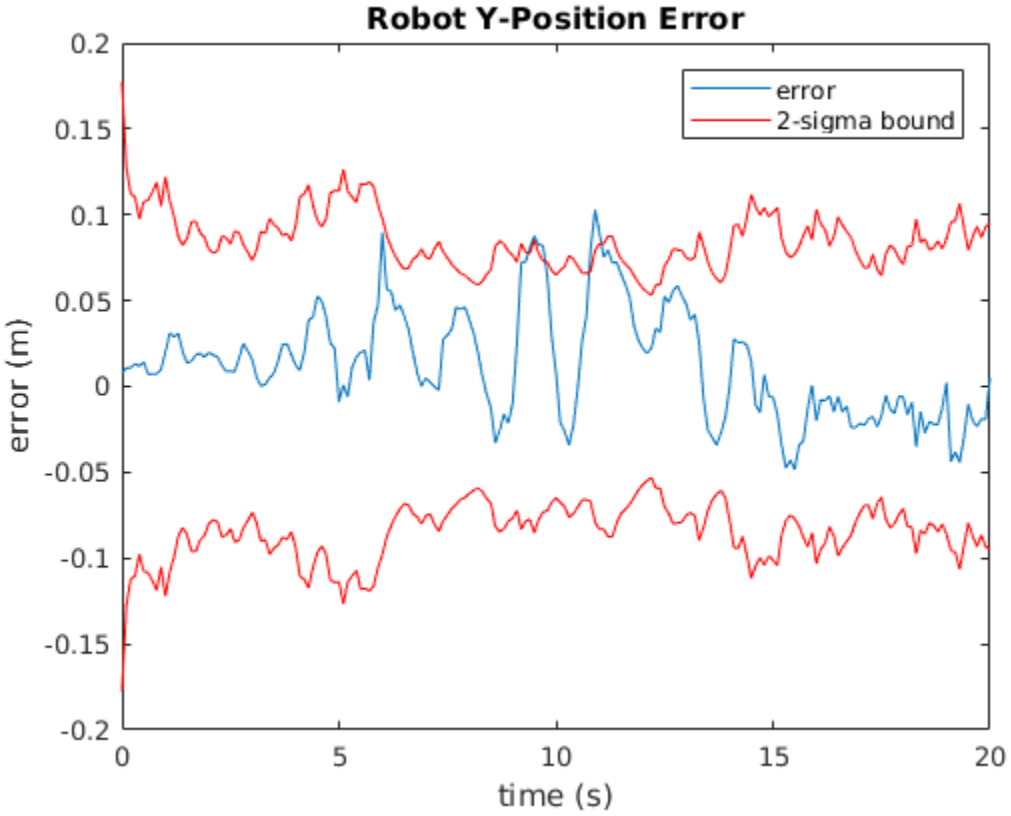

Robot Position
truth
estimate
y pos (m)
x pos (m)
4
3
2
1
0
-1
-2
-3
-4
-6
-5
-4
-3
-2
-1
0
1
2
3

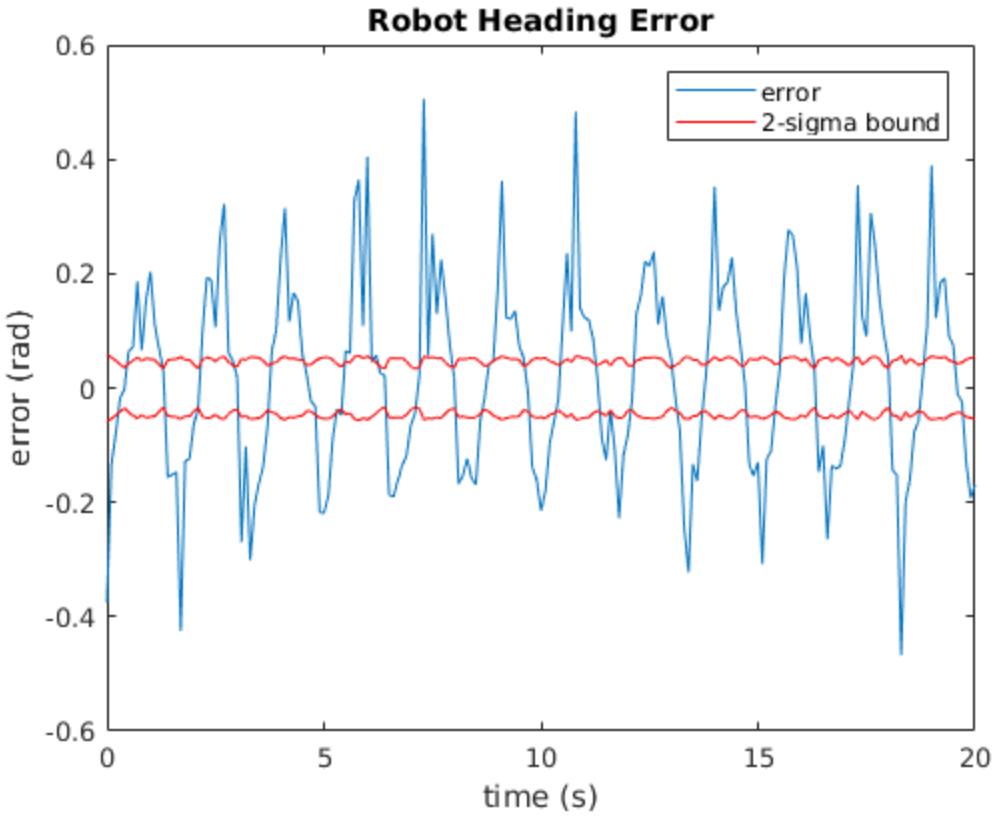

Robot X-Position
truth
estimate
x pos (m)
time (s)
3
2
1
0
-1
-2
-3
-4
-5
-6
0
5
10
15
20


Robot Y-Position
truth
estimate
y pos (m)
time (s)
4
3
2
1
0
-1
-2
-3
-4
0
5
10
15
20


Robot Heading vs Time
truth
estimate
heading (rad)
time (s)
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
5
10
15
20


Robot X-Position Error
error
2-sigma bound
error (m)
time (s)
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
5
10
15
20


Robot Y-Position Error
error
2-sigma bound
error (m)
time (s)
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
5
10
15
20



Published with MATLAB® R2017a