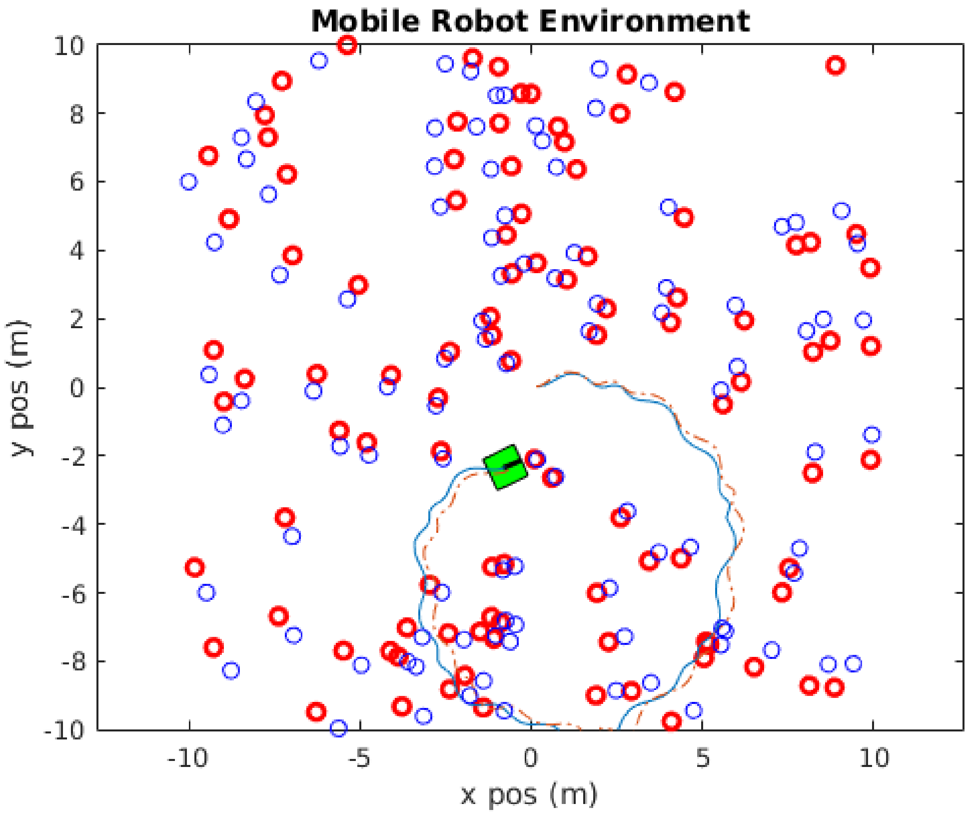
```
% Homework 6 Autonomous Two-wheele robot EKF-SLAM
% Jesse Wynn
% November 2, 2017

clc;
clear all;
close all;

% time params
Ts = 0.1;   % sec
t = 0:Ts:30;

% 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, 5, 3, 3, -2, -2, -7, -10;   % [x_positions;
 y_positions]
%              4, 8, -4, -9, -1, 9, 2, -7, -5, 3];
% landmarks = [6, -7, 6, 5, 3, 3, -2, -2, -7, -10, -7, -2, 9, -1, -5,
 -1, 10, 1, 9, -9;   % [x_positions; y_positions]
%              4, 8, -4, -9, -1, 9, 2, -7, -5, 3, 0, 6, 9, -2, -9, 9,
 0, 5, -9, -9];

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

min = -10;
max = 10;
num_landmarks = 100;
landmarks_x = zeros(1,num_landmarks);
landmarks_y = zeros(1,num_landmarks);
for i = 1:num_landmarks
    landmarks_x(i) = (max-min).*rand(1,1) + min;
    landmarks_y(i) = (max-min).*rand(1,1) + min;
end
landmarks = [landmarks_x; landmarks_y];
```

```
% initial state (always zero for EKF-SLAM)
mu_t = zeros(3 + num_landmarks * 2, 1);

% initial covariance
infinity = 10e10;
Sigma_t = [zeros(3), zeros(3, num_landmarks * 2);
           zeros(num_landmarks * 2, 3), infinity * eye(num_landmarks *
 2)];

% 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),num_landmarks);
bearing = zeros(length(t),num_landmarks);

% plot the first time
first = 0;
x = mu_t(1);
y = mu_t(2);
theta = mu_t(3);
drawRobot(x,y,theta,landmarks,first);
first = 1;

% loop through each time step
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
    ranges = getranges(x, y, landmarks, sigma_r);
    range(i,:) = ranges;

    bearings = getbearings(x, y, theta, landmarks, sigma_phi);
    bearing(i,:) = bearings;

    % Task 3: Implement the full EKF-SLAM algorithm

    % input velocities
    v_t = v_c(i);
    w_t = w_c(i);

    % line2
    F_x = [eye(3), zeros(3,num_landmarks * 2)];

    % line 3 mu_t = [x, y, theta, m1x, m1y, m2x, m2y,..., mNx,
 mNy]' (3 + 2*N)
    mu_t = mu_t + F_x' * [-(v_t/w_t)*sin(mu_t(3)) + (v_t/
w_t)*sin(mu_t(3) + w_t*Ts);
                          (v_t/w_t)*cos(mu_t(3)) - (v_t/
w_t)*cos(mu_t(3) + w_t*Ts);
                          w_t*Ts];

    % line 4
    G_t = eye(3 + num_landmarks * 2) + F_x' * [0, 0, -(v_t/
w_t)*cos(mu_t(3)) + (v_t/w_t)*cos(mu_t(3) + w_t*Ts);
                                               0, 0, -(v_t/
w_t)*sin(mu_t(3)) + (v_t/w_t)*sin(mu_t(3) + w_t*Ts);
                                               0, 0, 0] * F_x;

    % detour...need to compose R_t
    V_t = [(-sin(mu_t(3))+sin(mu_t(3) + w_t*Ts))/w_t,
 v_t*(sin(mu_t(3))-sin(mu_t(3) + w_t*Ts))/w_t^2 + v_t*cos(mu_t(3) +
 w_t*Ts)*Ts/w_t;
           (cos(mu_t(3)) - cos(mu_t(3) + w_t*Ts))/w_t, -
v_t*(cos(mu_t(3)) - cos(mu_t(3) + w_t*Ts))/w_t^2 + v_t*sin(mu_t(3) +
 w_t*Ts)*Ts/w_t;
           0, Ts];

    M_t = [alpha_1*v_t^2 + alpha_2*w_t^2, 0;
           0, alpha_3*v_t^2 + alpha_4*w_t^2];

    R_t = V_t * M_t * V_t';

    % line 5
    Sigma_t = G_t * Sigma_t * G_t' + F_x' * R_t * F_x;

    % line 6
    Q_t = [sigma_r^2, 0;
           0, sigma_phi^2];
```

```
    % line 7
    for j = 1:num_landmarks
        % ignore correspondance on line 8 for now

        % line 9
        % if we haven't seen this landmark before
        if mu_t(3 + 2*j-1) == 0
            % line 10
            mu_t(3 + 2*j-1) = mu_t(1) + (ranges(j)*cos(bearings(j) +
mu_t(3)));
            mu_t(3 + 2*j) = mu_t(2) + (ranges(j)*sin(bearings(j) +
mu_t(3)));

        end

        % line 12
        delta = [mu_t(2 + 2 * j) - mu_t(1);
                 mu_t(3 + 2 * j) - mu_t(2)];

        % line 13
        q = delta' * delta;

        % line 14
        zhat_t = [sqrt(q);
                  atan2(delta(2), delta(1)) - mu_t(3)];

        % line 15
        F_x_j = [eye(3), zeros(3, 2 * j - 2), zeros(3,2), zeros(3,
num_landmarks * 2 - 2 * j);
                 zeros(2,3), zeros(2, 2 * j - 2), eye(2), zeros(2,
num_landmarks * 2 - 2 * j)];

        % line 16
        H_t = (1/q) .* [-sqrt(q)*delta(1), -sqrt(q)*delta(2), 0,
sqrt(q)*delta(1), sqrt(q)*delta(2);
                        delta(2), -delta(1), -q, -delta(2), delta(1)] *
F_x_j;

        % line 17
        K_t = Sigma_t * H_t' / (H_t * Sigma_t * H_t' + Q_t);

        % detour...wrap heading measurement
        range_meas = ranges(j);
        bearing_meas = bearings(j);
        while bearing_meas - zhat_t(2) > pi
            bearing_meas = bearing_meas - 2 * pi;
        end
        while bearing_meas - zhat_t(2) < -pi
            bearing_meas = bearing_meas + 2 * pi;
        end

        mu_t = mu_t + K_t * ([range_meas; bearing_meas] - zhat_t);
```

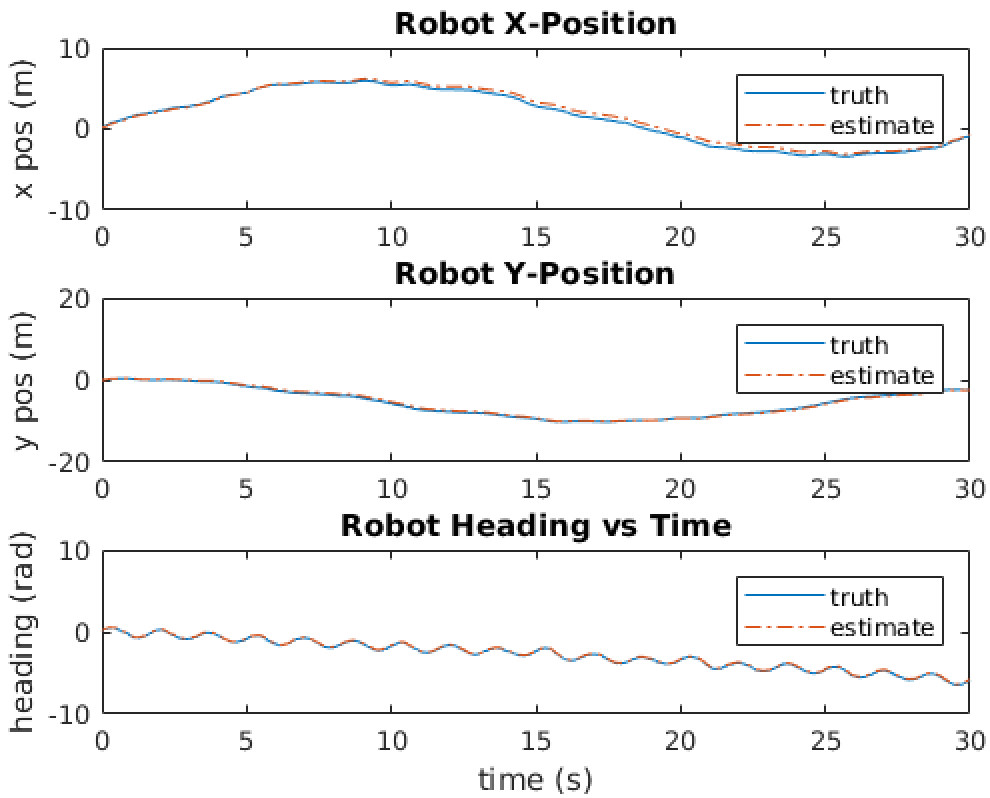
```
        % line 19
        Sigma_t = (eye(3 + num_landmarks * 2) - K_t * H_t)*Sigma_t;
    end

    % 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,'-.')
for i = 1:num_landmarks
    plot(mu_t(3+2*i-1), mu_t(3+2*i), 'ob')
end
```

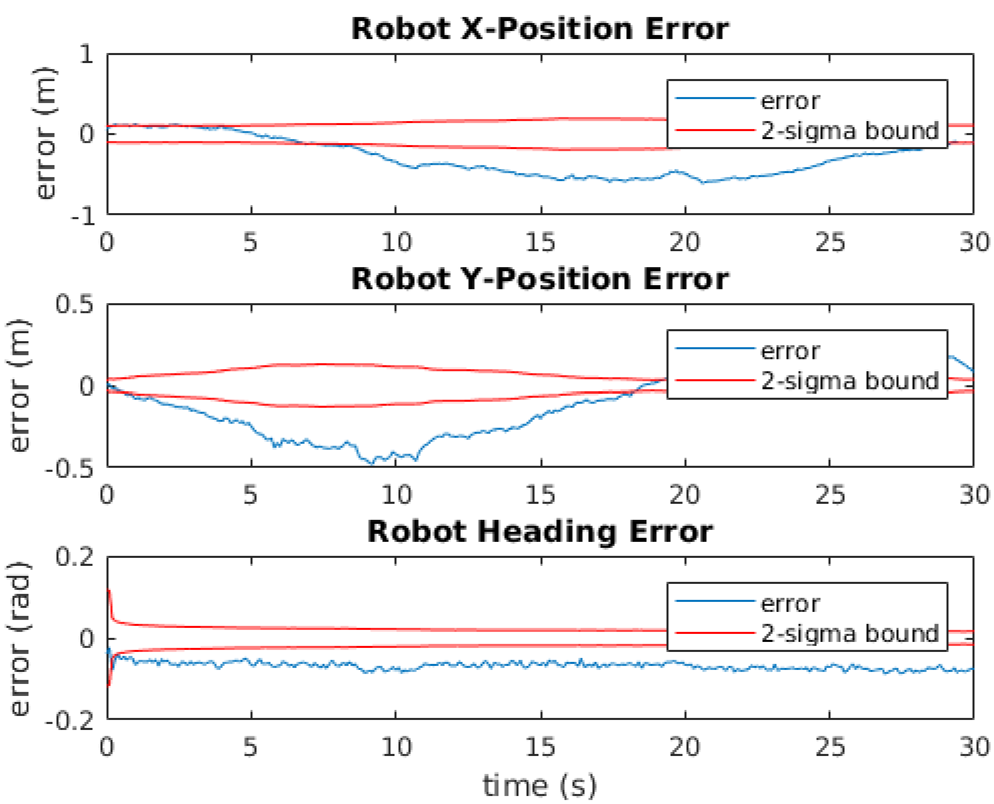


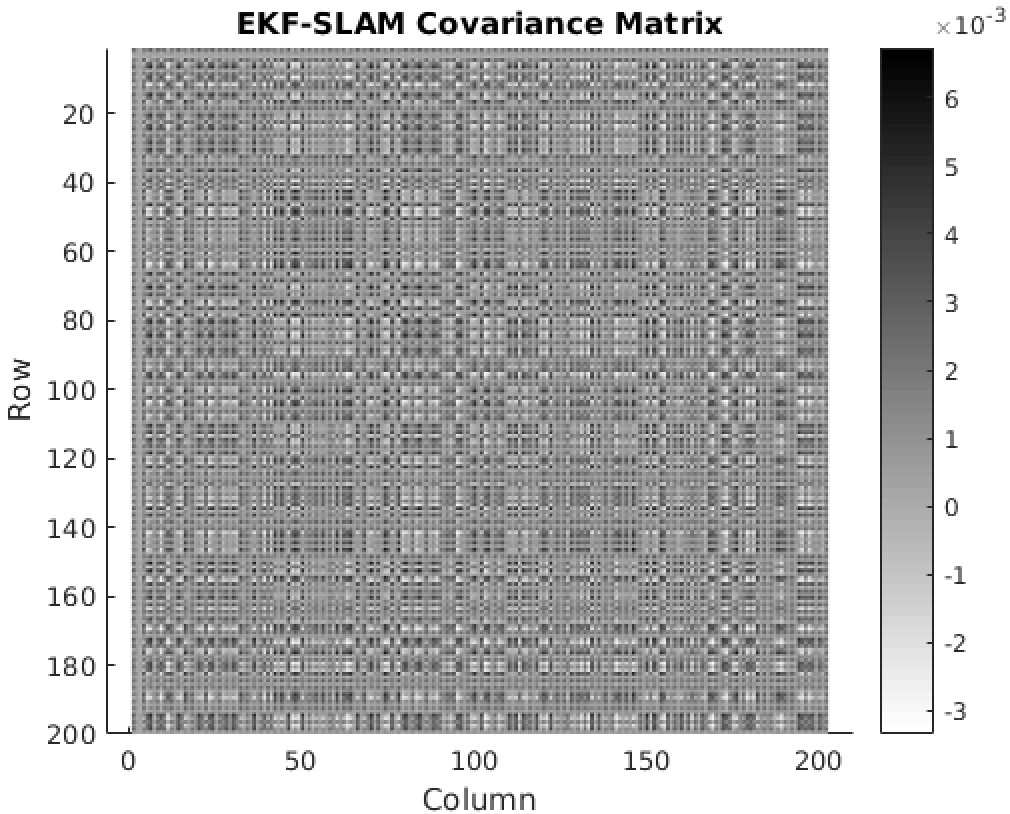
plots

```
figure(2), clf
subplot(3,1,1)
plot(t, x_true, t, x_est,'-.')
title('Robot X-Position')
```

```
ylabel('x pos (m)')
legend('truth','estimate')

subplot(3,1,2)
plot(t, y_true, t, y_est,'-.')
title('Robot Y-Position')
ylabel('y pos (m)')
legend('truth','estimate')

subplot(3,1,3)
plot(t,theta_true,t,theta_estimated,'-.')
title('Robot Heading vs Time')
xlabel('time (s)')
ylabel('heading (rad)')
legend('truth','estimate')

% error plots
figure(3), clf
subplot(3,1,1)
plot(t,x_true - x_est,t,2*sqrt(Sigma_x),'r',t,-2*sqrt(Sigma_x),'r')
title('Robot X-Position Error')
ylabel('error (m)')
legend('error','2-sigma bound')

subplot(3,1,2)
plot(t,y_true - y_est,t,2*sqrt(Sigma_y),'r',t,-2*sqrt(Sigma_y),'r')
title('Robot Y-Position Error')
ylabel('error (m)')
legend('error','2-sigma bound')

subplot(3,1,3)
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')

% plot final covariance matrix Sigma_t
figure(4), clf
% plot using surf
surf(Sigma_t,'LineStyle', 'none');
title('EKF-SLAM Covariance Matrix')
xlabel('Column')
ylabel('Row')
colorbar;
colormap(flipud(gray));
view(0,90)
axis([1, num_landmarks*2, 1, num_landmarks*2])
axis equal
axis ij
grid off
```


Robot X-Position
x pos (m)
truth
estimate
Robot Y-Position
y pos (m)
truth
estimate
Robot Heading vs Time
heading (rad)
truth
estimate
time (s)


Robot X-Position Error
error (m)
error
2-sigma bound
Robot Y-Position Error
error (m)
error
2-sigma bound
Robot Heading Error
error (rad)
error
2-sigma bound
time (s)



Published with MATLAB® R2017a