Appendices:

A.1. Code for Q1

```
% MEC
% Q1B
clear;

% Load data
load('calibration.mat');

% Measurement covariance
gps_diffs = [];

for index_y = 1:length(t_y)

    index_q = find(t_groundTruth == t_y(index_y));

    gps_meas = y(:,index_y);
    gt_state = q_groundTruth(1:2, index_q);

    gps_diffs = [gps_diffs, (gps_meas - gt_state)];

end

W = cov(gps_diffs');

% Process covariance
process_diffs = [];
T = 0.01;

for index_q = 1:(length(t_groundTruth) - 1)

    % Find state of robot at times k+1 and k
    q1_kp1 = q_groundTruth(1, index_q + 1);
    q1_k = q_groundTruth(1, index_q);
    q2_kp1 = q_groundTruth(2, index_q + 1);
    q2_k = q_groundTruth(2, index_q);
    q3_kp1 = q_groundTruth(3, index_q + 1);
    q3_k = q_groundTruth(3, index_q);

    % Find input vector at time k
    u1_k = u(1, index_q);
    u2_k = u(2, index_q);

    % Calculate process noise terms
    v1_from_q1 = ((q1_kp1 - q1_k) / (T * cos(q3_k))) - u1_k;
    v1_from_q2 = ((q2_kp1 - q2_k) / (T * sin(q3_k))) - u1_k;
    v2 = ((q3_kp1 - q3_k) / T) - u2_k;

    % Up to numerical errors, v1_from_q1 and v1_from_q2 are the same since
    % q and u are ground truth values
    v = [v1_from_q1; v2];

    process_diffs = [process_diffs, v];

end

V = cov(process_diffs');

% Q1C
```

```
clearvars -except V W

% Load data
load('kfData.mat');

% Initial parameters
T = 0.01;
q_hat = [0.355; -1.590; 0.682];
P = [25, 0, 0; 0, 25, 0; 0, 0, 0.154];

% Results storage
num_steps = length(t);
q_estimates = zeros(3, num_steps);
q_estimates(:, 1) = q_hat;

% EKF loop
for i = 1:(num_steps - 1)

    % Prediction step
    % Update mean
    q_estimates(1, i+1) = q_estimates(1, i) + T * u(1, i) * cos(q_estimates(3,
i));
    q_estimates(2, i+1) = q_estimates(2, i) + T * u(1, i) * sin(q_estimates(3,
i));
    q_estimates(3, i+1) = q_estimates(3, i) + T * u(2, i);

    % Update covariance
    F = [1, 0, -T * u(1, i) * sin(q_estimates(3, i)); 0, 1, T * u(1, i) *
cos(q_estimates(3, i)); 0, 0, 1];
    Gamma = [T * cos(q_estimates(3, i)), 0; T * sin(q_estimates(3, i)), 0; 0,
T];
    P = F * P * F' + Gamma * V * Gamma';

    % Update step (only if a new GPS measurement is received)
    if ismember(t(i+1), t_y)
        H = [1, 0, 0; 0, 1, 0];
        K = P * H' * inv(H * P * H' + W);
        y_meas = y(:, (i+1)/10);

        % The measurement equation is linear, so H = h
        q_estimates(:, i+1) = q_estimates(:, i+1) + K * (y_meas - H *
q_estimates(:, i+1));
        P = (eye(3) - K * H) * P;
    end
end

% Plotting
figure;
hold on;
plot(q_groundtruth(1, :), q_groundtruth(2, :), 'b-', 'DisplayName', 'Ground
Truth');
scatter(y(1, :), y(2, :), 'g', 'DisplayName', 'GPS Measurements');
plot(q_estimates(1, :), q_estimates(2, :), 'r-', 'DisplayName', 'EKF Estimate');

xlabel('x (m)');
ylabel('y (m)');
legend;
title('EKF Trajectory Estimation');
hold off;
```

A.4. Code for Q2

```
function M = pfTemplate()
% template and helper functions for 16-642 PS3 problem 2

rng(0); % initialize random number generator

b1 = [5,5]; % position of beacon 1
b2 = [15,5]; % position of beacon 2

% load pfData.mat
load('pfData.mat');

% initialize movie array
numSteps = length(t);
T = t(2) - t(1);
M(numSteps) = struct('cdata',[],'colormap',[]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         put particle filter initialization code here                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The grid is 20x10 with orientations from 0 to 2*pi
numParticles = 1000;
particles = [20 * rand(1, numParticles); 10 * rand(1, numParticles); 2 * pi *
rand(1, numParticles)];

% Process noise covariance V and measurement noise covariance W
V = [1, 0; 0, 0.5];
W = [0.75, 0; 0, 0.75];

% here is some code to plot the initial scene
figure(1)
plotParticles(particles); % particle cloud plotting helper function
hold on
plot([b1(1),b2(1)],[b1(2),b2(2)],'s',...
    'LineWidth',2,...
    'MarkerSize',10,...
    'MarkerEdgeColor','r',...
    'MarkerFaceColor',[0.5,0.5,0.5]);
drawRobot(q_groundTruth(:,1), 'cyan'); % robot drawing helper function
axis equal
axis([0 20 0 10])
M(1) = getframe; % capture current view as movie frame
pause
disp('hit return to continue')

% iterate particle filter in this loop
for k = 2:numSteps

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %               put particle filter prediction step here                    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for i = 1:numParticles
        % Generate process noise
        v = mvnrnd([0, 0], V);

        % Move particle
        particles(1, i) = particles(1, i) + T * (u(1, k) + v(1)) *
cos(particles(3, i));
```

```
        particles(2, i) = particles(2, i) + T * (u(1, k) + v(1)) *
sin(particles(3, i));
        particles(3, i) = particles(3, i) + T * (u(2, k) + v(2));
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                 put particle filter update step here                  %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % weight particles

    % Calculate expected measurement
    y_hat = zeros(2, numParticles);

    for i = 1:numParticles
        y_hat(1, i) = sqrt((particles(1, i) - b1(1))^2 + (particles(2, i) -
b1(2))^2);
        y_hat(2, i) = sqrt((particles(1, i) - b2(1))^2 + (particles(2, i) -
b2(2))^2);
    end

    % Calculate probability density of actual measurement
    weights = zeros(1, numParticles);

    for i = 1:numParticles
        weight1 = normpdf(y(1, k), y_hat(1, i), W(1, 1));
        weight2 = normpdf(y(2, k), y_hat(2, i), W(2, 2));
        weights(i) = weight1 * weight2;
    end

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

    % Cumulative weight vector
    CW = cumsum(weights);

    % resample particles
    new_particles = zeros(3, numParticles);

    for i = 1:numParticles
        % Generate random number and find smallest index in CW greater than
        % number
        z = rand();
        index = find(CW > rand, 1);

        % Update particle
        new_particles(:, i) = particles(:, index);
    end

    particles = new_particles;

    % Get robot pose estimate by taking the average of the particles
    avg_particle = mean(particles, 2);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % plot particle cloud, robot, robot estimate, and robot trajectory here %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Plot beacon location, particle cloud, robot ground truth pose
    clf()
    plotParticles(particles); % particle cloud plotting helper function
    hold on
```

```
    plot([b1(1),b2(1)],[b1(2),b2(2)],'s',...
        'LineWidth',2,...
        'MarkerSize',10,...
        'MarkerEdgeColor','r',...
        'MarkerFaceColor',[0.5,0.5,0.5]);
    drawRobot(q_groundTruth(:,k), 'cyan'); % robot drawing helper function
    axis equal
    axis([0 20 0 10])

    % Plot robot ground truth trajectory
    plot(q_groundTruth(1, 1:k), q_groundTruth(2, 1:k), 'k-', 'DisplayName',
'Ground Truth');

    % Plot robot pose estimate from particle cloud
    plot(avg_particle(1), avg_particle(2), 'r.', 'MarkerSize', 25);

    % capture current figure and pause
    M(k) = getframe; % capture current view as movie frame
    pause
    disp('hit return to continue')

end

% when you're ready, the following block of code will export the created
% movie to an mp4 file
videoOut = VideoWriter('result.mp4','MPEG-4');
videoOut.FrameRate=5;
open(videoOut);
for k=1:numSteps
  writeVideo(videoOut,M(k));
end
close(videoOut);


% helper function to plot a particle cloud
function plotParticles(particles)
plot(particles(1, :), particles(2, :), 'go')
line_length = 0.1;
quiver(particles(1, :), particles(2, :), line_length * cos(particles(3, :)),
line_length * sin(particles(3, :)))


% helper function to plot a differential drive robot
function drawRobot(pose, color)

% draws a SE2 robot at pose
x = pose(1);
y = pose(2);
th = pose(3);

% define robot shape
robot = [-1 .5 1 .5 -1 -1;
          1  1 0 -1  -1 1 ];
tmp = size(robot);
numPts = tmp(2);
% scale robot if desired
scale = 0.5;
robot = robot*scale;

% convert pose into SE2 matrix
H = [ cos(th)   -sin(th)  x;
      sin(th)    cos(th)  y;
```

```
      0          0        1];

% create robot in position
robotPose = H*[robot; ones(1,numPts)];

% plot robot
plot(robotPose(1,:),robotPose(2,:),'k','LineWidth',2);
rFill = fill(robotPose(1,:),robotPose(2,:), color);
alpha(rFill,.2); % make fill semi transparent
```