16-642

PS3

Boxiang Fu

boxiangf

11/06/2024

Q1.

Given $q = \begin{bmatrix} x_r \\ y_r \\ \theta \end{bmatrix}$, $u = \begin{bmatrix} v_f \\ \omega \end{bmatrix}$, $T = 0.01$ sec,

$$q[k+1] = \begin{bmatrix} q_1[k] + T(u_1[k] + v_1[k])\cos q_3[k] \\ q_2[k] + T(u_1[k] + v_1[k])\sin q_3[k] \\ q_3[k] + T(u_2[k] + v_2[k]) \end{bmatrix}$$

$$y[k] = \begin{bmatrix} q_1[k] \\ q_2[k] \end{bmatrix} + \omega[k]$$

a. The linearization of the system is given as

$$q[k+1] \approx F(q[k], u[k]) \cdot q[k] + G(q[k]) \cdot u[k] + P(q[k]) \cdot v[k]$$

where

$$F(q[k], u[k]) = \left.\frac{\partial f}{\partial q}\right|_{q=q[k],\, u=u[k],\, v=0}$$

$$= \left.\begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \frac{\partial f_1}{\partial q_3} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} & \frac{\partial f_2}{\partial q_3} \\ \frac{\partial f_3}{\partial q_1} & \frac{\partial f_3}{\partial q_2} & \frac{\partial f_3}{\partial q_3} \end{bmatrix}\right|_{q=q[k],\, u=u[k],\, v=0}$$

$$= \begin{bmatrix} 1 & 0 & -T(u_1[k]+v_1[k])\sin q_3[k] \\ 0 & 1 & T(u_1[k]+v_1[k])\cos q_3[k] \\ 0 & 0 & 1 \end{bmatrix}\Bigg|_{q=q[k],\, u=u[k],\, v=0}$$

$$\therefore F = \begin{bmatrix} 1 & 0 & -T(u_1[k])\sin q_3[k] \\ 0 & 1 & T(u_1[k])\cos q_3[k] \\ 0 & 0 & 1 \end{bmatrix}$$

$$G(q[k]) = \frac{\partial f}{\partial u}\Bigg|_{q=q[k],\, u=u[k],\, v=0}$$

$$= \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} \end{bmatrix}\Bigg|_{q=q[k],\, u=u[k],\, v=0}$$

$$\therefore G = \begin{bmatrix} T\cos q_3[k] & 0 \\ T\sin q_3[k] & 0 \\ 0 & T \end{bmatrix}$$

$$P(q[k]) = \frac{\partial f}{\partial v}\Bigg|_{q=q[k],\, u=u[k],\, v=0}$$

$$= \begin{bmatrix} \frac{\partial f_1}{\partial v_1} & \frac{\partial f_1}{\partial v_2} \\ \frac{\partial f_2}{\partial v_1} & \frac{\partial f_2}{\partial v_2} \\ \frac{\partial f_3}{\partial v_1} & \frac{\partial f_3}{\partial v_2} \end{bmatrix}\Bigg|_{q=q[k],\, u=u[k],\, v=0}$$

$$\therefore P = \begin{bmatrix} T\cos q_3[k] & 0 \\ T\sin q_3[k] & 0 \\ 0 & T \end{bmatrix}$$

b. The calculated measurement covariance is:

$$W = \begin{bmatrix} 1.8817 & 0.0632 \\ 0.0632 & 2.1384 \end{bmatrix}$$

The MATLAB code is included in the appendicies.

The computation is from the following given output equation:

$$y[k] = \begin{bmatrix} q_1[k] \\ q_2[k] \end{bmatrix} + w[k]$$

We know $y[k]$ from the k'th column of matrix y, we also know $q_1[k]$ and $q_2[k]$ from the 10k'th column of q-groundtruth. Rearranging, we obtain

$$\begin{bmatrix} w_1[k] \\ w_2[k] \end{bmatrix} = \begin{bmatrix} y_1[k] - q_1[k] \\ y_2[k] - q_2[k] \end{bmatrix}$$

for each time step k.

We run the function cov on w^T after adding up all timesteps to obtain the covarlance matrix W (we use w^T since the cov function speeifies observations down its rows).

The calculated process covariance is:

$$V = \begin{bmatrix} 0.2591 & 0.0010 \\ 0.0010 & 0.0625 \end{bmatrix}$$

The MATLAB code is included in the appendicies.

The computation is from the following equation of motion:

$$q[k+1] = \begin{bmatrix} q_1[k] + T(u_1[k]+v_1[k])\cos q_3[k] \\ q_2[k] + T(u_1[k]+v_1[k])\sin q_3[k] \\ q_3[k] + T(u_2[k]+v_2[k]) \end{bmatrix}$$

Rearranging in terms of v, we have:

$$v_1[k] = \frac{q_1[k+1]-q_1[k]}{T\cos q_3[k]} - u_1[k]$$

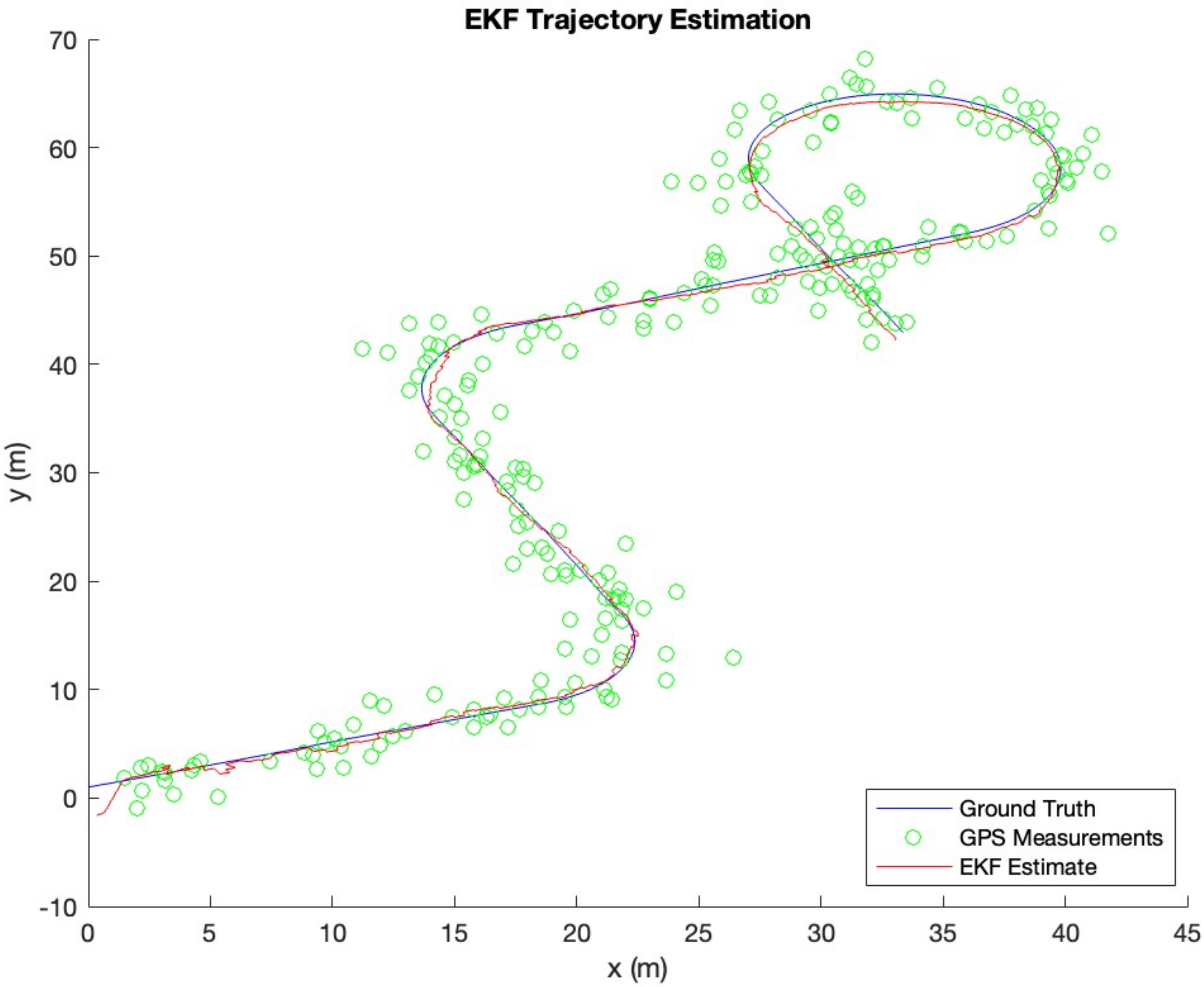
$$v_1[k] = \frac{q_2[k+1]-q_2[k]}{T\sin q_3[k]} - u_1[k]$$

$$v_2[k] = \frac{q_3[k+1]-q_3[k]}{T} - u_2[k]$$

For timesteps k and $k+1$, we know the RHS values from matrices q-groundtruth and u. We collect $v = \begin{bmatrix} v_1[k] \\ v_2[k] \end{bmatrix}$ for timesteps $k \in [0, K-1]$, where K = total time steps.

We run the function cov on v^T after adding up all timesteps to obtain the covariance matrix V.

c. The EKF plot output is displayed below.
The code is in the appendices.



Q2. The code for this section is included in the appendicles.

Some liberties chosen when creating the filter include:

1. The number of particles

   numParticles=1000

   The reason for this is because this is the recommended number from Lec 13 Slide 24

2. Process noise covariance

$$V = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

This takes inspiration from Q1 in that the directional noise is generally larger than angular noise.

A direction noise variance of 1 is chosen to allow the particles to explore the environment, but also not disperse too far so that the robot remains fairly localized.

An angular noise variance of 0.5 is chosen to allow the particls to explore different directions, but also not chage its heading too dramatically.

Overall, after tuning, $V = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$ seems to perform relatively well on the dataset.

The off-diagonals are 0 as the noise measurements are uncorrelated.

3. Measurement noise covariance

$$W = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.75 \end{bmatrix}$$

Too large diagonal values will cause the particle cloud to disperse will too small will cause the particles to update sluggishly. 0.75 seems to provide relatively well performance.

We also set the variances equal as it is reasonable to assume the GPS measurements to the beacons are similarly noisy.

The off-diagonals are 0 as the noise measurements are uncorrelated.

4. Robot pose estimate is taken as the average location of the particle cloud. This estimate starts off poorly, but after the particles converge, the pose estimate is relatively accurate.

A .mp4 movie showing the plot is included as an zip file.

Q3.

Let $\{S\}$ denote the event of containing hazardous structures
$\{W_s\}$ denote the event of sonar sensor detection
$\{C\}$ denote the event of strong currents
$\{W_c\}$ denote the event of current flow sensor detection
$\{W\} = \{W_s\} \cup \{W_c\}$ denote the event of a sensor detection

Let $'$ denote the complement of an event

From the question, we know the following:

$P(S) = 0.35$ $\quad$ $P(S') = 0.65$

$P(C) = 0.15$ $\quad$ $P(C') = 0.85$

$P(W_s|S) = 0.9$ $\quad$ $P(W_s'|S) = 0.1$

$P(W_s|S') = 0.02$ $\quad$ $P(W_s'|S') = 0.98$

$P(W_c|C) = 0.45$ $\quad$ $P(W_c'|C) = 0.55$

$P(W_c|C') = 0.04$ $\quad$ $P(W_c'|C') = 0.96$

$\{S\}$ and $\{C\}$ are mutually exclusive

$P(W_c)$ and $P(W_s)$ are independent

a. We calculate $P(C|W) = \dfrac{P(W|C)P(C)}{P(W)}$

For $P(W|C)$:

$$P(W|C) = P(W_c \cup W_s|C)$$
$$= P(W_c|C) + P(W_s|C) - P(W_c \cap W_s|C)$$

$$P(W|C) = P(W_c|C) + P(W_s|S') - P(W_c|C)P(W_s|C)$$
$$= P(W_c|C) + P(W_s|S') - P(W_c|C)\,P(W_s|S')$$
$$= 0.45 + 0.02 - 0.45 \times 0.02$$
$$= 0.461$$

For $P(W)$:

$$P(W) = P(W_c \cup W_s)$$
$$= P(W_c) + P(W_s) - P(W_c \cap W_s)$$
$$= P(W_c) + P(W_s) - P(W_c)P(W_s)$$

By Law of Total Probability,

$$P(W_c) = P(W_c|C)P(C) + P(W_c|C')P(C')$$
$$= 0.45 \times 0.15 + 0.04 \times 0.85$$
$$= 0.1015$$

$$P(W_s) = P(W_s|S)P(S) + P(W_s|S')P(S')$$
$$= 0.9 \times 0.35 + 0.02 \times 0.65$$
$$= 0.328$$

So $P(W) = 0.1015 + 0.328 - 0.1015 \times 0.328$
$= 0.396208$

So $P(C|W) = \dfrac{0.461 \times 0.15}{0.396208}$
$= 0.1745$
$= 17.45\%$

b. We want

$P(\{\text{At least 1 } W_c \text{ after } n \text{ trials}\} \mid c)$

$= 1 - P(\{\text{No } W_c \text{ in } n \text{ trials}\} \mid c)$

$= 1 - P(W_c' \mid c)^n$

$= 1 - 0.55^n$

Equaling this to $P(W_s \mid S) = 0.9$, we obtain

$0.9 = 1 - 0.55^n$

$0.55^n = 0.1$

$n = \dfrac{\ln(0.1)}{\ln(0.55)}$

≈ 3.85

$= 4$ (round up)

∴ We need 4 measurements

c. For the current flow sensor to have the detection reliability as the sonar sensor, we need the average noise over the 4 measurements to be $N(0,7)$, where $\sigma_s^2 = 7$.

For averages, $\bar{\sigma}_c^2 = \dfrac{\sigma_c^2}{n}$

Equation $\bar{\sigma}_c^2 = \sigma_s^2$, we obtain

$\dfrac{\sigma_c^2}{4} = 7$

$\Rightarrow \sigma_c^2 = 28$

So the current flow sensor noise can maximally be $N(0,28)$

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));
```

```matlab
        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
```