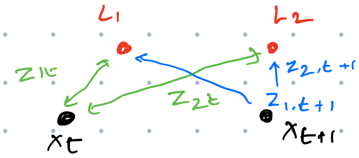


EECE 5550 — HW 02

Problem 1: Extended Kalman Filter

In this exercise you will apply the extended Kalman filter to solve a basic 2D robot localization problem. To that end, we will consider a ground robot that is navigating in a 2D environment containing two landmarks at *known positions* $l_i = [l_x^i, l_y^i] \in \mathbb{R}^2$ for $i \in \{1, 2\}$.

(a) We will model our robot as a simple point moving in the plane (i.e. ignoring its orientation), and assume that it is possible to directly control its velocity. Denoting its position and velocity at time t by $p(t) = (p_x(t), p_y(t)) \in \mathbb{R}^2$ and $v(t) = (v_x(t), v_y(t)) \in \mathbb{R}^2$ (respectively), write down the discrete-time state transition model that predicts the robot's position $p(t + \Delta t)$ at time $t + \Delta t$ given its position $p(t)$ at time t, assuming constant velocity and additive mean-zero Gaussian noise with covariance $R \in \mathbb{S}^2_+$ on the final position.

(b) At each timestep, we assume that the robot is able to collect a noisy measurement of the *range* to each of the two landmarks in the environment, subject to additive mean-zero Gaussian noise with covariance $Q \in \mathbb{S}^2_+$. Write down the corresponding measurement model for the resulting measurements.

(c) As we saw in class, implementing an extended Kalman filter requires linearizing any nonlinear state transition or measurement models about the current state estimate, and using the resulting linear approximations in the standard Kalman filter equations. Derive the Jacobians for the state transition and measurement models in parts (a) and (b).

(d) Using your results from parts (a)–(c), derive the corresponding state propagation and measurement update equations for an extended Kalman filter to solve this localization problem.

(e) Using the EKF that you designed in parts (a)–(d), write a script that applies it to solve the following simulated localization scenario:

- The landmarks are located at $l_1 = (5, 5)$ and $l_2 = (-5, 5)$.
- The covariances for the state transition and measurement models are given by $R = .1I_2$, $Q = .5I_2$ for all $t \in [0, 40]$.
- The robot's initial belief over its position is $p(0) \sim \mathcal{N}(0, I_2)$.
- The timestep interval is given by $\Delta t = .5$, and the sequence of velocity commands is given by:
  - $v = (1, 0)$ for all $0 \le t \le 10$,
  - $v = (0, -1)$ for all $10 < t \le 20$,
  - $v = (-1, 0)$ for all $20 < t \le 30$,

a) $P(t+\Delta t) = f[P(t), V] + \varepsilon_t$ where $P(t) = \begin{bmatrix} P_x(t) \\ P_y(t) \end{bmatrix}$

$V = \begin{bmatrix} V_x \\ V_y \end{bmatrix}$

$$\begin{bmatrix} P_x(t+\Delta t) \\ P_y(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_x(t) \\ P_y(t) \end{bmatrix} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} + \varepsilon_t$$

where $\varepsilon_t = (0, R_t) \rightarrow R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$

b) $z(t) = h(x_t) + \delta_t$ where $t \sim N(0, \theta_t)$

$$\Rightarrow \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} \sqrt{(P_x(t) - l_{1x})^2 + (P_y(t) - l_{1y})^2} \\ \sqrt{(P_x(t) - l_{2x})^2 + (P_y(t) - l_{2y})^2} \end{bmatrix} + \delta_t$$

where $\delta_t \sim N(0, \theta_t)$, $\theta_t = \begin{bmatrix} \theta_{11} & \theta_{12} \\ \theta_{21} & \theta_{22} \end{bmatrix}$

c) Jacobian for state transition model

$$P(t+\Delta t) = \begin{matrix} P_x + V_x \Delta t \leftarrow f_1 \\ P_y + V_y \Delta t \leftarrow f_2 \end{matrix}$$

$$\frac{\partial f_1}{\partial P_x} = 1 \quad \frac{\partial f_1}{\partial P_y} = 0 \quad \Rightarrow J(P(t+\Delta t)) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\frac{\partial f_2}{\partial P_x} = 0 \quad \frac{\partial f_2}{\partial P_y} = 1$$

Jacobian for measurement model

$$z(t) = \begin{matrix} \left((P_x(t) - l_{1x})^2 + (P_y(t) - l_{1y})^2\right)^{1/2} \leftarrow f_1 \\ \left((P_x(t) - l_{2x})^2 + (P_y(t) - l_{2y})^2\right)^{1/2} \leftarrow f_2 \end{matrix}$$

$$\frac{\partial f_1}{\partial P_x} = \frac{2(P_x - l_{1x})}{2\sqrt{(P_x(t) - l_{1x})^2 + (P_y(t) - l_{1y})^2}} = \frac{P_x - l_{1x}}{z_1}$$

$$\frac{\partial f_1}{\partial P_y} = \frac{P_y - l_{1y}}{z_1}$$

where $z_1 = \sqrt{(P_x(t) - l_{1x})^2 + (P_y(t) - l_{1y})^2}$

$$\frac{\partial f_2}{\partial P_x} = \frac{2(P_x - l_{2x})}{2\sqrt{(P_x(t)-l_2x)^2+(P_y(t)-l_2y)^2}} = \frac{P_x - l_2x}{z_2}$$

$$\frac{\partial f_2}{\partial P_y} = \frac{P_y - l_2y}{z_2}$$

$$\Rightarrow \boxed{H_t = \begin{bmatrix} \frac{P_x - l_1x}{z_1} & \frac{P_x - l_1y}{z_1} \\ \frac{P_x - l_2x}{z_2} & \frac{P_y - l_2y}{z_2} \end{bmatrix}}$$

$$G_T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

d) Algorithm : Predict

$$\mu_{t+1} = g_t(x_t, u_t) + \varepsilon_t \Rightarrow \begin{bmatrix} P_x(t+\Delta t) \\ P_y(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} P_x(t) \\ P_y(t) \end{bmatrix} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}\begin{bmatrix} V_x \\ V_y \end{bmatrix} + \varepsilon_t$$

where $\varepsilon_t = (0, R_t) \rightarrow R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$

: $x_t = N(\mu_t, \Sigma_t)$

$\Sigma_{t+1} = G_t \Sigma_t G_T^T + R_t$ and $G_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (Jacobian from process model)

Update step : to update current belief $\rightarrow$ plugging $x_t(\hat{\mu}, \hat{\Sigma}) = x_{t+1}$ (from update step)

$x_t = \hat{\mu}_t + K_t(z - h_t(\hat{\mu}_t))$, $\Sigma_t = (I - K_t H_t)\hat{\Sigma}_t$ ← from update step

from update step

and kalman gain $K_t = \hat{\Sigma}_t H_t^T (H_t \hat{\Sigma}_t H_t^T + Q_t)^{-1}$

where H_t is $\begin{bmatrix} \frac{P_x - l_1x}{z_1} & \frac{P_x - l_1y}{z_1} \\ \frac{P_x - l_2x}{z_2} & \frac{P_y - l_2y}{z_2} \end{bmatrix}$

$(x_t, \Sigma_t) \rightarrow$ output of update step is kalman filtered belief.

$\rightarrow$ Plug (x_t, Σ_t) into predict step and continue loop for num_iter.

```
% Landmark positions
L1 = [5;5];
L2 = [-5;5];
% State Model Parameters
g_t = [1,0;0,1]
%Measurement Model parameters
z_t = [];

%Covariance
Rt = [0.1,0;0,0.1];
%Measurement Covariance
Qt = [0.5,0;0,0.5];

%Belief mean and sigma

mu=[];
sig=[];

mu_0 = [0;0] + randn(2,1);
sig_0 = [1,0;0,1];

sig_init = [];
sig_next = [];

% Timestep
dt = 0.5;
%Blank Velocity Vector
V = []

%Jacobian for state model
G_t = [1,0;0,1]

%initialize
sig_init = sig_0;
mu = mu_0

mu_list =[]; % list of beliefss
sig_list = []; %list of standard deviations for every belief
true_pose = []; %list of true poses based on motion model and noise
% start loop here
x_t = [0;0]
n = 1;
for t=0:dt:40
    if t <= 10
        V = [1; 0];
    elseif t <= 20
        V = [0; -1];
    elseif t <= 30
        V = [-1; 0];
    else
        V = [0; 1];
    end
```

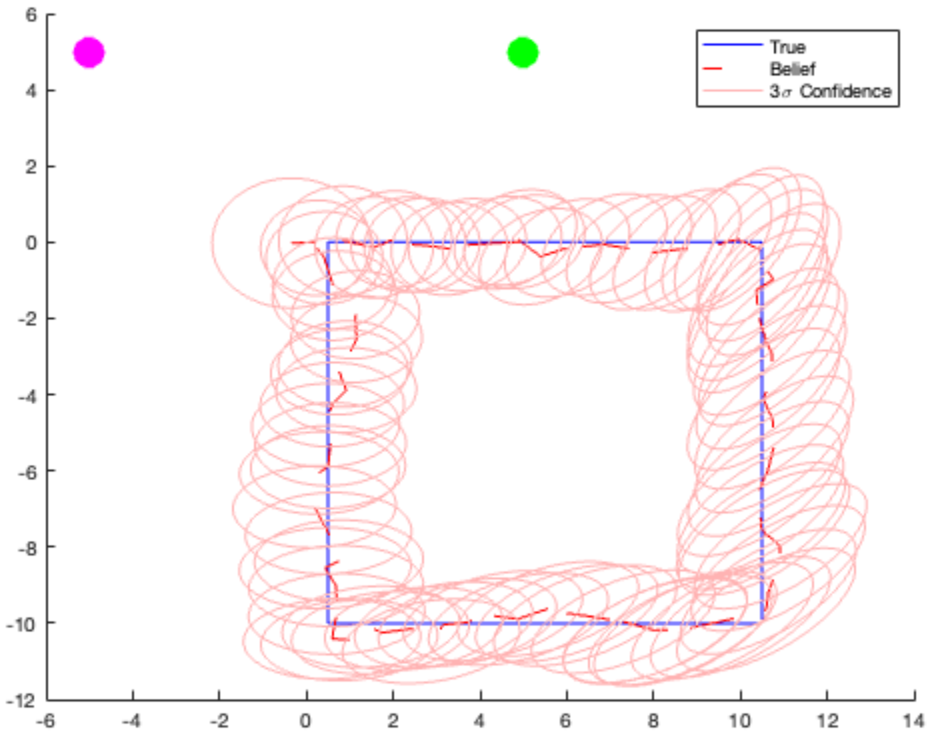
```
    %Predict Step
    init_pose = mu;
    next_pose = (g_t * init_pose) + dt*V; % updating mu with velocity model
    sig_next = G_t*sig_init*G_t' + Rt; % updating sigma with jacobian approach

    x_t = x_t + dt*V; %+ ((Rt)*randn(2, 1)); % calculating true pose with
measurement noise
    true_pose(:, n) = x_t; % adding true pose to list of historical poses
indexed by n

    %Update Step
    z1 = norm(x_t - L1) + (Qt(1,1))*randn();
    z2 = norm(x_t - L2) + (Qt(2,2))*randn();
    z_t = [z1; z2];

    % Measurement Jacobian
    diff1 = next_pose - L1;
    d1 = norm(diff1);
    diff2 = next_pose - L2;
    d2 = norm(diff2);
    H_t = [diff1'/d1; diff2'/d2]; % calculated jacobian of measurement model

    h_pred = [norm(diff1); norm(diff2)];

    % calculating Kalman gain
    Kt = sig_next * H_t' / (H_t * sig_next * H_t' + Qt);

    % Update Mu and Sigma
    mu = next_pose + Kt * (z_t - h_pred);
    sig_init = (eye(2) - Kt * H_t) * sig_next;

    mu_list(:,n) = mu;
    sig_list(:,:,n)=sig_init;
    n=n+1;
end

hold on
% Plot true trajectory
plot(true_pose(1, :), true_pose(2, :), 'b-');

% Plot estimated trajectory
plot(mu_list(1, :), mu_list(2, :), 'r--');

plot(L1(1), L1(2), 'go', 'MarkerSize', 15, 'MarkerFaceColor', 'g',
'DisplayName', 'Landmark 1');
plot(L2(1), L2(2), 'mo', 'MarkerSize', 15, 'MarkerFaceColor', 'm',
'DisplayName', 'Landmark 2');
legend('True','Belief')

% Plot 3-sigma confidence bounds (ellipses at each timestep)
for k = 1:size(mu_list, 2)
    % Extract mean and covariance at timestep k
    mu_k = mu_list(:, k);
```

```
    sig_k = sig_list(:, :, k);

    % Compute eigenvalues and eigenvectors of covariance
    [V_eig, D] = eig(sig_k);

    % Create points on a circle
    theta = linspace(0, 2*pi, 100);

    % Create ellipse
    ellipse = 3 * [sqrt(D(1,1)) * cos(theta);
                   sqrt(D(2,2)) * sin(theta)];

    ellipse = V_eig * ellipse;

    % Translate ellipse to mean position
    ellipse(1, :) = ellipse(1, :) + mu_k(1);
    ellipse(2, :) = ellipse(2, :) + mu_k(2);

    % Plot ellipse (light red color)
    if k == 1
        plot(ellipse(1, :), ellipse(2, :), 'r-', 'LineWidth', 0.5, ...
             'Color', [1 0.7 0.7], 'DisplayName', '3\sigma Confidence');
    else

        plot(ellipse(1, :), ellipse(2, :), 'r-', 'LineWidth', 0.5, ...
             'Color', [1 0.7 0.7], 'HandleVisibility', 'off');
    end
end
```

```
g_t =

     1     0
     0     1


V =

     []


G_t =

     1     0
     0     1


mu =

   -0.1789
   -0.5106


x_t =
```

0
0

Plot showing True location of Robot pose, belief pose from ERF, and the 3σ confidence intervals.



Published with MATLAB® R2024b

Problem 02

```
t0 = [0;0;0]; % setting initial transformation
R0 = [1,0,0;0,1,0;0,0,1]; % setting initial rotation
X = load('pclX.txt');
Y = load("pclY.txt");
n_x = size(X,1); % calculating length of X points
n_y = size(Y,1); % calculating length of Y points
d_max = 0.25; % setting maximum distance threshold

num_iter = 30;

% assigning initial transformation matrices
t=t0;
R=R0;


for i = 1:1:num_iter %setting num_iter here
    disp(i)
C = []; % creating blank correspondence matrix

    for i = 1:1:n_x % looping through every X point in list
        Xi = X(i,:)'; % extracting points from X Matrix and making it a
column vector
        Xi_trans = R*Xi + t; % transforming point X to its image under T

        % Reset min_dist for each new point in X + create default condition
        min_dist = inf;
        best_j = -1; % index of Y that best matches X_i

        for j = 1:1:n_y %  looping through Y
            Yj = Y(j,:)'; % extracting points from Y matrix and making it a
column vector
            distance = norm(Yj - Xi_trans); % Euclidean distance between Y_j
and transformed X_i

            if distance < min_dist
                min_dist = distance; % updating default condition to find the
point Y that best corresponds to X_i
                best_j = j; % updating default condition index to find the Y
that best corresponds to X_i
            end
        end

        % Checking if min point < d_max
        if min_dist < d_max % parent ICP condition
            C = [C; i, best_j]; % Add correspondence as new row where the
elements are the indices of the parent X and Y matrices that correspond
        end
    end

    % Horns Method

    K = size(C, 1) ;     % finding the dimension of the correspondence matix
```

```
for calcilating weighted means

    X_corr = X(C(:,1), :);     % extracting correlated X and Y points per C
matrix
    Y_corr = Y(C(:,2), :);

    % Centroids:
    x_bar = mean(X_corr, 1)';  % finding centroid of X dataset as a column
vector
    y_bar = mean(Y_corr, 1)';      % finding centroid of Y dataset as a
column vector

    %Calculate deviations of each point from the centroid of its pointcloud:

    X_prime = (X_corr - x_bar')';      %re-transposing X_bar/Y_Bar to find
the deviation from the points before transposing into column vector
    Y_prime = (Y_corr - y_bar')';

    %cross covariance matrix
    W = (Y_prime * X_prime') / K ;

    %symmetric value decomposition

    [U,S,V] = svd(W);

    %Constructing optimal rotation.
    diag_matrix = eye(3);
    diag_matrix(3, 3) = det(U * V');

    %finding rotation matrix
    R_hat = U * diag_matrix * V';
    %finding translation component
    t_hat = y_bar - R_hat*x_bar;

    %updating parent rotation and translation matrices ahead of next
    %iteration

    R=R_hat;
    t=t_hat;


end

%calculating RMSE in 2 steps.

%STEP 1 - SSE of Corresponded points
SSE = 0;
for k = 1:1:K %looping over length of C

    i = C(k, 1);  % indices of X
    j = C(k, 2);  % indices of Y

    x_trans = R * X(i,:)' + t; %transformed X Coorindates with final R and t
    error_vec = Y(j,:)' - x_trans; % calculating error between the
```

```
transformed X and Y
    SSE = SSE + norm(error_vec)^2;
end

RMSE = sqrt(SSE / K);
disp(RMSE)
```

```
25

26

27

28

29

30

0.0090
```

RMSE for point correspondences

Published with MATLAB® R2024b

```
X = load("pclX.txt");
Y= load("pclY.txt");

transformed=[]

% R Matrix from prior script
R = [0.951266006558687    -0.150430576214007    -0.269190687999811;
    0.223236284835894    0.938163601035720    0.264602756645424;
    0.21274056006920    -0.311800736740008    0.926024705215704]
```

$\hat{R}$

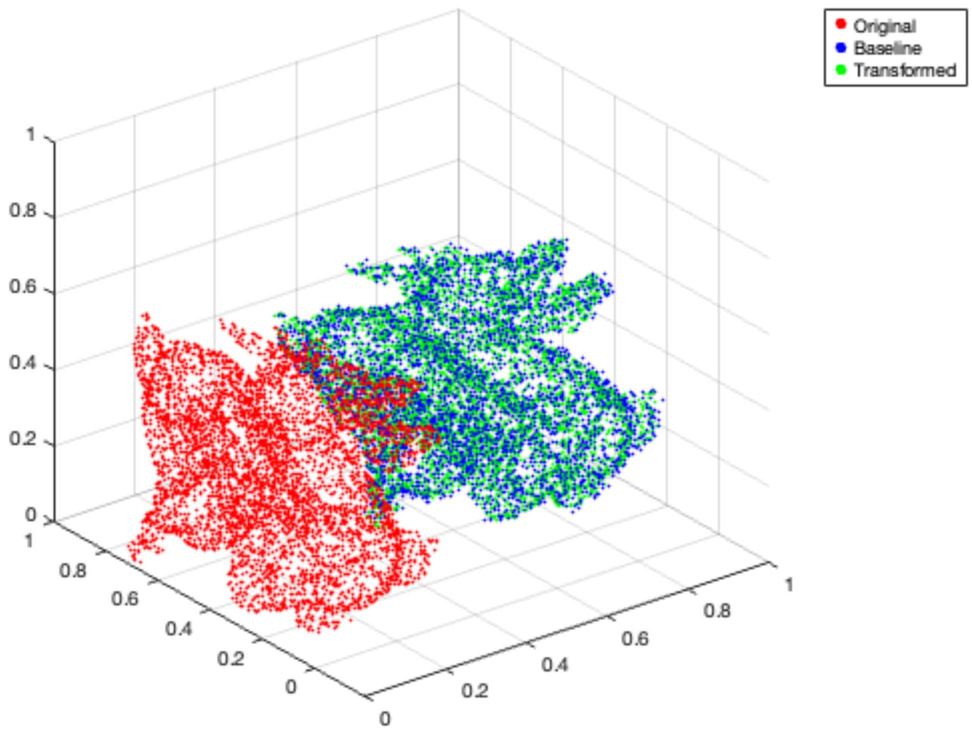
```
%T Matrix from prior script
t = [0.496614869133391
-0.293929711917010
0.296450043082626]
```

$\hat{t}$

```
scatter3(X(:,1),X(:,2),X(:,3),5,'filled','red')
hold on
scatter3(Y(:,1),Y(:,2),Y(:,3),5,'filled','blue')

length = size(X,1)

for i=1:1:length
    xi = X(i,:)';
    trans = R*xi + t;
    transformed(i,:) = trans';
end

scatter3(transformed(:,1),transformed(:,2),transformed(:,3),5,'filled','green'
)
hold on
scatter3(Y(:,1),Y(:,2),Y(:,3),5,'filled','blue')
legend ('Original','Baseline','Transformed')
```

```
transformed =

     []


R =

    0.9513   -0.1504   -0.2692
    0.2232    0.9382    0.2646
    0.2127   -0.3118    0.9260


t =

    0.4966
   -0.2939
    0.2965
```

```
length =

        5750
```



Published with MATLAB® R2024b

Red (X): Original point cloud
Blue (Y): New point cloud
Green (Trans:): Transformed X

Problem 3

a) $r(t) = X_0 \exp(t\dot{\Omega})$ where $X_0 = X @ t=0$

$$\dot{\Omega} \begin{bmatrix} 0 & -r/w\left((\dot{\varphi}_r - \dot{\varphi}_\ell + \varepsilon_r + \varepsilon_\ell)\right) & \frac{r}{2}\left((\dot{\varphi}_r - \dot{\varphi}_\ell) + \varepsilon_r + \varepsilon_\ell\right) \\ \frac{r}{w}(\dot{\varphi}_r - \dot{\varphi}_\ell) + \varepsilon_r + \varepsilon_\ell & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

then plug into $r(t) = X_0 \exp(t\dot{\Omega})$ to go to euclidean space

b) $f(x \mid \mu, \sigma^2)$

$$f(z \mid x_t) = \frac{1}{\sqrt{2\sigma_p^2 \pi}} e^{\frac{-(z - \ell_t)^2}{2\sigma_p^2}}$$

we use ℓ_t instead of $X_t(\ell_t, R_t)$ since GPS only measures ℓ_t

c $\rightarrow$ g : In code

Part e

```
% % Particle Filter
N = 1000;

Xt1 = cell(1,N); % input N Particles
Xt2 = cell(1,N);
for i = 1:N
    Xt1{i} = [0;0;0]; % assigning initial pose
end

t1=0; %first time step



  t2 =10;%second time step



  phi_l = 1.5; % left wheel commanded angular velocity
  phi_r = 2; % right wheel commanded angular velocity

  r = 0.25; %wheel radius
  w = 0.5 ;%wheel track width

  sig_l = 0.05; % uncertainty in left wheel speed
  sig_r = 0.05; % uncertainty in right wheel speed
  sig_p = 0.10; % uncertainty in measurement speed
  Xi=[];

  dt = t2-t1;
    for i=1:1:N

        xi = Xt1{i}; % extracting particle from array

        x = xi(1);
        y = xi(2); % extracting X, Y and theta from particle
        angle = xi(3);

        T_x1 = [cos(angle),-sin(angle),x  %creating homogenous T for particle
xi
                sin(angle),cos(angle),y
                0, 0, 1];

       % calculating motion model on lie group
        phi_r_noise = phi_r + (sig_r*randn());
        phi_l_noise = phi_l + (sig_l*randn());

% converting lie group to euclidean space and updating particle
    % position using exp map

        omega_dot = [0,-(r/w)*(phi_r_noise-phi_l_noise),(r/
2)*(phi_r_noise+phi_l_noise)
                 (r/w)*(phi_r_noise-phi_l_noise),0,0
```

```
                 0,0,0];

        T_x2 = T_x1 * expm(dt*omega_dot); %updating particle position in
SE(3) using exponential map

    % extracting X Y and theta from new particle pose and reassining it to
updated particle vector
        Xt2{i} = [T_x2(1,3);T_x2(2,3);atan2(T_x2(2,1),T_x2(1,1))];

    end

%Extract positions from Xt1 and Xt2
pos_t1 = [];
pos_t2 = [];

for i = 1:N
    pos_t1(i, :) = Xt1{i}(1:2)';
    pos_t2(i, :) = Xt2{i}(1:2)';

end

% Calculate statistics for both
mean_t1 = mean(pos_t1, 1)
mean_t2 = mean(pos_t2, 1)
cov_t1 = cov(pos_t1)
cov_t2 = cov(pos_t2)

% Plotting means with particle distributions

hold on;
plot(pos_t1(:,1), pos_t1(:,2), 'b.', 'MarkerSize', 8);
plot(mean_t1(1), mean_t1(2), 'k*', 'MarkerSize', 15, 'LineWidth', 2);
title('Initial Particles (Xt1)');

plot(pos_t2(:,1), pos_t2(:,2), 'r.', 'MarkerSize', 8);
plot(mean_t2(1), mean_t2(2), 'k+', 'MarkerSize', 15, 'LineWidth', 2);
title('Propagated Particles (Xt2)');
legend('Particles', 'Mean for T=0','Particles at T=10','Mean for T=10');
grid on
axis equal
```

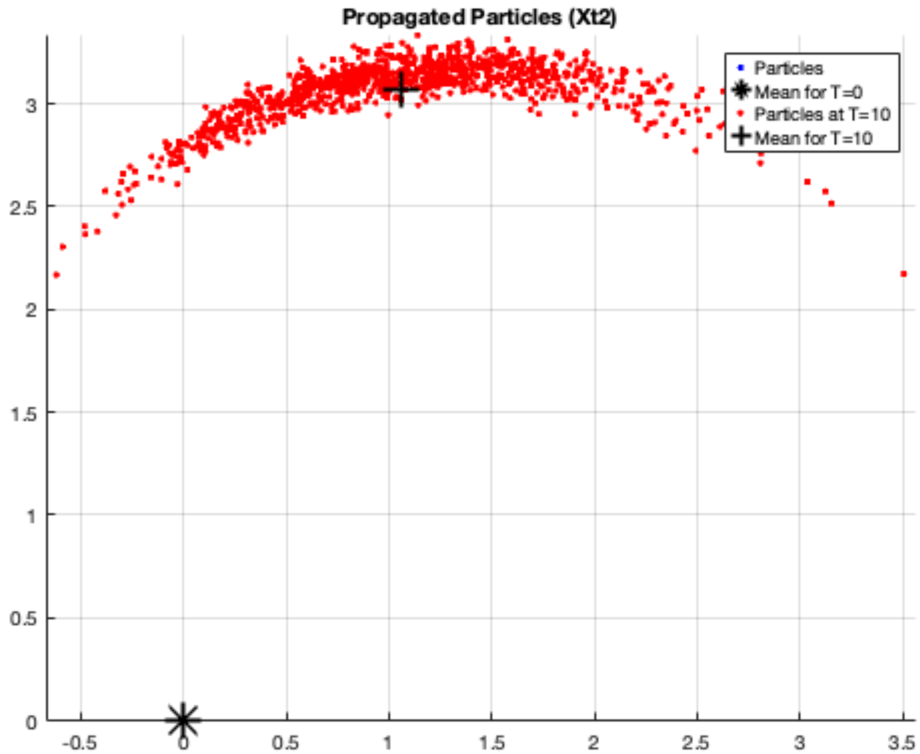
```
mean_t1 =

     0     0

mean_t2 =

    1.0582    3.0674
```

Mean of sampled partiles @ T = 10s

```
cov_t1 =
```

```
     0         0
     0         0

cov_t2 =

    0.4163    0.0400
    0.0400    0.0217
```

← Covariance of sampled paitches @ T=10s



Published with MATLAB® R2024b

Part f code

```
% % Particle Filter
N = 1000;

Xt1 = cell(1,N); % input N Particles
Xtn = cell(5,1); % blank array to store particle poses at t=0 to t=20
for i = 1:N
    Xt1{i} = [0;0;0]; % assigning initial pose at t=0
end

Xtn{1} = Xt1;

t1=0; %first time step

  phi_l = 1.5; % left wheel commanded angular velocity
  phi_r = 2; % right wheel commanded angular velocity

  r = 0.25; %wheel radius
  w = 0.5; %wheel track width

  sig_l = 0.05; %left wheel speed uncertainty
  sig_r = 0.05; %rght wheel speed uncertainty
  sig_p = 0.10; %measurement uncertainty
  Xi=[];

  t_init = t1;

  count = 2; %  just so we're starting from index #2 when assigning new
particle positions

  for t=5:5:20

      dt = t-t_init; % time step given by time interval
      Xt2 = cell(1,N); % blank array for updated position

    for i=1:1:N

        xi = Xt1{i}; % extracting particle from array

        x = xi(1);
        y = xi(2); % extracting X, Y and theta from particle
        angle = xi(3);

        T_x1 = [cos(angle),-sin(angle),x % creating homogenous T for particle
xi
                sin(angle),cos(angle),y
                0, 0, 1];

    % calculating motion model on lie group
        phi_r_noise = phi_r + (sig_r*randn());
        phi_l_noise = phi_l + (sig_l*randn());

        omega_dot = [0,-(r/w)*(phi_r_noise-phi_l_noise),(r/
```

```
2)*(phi_r_noise+phi_l_noise)
                 (r/w)*(phi_r_noise-phi_l_noise),0,0
                 0,0,0];
    % converting lie group to euclidean space and updating particle
    % position using exp map

        T_x2 = T_x1 * expm(dt*omega_dot);
   % extracting X Y and theta from new particle pose and reassining it to
updated particle vector
        Xt2{i} = [T_x2(1,3);T_x2(2,3);atan2(T_x2(2,1),T_x2(1,1))] ;
    end

    Xtn{count} = Xt2; % transferring new pose to matrix containing all time
poses
   Xt1 = Xt2; % resetting starting pose
   t_init = t; % updating time step
   count = count+1; %updating counter for next iteration
  end
```

Plotting Code

```
% Calculating Mean for every position
num_iters = 5;
times = [0, 5, 10, 15, 20];

for t = 1:num_iters
    coords = [N,2];

    % Extract positions
    for i = 1:N
        positions(i, :) = Xtn{t}{i}(1:2)'; % extracting X,Y position from
parent array containing all iteration information
    end

    % Calculate mean and covariance
    t
    mean_pos = mean(positions, 1)
    cov_pos = cov(positions)

end

% Plot all particle sets on one plot
figure;
hold on;

colors = {'b', 'r', 'g', 'k', 'c'};
markers = {'.', '.', '.', '.', '.'};

for t = 1:num_iters
    positions = [N,2];

    % Extract positions
    for i = 1:N
```

```
        positions(i, :) = Xtn{t}{i}(1:2)';
    end

    % Plot particles
    plot(positions(:,1), positions(:,2), [colors{t},
markers{t}],'MarkerSize', 5, 'DisplayName', sprintf('t = %d s', times(t)));
end

xlabel('x (m)');
ylabel('y (m)');
title('Particle Filter: Positions at given time steps');
legend('Location', 'best');
grid on;
axis equal;
hold off;
```

Results of mean and covariance of particle positions

```
t =

     1
```

0seconds

```
mean_pos =

     0     0


cov_pos =

     0     0
     0     0


t =

     2
```

5 seconds

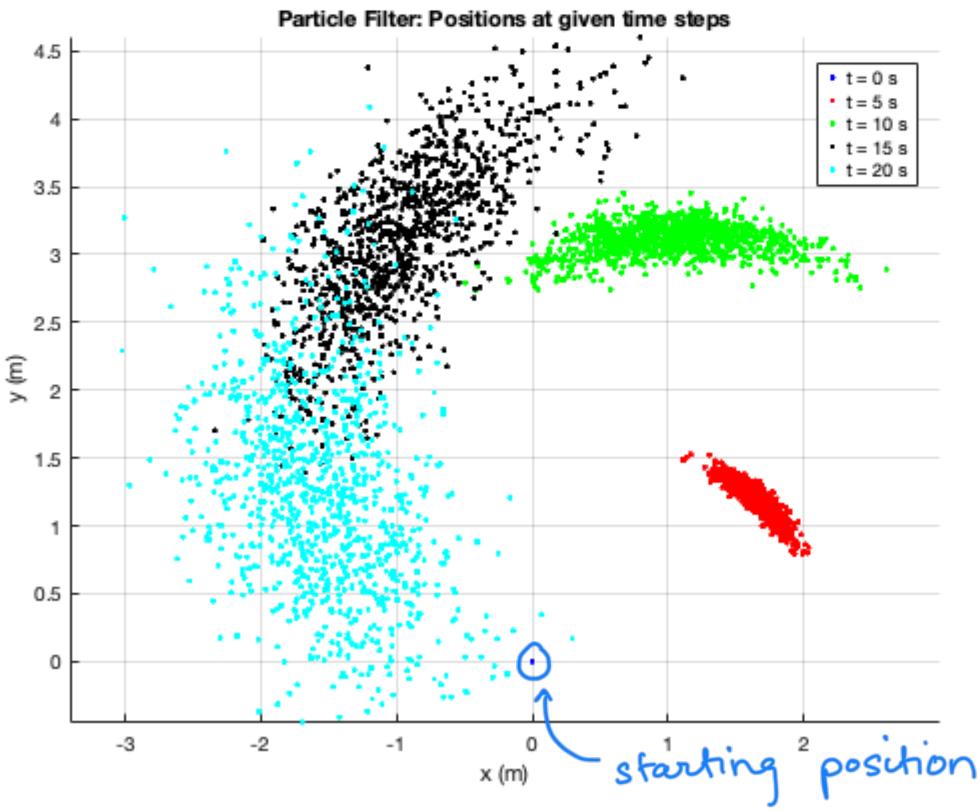
```
mean_pos =

    1.6467    1.1964


cov_pos =

    0.0200   -0.0159
   -0.0159    0.0157


t =

     3
```

10seconds

```
mean_pos =

    1.0317    3.1078


cov_pos =

    0.2504    0.0062
    0.0062    0.0143


t =

     4

mean_pos =

   -0.9387    3.1189


cov_pos =

    0.2725    0.2214
    0.2214    0.3467


t =

     5

mean_pos =

   -1.5512    1.2755


cov_pos =

    0.2325   -0.1188
   -0.1188    0.6127
```

15 seconds

20 seconds



Published with MATLAB® R2024b

part g code

```
N = 1000;

Xt1 = cell(1,N); % input N Particles
Xtn = cell(5,1); % blank array to store particle poses at t=0 to t=20
for i = 1:N
    Xt1{i} = [0;0;0];  % assigning initial pose at t=0
end
Xtn{1} = Xt1;

t1=0; %first time step

  %t2 =10;%second time step

  phi_l = 1.5; % left wheel commanded angular velocity
  phi_r = 2; % right wheel commanded angular velocity

  r = 0.25; %wheel radius
  w = 0.5 ;%wheel track width

  sig_l = 0.05; % left wheel speed uncertainty
  sig_r = 0.05; % right wheel speed uncertainty
  sig_p = 0.10; %measurement uncertainty

  z = cell(1,4); % aarray containing measured positions at 4 different time
steps

  z{1} = [1.6561;1.2847];
  z{2} = [1.0505;3.1059];
  z{3} = [-0.9875;3.2118];
  z{4} = [-1.645;1.1978];

  count = 2;
  z_count = 1;

  for t=5:5:20

      dt = t-t1;
      Xt2 = cell(1,N);

    for i=1:1:N

        xi = Xt1{i};

        x = xi(1);
        y = xi(2);
        angle = xi(3);

        T_x1 = [cos(angle),-sin(angle),x
                sin(angle),cos(angle),y
                0, 0, 1];


        phi_r_noise = phi_r + (sig_r*randn());
```

```
        phi_l_noise = phi_l + (sig_l*randn());

        omega_dot = [0,-(r/w)*(phi_r_noise-phi_l_noise),(r/
2)*(phi_r_noise+phi_l_noise)
                 (r/w)*(phi_r_noise-phi_l_noise),0,0
                 0,0,0];

        T_x2 = T_x1 * expm(dt*omega_dot); % confirm order of multiplication

        Xt2{i} = [T_x2(1,3);T_x2(2,3);atan2(T_x2(2,1),T_x2(1,1))];
    end

% starting partcile filter sampling / importance
wi=[1:N]; % array of probabilities
den = 2*sig_p^2;
diff = [1:N]; % empty array to store difference values
for i = 1:1:N
    current_particle = Xt2{i};
    lt = current_particle(1:2,1);
    diff(i) = (norm(z{z_count} - lt))^2;  % measurement - predicted position
    wi(i) = (1/sqrt(den*pi)) * exp(-diff(i)/ den); % Changed expm to exp

end

cumulative=0;
for i = 1:1:N
    cumulative = cumulative + wi(i);
end

wi_weighted = wi/cumulative;
cdf = cumsum(wi_weighted);
X_bar = cell(1,N);
% resampling step

% Generate systematic samples
u0 = rand() / N;  % Random starting point
u = u0 + (0:N-1)' / N;  % Equally spaced samples

% Resample
j = 1;
for i = 1:N
    while u(i) > cdf(j)
        j = j + 1;
    end
    X_bar{i} = Xt2{j};  % Copy selected particle
end

    Xtn{count} = X_bar;
   Xt1 = X_bar;
   t1 = t;
   count = count+1;
   z_count = z_count + 1;

  end
```

Plotting Code

Calculating Mean for every position

```
num_iters = 5;

for t = 1:num_iters
    coords = [N,2];

    % Extract positions
    for i = 1:N
        positions(i, :) = Xtn{t}{i}(1:2)'; % extracting X,Y position from
parent array containing all iteration information
    end

    % Calculate mean and covariance
    t
    mean_pos = mean(positions, 1)
    cov_pos = cov(positions)

end

times = [0, 5, 10, 15, 20];

for t = 1:num_iters
    coords = [N,2];

    % Extract positions
    for i = 1:N
        positions(i, :) = Xtn{t}{i}(1:2)'; % extracting X,Y position from
parent array containing all iteration information
    end

    % Calculate mean and covariance
    t;
    mean_pos = mean(positions, 1);
    cov_pos = cov(positions);

end

% Plot all particle sets on one plot
figure;
hold on;

colors = {'b', 'r', 'g', 'k', 'c'};
markers = {'.', '.', '.', '.', '.'};

for t = 1:num_iters
    positions = [N,2];

    % Extract positions
    for i = 1:N
        positions(i, :) = Xtn{t}{i}(1:2)';
```

```
    end

    % Plot particles
    plot(positions(:,1), positions(:,2), [colors{t},
markers{t}],'MarkerSize', 5, 'DisplayName', sprintf('t = %d s', times(t)));
end

xlabel('x (m)');
ylabel('y (m)');
title('Particle Filter: Measured and Filtered Positions');
legend('Location', 'best');
grid on;
axis equal;
hold off;
```

mean and Covariance of particle positions

```
t =

     1
```

t = 0 (start)

```
mean_pos =

     0     0


cov_pos =

     0     0
     0     0


t =
```

t = 5 seconds

```
     2


mean_pos =

    1.6292    1.2362


cov_pos =

    0.0051   -0.0031
   -0.0031    0.0039


t =
```

t = 10 seconds

```
     3


mean_pos =
```

```
    1.0304    3.1364


cov_pos =

    0.0089    0.0008
    0.0008    0.0047
```

```
t =
```

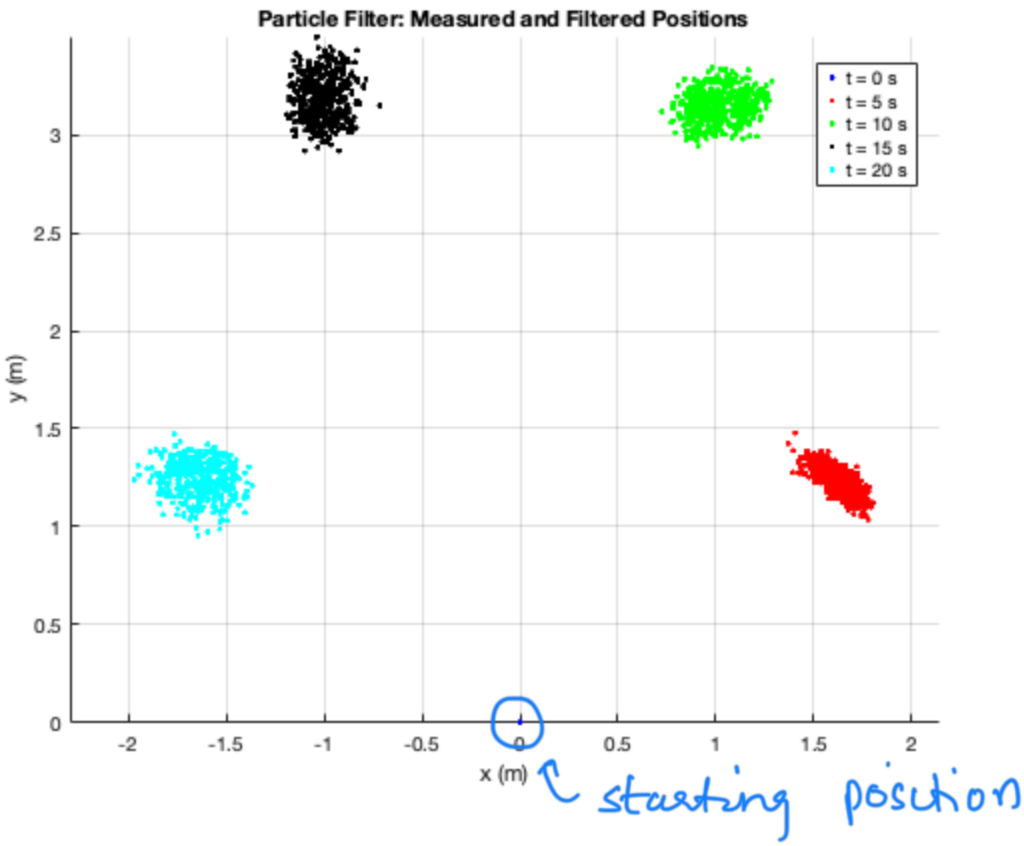
t=15 seconds

```
     4


mean_pos =

   -1.0012    3.2015


cov_pos =

    0.0050    0.0004
    0.0004    0.0084
```

```
t =
```

t=20 seconds

```
     5


mean_pos =

   -1.6421    1.2282


cov_pos =

    0.0083   -0.0010
   -0.0010    0.0062
```



Published with MATLAB® R2024b