Mobile Robotics EE/CE 468

Homework Assignment 03

Lyeba Abid - la07309
Ali Muhammad - aa07190

Question:	1	2	3	4	5	Total
Points:	20	20	20	20	20	100
Score:						

Problem 1 [20 Points]

Solution: The basic equations of motion are:
1.$r = r_0 + v\triangle t + \frac{1}{2}a\triangle t^2$
2.$v = v_0 + a\triangle t$

Since the acceleration $\ddot{x}$ is the sum of the commanded accceleration and a noise term, ε, we can denote it as $\ddot{x} = a_t + \varepsilon$ where a_t is the commanded acceleration and ε is the noise term with zero-mean and variance σ^2, and $E[\ddot{x}_t] = a_t$, $\text{Var}(\ddot{x}_t) = \sigma^2$.

Then our equations become:
1. $x_{t+1} = x_t + \dot{x}_t\triangle t + \frac{1}{2}(a_t + \varepsilon)\triangle t^2$
2. $\dot{x}_{t+1} = \dot{x}_t + (a_t + \varepsilon)\triangle t$

(a) Based on the above equations, let our state vector be $x_{t+1} = \begin{bmatrix} x_{t+1} \\ \dot{x}_{t+1} \end{bmatrix}$.
Then $x_{t+1} = f(x_t, a_t)$. And we can rearrange our equations to get:

$$x_{t+1} = \begin{bmatrix} 1 & \triangle t & \frac{1}{2}\triangle t^2 \\ 0 & 1 & \triangle t \end{bmatrix} \begin{bmatrix} x_t \\ \dot{x}_t \\ a_t \end{bmatrix} = \begin{bmatrix} x_t + \dot{x}_t\triangle t + \frac{1}{2}a_t\triangle t^2 \\ \dot{x}_t + a_t\triangle t \end{bmatrix}$$

We know the covariance matrix can be calculated as follows:

$$P_{k+1} = \mathcal{J}_1 P_k \mathcal{J}_1^T + \mathcal{J}_2 \Sigma a_o \mathcal{J}_2^T$$

where P_t is the covariance of x_t at time t, and $\mathcal{J}_1 = \nabla_{x_t} f = \begin{bmatrix} \frac{\partial f}{\partial x_t} & \frac{\partial f}{\partial \dot{x}} \end{bmatrix}$ and $\mathcal{J}_2 = \nabla_a f$

Keeping our equations in mind, we get $\mathcal{J}_1 = \begin{bmatrix} 1 & \triangle t \\ 0 & 1 \end{bmatrix}$ and $\mathcal{J}_2 = \begin{bmatrix} \frac{1}{2}\triangle t^2 \\ \triangle t \end{bmatrix}$

Pluggin these into our covariance equation (assuming that $P_0 = 0$):

$$P_1 = \begin{bmatrix} 1 & \triangle t \\ 0 & 1 \end{bmatrix} P_k \begin{bmatrix} 1 & 0 \\ \triangle t & 1 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\triangle t^2 \\ \triangle t \end{bmatrix} \sigma^2 \begin{bmatrix} \frac{1}{2}\triangle t^2 & \triangle t \end{bmatrix} = 0 + \sigma^2 \begin{bmatrix} \frac{1}{2}\triangle t^2 \\ \triangle t \end{bmatrix} \begin{bmatrix} \frac{1}{2}\triangle t^2 & \triangle t \end{bmatrix}$$

$$P_1 = \sigma^2 \begin{bmatrix} \frac{1}{4}\triangle t^4 & \frac{1}{2}\triangle t^3 \\ \frac{1}{2}\triangle t^3 & \triangle t^2 \end{bmatrix} = \sigma^2\triangle t^2 \begin{bmatrix} \frac{1}{4}\triangle t^2 & \frac{1}{2}\triangle t \\ \frac{1}{2}\triangle t & 1 \end{bmatrix}$$

$$\therefore \text{Covariance} = \begin{bmatrix} \frac{1}{4}\triangle t^2 & \frac{1}{2}\triangle t \\ \frac{1}{2}\triangle t & 1 \end{bmatrix}$$

The existence of non-zero terms on the diagonals implies that there exists a correlation between the location, x, and the velocity, $\dot{x}$. [Our assumption being that $P_0 = 0$, therefore we get P_1, the covariance matrix at the first time step, which is then propogated forward in time.]

(b) The position of our robot is determined by a culmination of past velocities, and acceleration, and therefore, noise in the system. We established in our previous answer that there is a correlation between x and $\dot{x}$. So for very large values of T, this will

also be particularly true as there's a likelihood that x and $\dot{x}$ will be correlated as the accumalated effects of noise in the acceleration will significantly impact both the position and velocity. We can show this formally as well.

Let $Q = \mathcal{J}_2 \Sigma_a \mathcal{J}_2^T = \sigma^2 \triangle t^2 \begin{bmatrix} \frac{1}{4}\triangle t^2 & \frac{1}{2}\triangle t \\ \frac{1}{2}\triangle t & 1 \end{bmatrix}$

Then the covariance at any given time T is $P_T = \mathcal{J}_1 P_{T-1} \mathcal{J}_1^T + Q$

Since the covariance at any given step depends on the covariance on the previous steps as well, we can expand the above equation as follows:

$$
\begin{aligned}
P_T &= \mathcal{J}_1 P_{T-1} \mathcal{J}_1^T + Q \\
P_T &= \mathcal{J}_1 (\ \mathcal{J}_1 P_{T-2} \mathcal{J}_1^T + Q\ ) \mathcal{J}_1^T + Q \\
P_T &= \mathcal{J}_1^2\ P_{T-2}\ (\mathcal{J}_1^T)^2 + \mathcal{J}_1 Q \mathcal{J}_1^T + Q \\
P_T &= \mathcal{J}_1^3\ P_{T-3}\ (\mathcal{J}_1^T)^3 + \mathcal{J}_1^2\ Q\ (\mathcal{J}_1^T)^2 + \mathcal{J}_1 Q \mathcal{J}_1^T + Q \\
&\ \ \vdots \qquad\qquad \vdots \\
P_T &= (\mathcal{J}_1)^T\ P_0\ (\mathcal{J}_1^T)^T + (\mathcal{J}_1)^{T-1}\ Q\ (\mathcal{J}_1^T)^{T-1} + ... + \mathcal{J}_1\ Q\ \mathcal{J}_1^T + Q \\
P_T &= (\mathcal{J}_1)^{T-1}\ Q\ (\mathcal{J}_1^T)^{T-1} + ... + \mathcal{J}_1\ Q\ \mathcal{J}_1^T + Q \qquad\qquad [P_0 = 0]
\end{aligned}
$$

The above equation shows that for a very large T, the covariance is the sum of all the previous covariances. The Jacobian Matrices $\mathcal{J}_1$ and $\mathcal{J}_2$, and Q will all be positive since they involve the change in time, and as we are moving forward in time, their products will also be positive (non-zero). Hence, x and $\dot{x}$ are correlated.

Problem 2 [20 Points]

Solution:

(a) Initial State Esimates:

$$\hat{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} \qquad \text{and} \qquad g(\hat{x}_{k-1}, u_k) = \hat{x}_{t-1} = \begin{bmatrix} x_t + v_x \triangle t cos(\theta_t) \\ y_t + v_y \triangle t sin(\theta_t) \\ \theta_t + \omega \triangle t \end{bmatrix}$$

Then our prediction step becomes $\overline{x}_k = g(\hat{x}_{k-1}, u_k)$.

Considering a specific case of $\omega_k = 0$, we get $\overline{x}_k = \begin{bmatrix} x_t + v_x \triangle t cos(\theta_t) \\ y_t + v_y \triangle t sin(\theta_t) \\ \theta_t \end{bmatrix}$

Then our covariance becomes $\overline{\Sigma}_k = G_k \Sigma_{k-1} G_k^T + R_k$ where $G_k = \dfrac{\partial g(\hat{x}_{k-1}, u_k)}{\partial \hat{x}_{k-1}}$, which

for the specific case becomes $G_{k+1} = \frac{\partial g(\hat{x}_t, u_{t+1})}{\partial \hat{x}_t} = \begin{bmatrix} 1 & 0 & -v_x \Delta t \sin(\theta_t) \\ 0 & 1 & v_y \Delta t \cos(\theta_t) \\ 0 & 0 & 1 \end{bmatrix}$

We define Kalman Gain as $K_k = \Sigma_k H_k^T (H_k \Sigma_k H_k^T + Q_k)^{-1}$ where $H_k = \frac{\partial \overline{h}(\hat{x}_k)}{\partial \hat{x}_k}$

In our case, $\overline{h}(\hat{x}_{t+1}) = \begin{bmatrix} d \\ \theta \end{bmatrix} = \begin{bmatrix} \sqrt{\hat{x}^2 + \hat{y}^2} \\ \tan^{-1}\left(\frac{\hat{y}}{\hat{x}}\right) \end{bmatrix}$

where $\hat{x}$ and $\hat{y}$ are obtained from sensor measurement.

Hence, $H_{k+1} = \frac{\partial \overline{h}(\hat{x}_{t+1})}{\partial \hat{x}_{t+1}} = \begin{bmatrix} \frac{\hat{x}}{\sqrt{\hat{x}^2 + \hat{y}^2}} & \frac{\hat{y}}{\sqrt{\hat{x}^2 + \hat{y}^2}} & 0 \\ -\frac{\hat{y}}{\hat{x}^2 + \hat{y}^2} & \frac{\hat{x}}{\hat{x}^2 + \hat{y}^2} & 0 \end{bmatrix}$

Keeping the above in mind, our update step, which is generally defined as

$$\hat{x}_k = \overline{x}_k + K_k(\theta_k - h(\overline{x}_k))$$
$$\Sigma_k = (I - K_k H_k)\overline{\Sigma}_k$$

which in our case become

$$\hat{x}_{k+1} = \overline{x}_{k+1} + K_{k+1}(z_{k+1} - \overline{h}(\overline{x}_{k+1}))$$
$$\Sigma_{k+1} = (I - K_{k+1} H_{k+1})\Sigma_{k+1}$$

(b) Propogation of 1000 samples of initial state

```
samples = 1000;
initial = mvnrnd([0; 0; 0], [0.01, 0, 0; 0, 0.01, 0; 0, 0, 10000], samples);

figure;
scatter(initial(:, 1), initial(:, 2), 'b.');
xlabel('X'); ylabel('Y');
title('Samples from Gaussian Distribution');
propagated_samples = zeros(samples, 3);
for i = 1:samples
    % Motion model (assumption: time step distance = 1)
    propagated_samples(i, :) = initial(i, :) + [cos(initial(i, 3)), sin(initial(i, 3)), 0];
end
hold on;
scatter(propagated_samples(:, 1), propagated_samples(:, 2), 'r.');
xlabel('X'); ylabel('Y');
title('Samples of x-y state at time 1');
legend('Initial Samples', 'Propagated Samples');
```

```
hold off;
```

Listing 1: Propogation Model - 1000 samples

The above code generates the following plot:

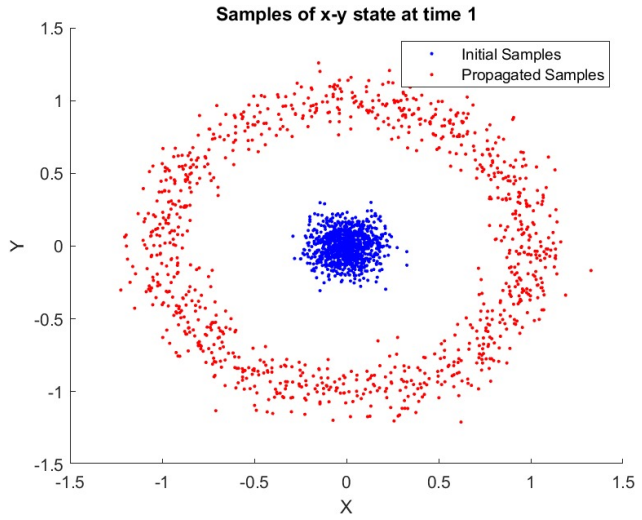


Figure 1: Propagation Model

(c) Building on the code provided in part (b), we add the following code to generate the ellipse:

```
% Constants
dt = 1;  % time step
Q = diag([0, 0, 0]);  % process noise covariance
R = diag([0.1, 0.1]);  % measurement noise covariance (assumed
    values)

% Initial states
x_hat = [0; 0; 0];
P = [0.01, 0, 0; 0, 0.01, 0; 0, 0, 10000];

num_steps = 1;  % number of steps

predicted_states = zeros(num_steps, 3);
predicted_covariances = zeros(num_steps, 3, 3);

% Kalman Filter Loop
for k = 1:num_steps
v_x = 1; w = 0;
```

```
18 x_hat = x_hat + [v_x*dt*cos(x_hat(3,:)); v_x*dt*sin(x_hat(3,:))
       ; w*dt];  % since there is no control input, the state
       remains the same
19 G = [1 0 v_x*dt*sin(x_hat(3,:)); 1 0 v_x*dt*cos(x_hat(3,:)); 0
       0 1];
20 P = G * P * G' + Q;
21
22 predicted_states(k, :) = x_hat;
23 predicted_covariances(k, :, :) = P;
24 end
25
26 scatter(propagated_samples(:, 1), propagated_samples(:, 2), 'b.
       ');
27 xlabel('X'); ylabel('Y');
28 title('Error Ellipse at time 1');
29 hold on;
30 for k = 1:num_steps
31 plotErrorEllipse(predicted_states(k, 1:2), squeeze(
       predicted_covariances(k, 1:2, 1:2)), 0.95);
32 end
33 legend('Propagated Samples', 'Error Ellipse');
34 hold off;
```

Listing 2: EKF: Predicting State with Corresponding Covariance

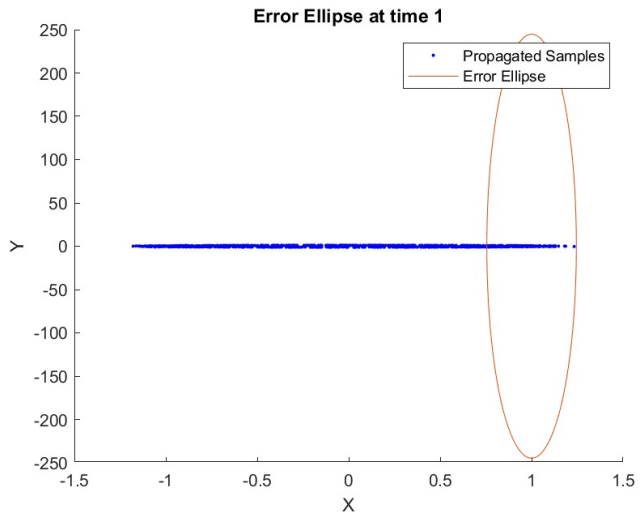


Figure 2: Error Ellipse of Samples and EKF Prediction

The above figure depicts that the propogated samples (blue dots), are tightly clustered along the y-axis, which is consistent with the motion model where the orientation θ was

initially zero, and the robot was moving along the x-axis. Since the robot's heading is not being controlled ($\omega = 0$), the motion in y-direction is due to the initial uncertainty in the orientation. The error ellipse appears to be oriented and stretched more along the y-axis, which may seem counter-intuitive since the robot was moving along the x-axis. But this can happen if the covariance in the orientation θ is large, causing the propogated uncertainty to spread out more along y as time progresses, which is reflected in our initial covariance matrix, where the variance for θ is much larger than for x or y.

(d) Again, buiding on the code in part (b), we add the following code to incorporate a noisy measurement, $z = d + \varepsilon$ where ε is a zero-mean with covariance 0.01:

```
%% 2d
% Number of samples
num_samples = 1000;

% Draw samples from Gaussian distribution
initial_samples = mvnrnd([0; 0; 0], [0.01, 0, 0; 0, 0.01, 0; 0,
    0, 10000], num_samples);

% Plot 3D samples
figure;
% scatter3(initial_samples(:, 1), initial_samples(:, 2),
   initial_samples(:, 3), 'b.');
scatter(initial_samples(:, 1), initial_samples(:, 2), 'b.');
xlabel('X');
ylabel('Y');
zlabel('Theta');
title('Samples from Gaussian Distribution');

% Propagate samples according to motion equation
propagated_samples = zeros(num_samples, 3);
for i = 1:num_samples
    % Motion model (assuming distance covered in one time step
        is 1)
    propagated_samples(i, :) = initial_samples(i, :) + [cos(
        initial_samples(i, 3)), sin(initial_samples(i, 3)), 0];
end

% Hold on to keep the current plot and add new plots
hold on;

% Plot 2D samples at time 1
scatter(propagated_samples(:, 1), propagated_samples(:, 2), 'r.
    ');
xlabel('X');
ylabel('Y');
title('Samples of x-y state at time 1');
legend('initial samples', 'predicted samples');
```

```
% Release the hold
hold off;

% the radius in each direction is the standard deviation sigma\_x and sigma\_y parametrized by a scale factor s,
% known as the Mahalanobis radius of the ellipsoid

% We can solve this equation using a chi square table or using Matlab function s=chi2inv(p, 2) or simply:
% s= -2log(1-p)

% Constants
dt = 1;  % time step
Q = diag([0, 0, 0]);  % process noise covariance
R = diag([0.1, 0.1]);  % measurement noise covariance (assumed values)

% Initial state estimate
x_hat = [0; 0; 0];
% Initial state covariance
P = [0.01, 0, 0; 0, 0.01, 0; 0, 0, 10000];

% Simulation parameters
num_steps = 1;  % number of steps

% Measurement Noise Covariance
R_measurement = 0.01;

% Kalman Filter Loop
for k = 1:num_steps
    v_x = 1;
    w = 0;       % omega = 0
    % Prediction Step
    x_bar = x_hat + [v_x*dt*cos(x_hat(3,1)); v_x*dt*sin(x_hat(3,1)); w*dt];  % since there is no control input, the state remains the same
    G = [1 0 -v_x*dt*sin(x_hat(3,1)); 0 1 v_x*dt*cos(x_hat(3,1)); 0 0 1];  % state transition matrix (no change in state)

    % Update Covariance
    P_bar = G * P * G' + Q;

    % Store predicted state and covariance
    predicted_states(k, :) = x_bar;
    predicted_covariances(k, :, :) = P_bar;

%     hold on
```

```
74  %     for k = 1:num_steps
75  %         plotErrorEllipse(predicted_states(k, 1:2), squeeze(
        predicted_covariances(k, 1:2, 1:2)), 0.95);
76  %     end
77      hold off
78
79      % Simulate a noisy measurement generate based on a normal
            distribution with mean 0 and standard deviation
80      d = sqrt(x_bar(1)^2 + x_bar(2)^2);
81  %     theta = atan2(x_bar(2), x_bar(1));
82       R_measurment = 0.0001;
83      E = mvnrnd(0, R_measurment);
84      z_k = d + E; % Noisy measurment
85  %     z_k = [d; theta] + E; % Noisy measurment
86
87      x = x_bar(1, :);
88      y = x_bar(2, :);
89      % Compute partial derivatives
90      partial_d_x = x / sqrt(x^2 + y^2);
91      partial_d_y = y / sqrt(x^2 + y^2);
92
93      partial_theta_x = -y / (x^2 + y^2);
94      partial_theta_y = x / (x^2 + y^2);
95
96      % Construct Jacobian matrix H
97      H = [partial_d_x, partial_d_y, 0];
98
99  % H = [1 0];
100     h_xk_bar = H * x_bar;
101
102     % Kalman Gain
103     S = H * P_bar * H' + R_measurement;
104     K = P_bar * H' / S;
105
106     % Update State Estimate
107     y = z_k - h_xk_bar;
108     x_hat_up = x_bar + K * y;
109
110     % Update Covariance
111     P_up = (eye(3) - K * H) * P_bar;
112
113
114     % Store updated state and covariance
115     updated_states(k, :) = x_hat_up;
116     updated_covariances(k, :, :) = P_up;
117 end
118
119
```

```
% Plotting Part (c) - Propagated samples and error ellipse for predicted step
figure;
scatter(propagated_samples(:, 1), propagated_samples(:, 2), 'b.');
hold on;
for k = 1:num_steps
    plotErrorEllipse(predicted_states(k, 1:2), squeeze(predicted_covariances(k, 1:2, 1:2)), 0.95);
end
xlabel('X');
ylabel('Y');
title('Propagated Samples and Predicted Error Ellipse');
legend('propagated samples', 'Error in prediction');
hold off;

% Plotting Part (d) - Propagated samples, error ellipse for predicted step, and error ellipse for updated step
figure;
scatter(propagated_samples(:, 1), propagated_samples(:, 2), 'b.', 'DisplayName', 'Propagated Samples');
hold on;
for k = 1:num_steps
    plotErrorEllipse(predicted_states(k, 1:2), squeeze(predicted_covariances(k, 1:2, 1:2)), 0.95);
end
for k = 1:num_steps
    plotErrorEllipse(updated_states(k, 1:2), squeeze(updated_covariances(k, 1:2, 1:2)), 0.95);
end
xlabel('X');
ylabel('Y');
title('Propagated Samples with Predicted and Updated Error Ellipse');
legend('propagated samples', 'Error in prediction', 'Error in Measurment');
hold off;
```

Listing 3: EKF: Incorporating Noisy Measurement

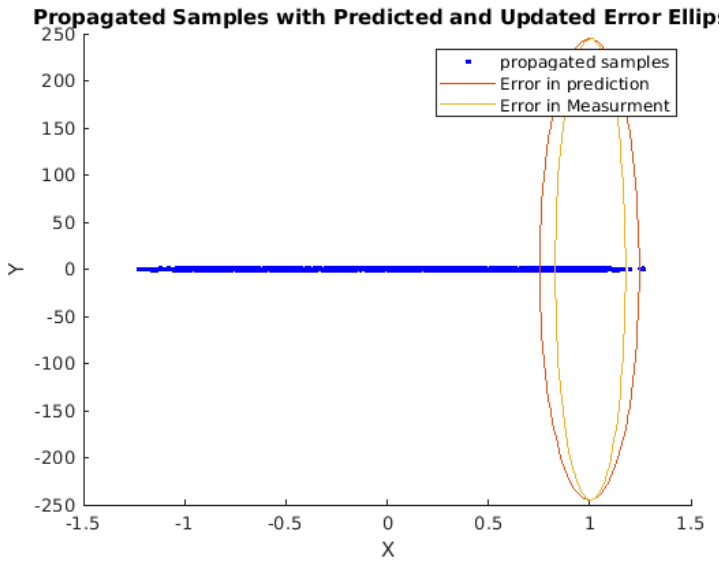


Figure 3: Propagated Samples and Error Ellipse for Predicted Step

Incorporating a noisy measurement into the Kalman filter simulation involves introducing uncertainty to sensor readings. The code introduces a noisy measurement $z = d + \varepsilon$, where ε is zero-mean Gaussian noise with a covariance of 0.01. After predicting the state using motion models, the noisy measurement is integrated into the update step. The resulting plot, as shown above, depicts the propagated samples, error ellipses for both predicted and updated states, and the additional error ellipse after assimilating the noisy measurement. This visual representation illustrates how the Kalman filter efficiently reduces uncertainty by integrating real-world sensor information, leading to a refined and more accurate state estimation, as shown by the difference in the ellipses in prediction, and measurement. The measurement can clearly be seen to be smaller, thus more accurate, than the prediction.

(e) Considering (a), our estimate for $x - y$ at time 1 would be the average of the samples from the distribution. Considering the EKF, the estimate would be better, as we can see from the previous parts, the uncertainty along y is high, however, the uncertainty along x is low, so the estimate is better. If the initial orientation was known, but the uncertainty along y was high, then the estimate would be better in the EKF case, since the covariance in the pose would be lower.

Problem 3 [20 Points]

Solution:

(a) **i. Data available to the day in Question; future data not available**

Day 2:

$$
\begin{aligned}
p(x_2|x_1, z_2) &= \eta p(z_2|x_1, x_2)p(x_2|x_1) \\
p(z_2|x_1, x_2) &= p(z_2|x_2) \\
p(x_2|x_1, z_2) &= \eta p(z_2|x_2) \\
\text{After normalization, } \eta &= 0.54 \\
p(x_2|x_1 = S, z_2 = S) &= \eta p(z_2 = S|x_2 = S)p(x_2 = S|x_1 = S) \\
&= \eta \times 0.6 \times 0.8 = \eta \times 0.48 \\
p(x_2|x_1 = C, z_2 = S) &= \eta p(z_2 = S|x_2 = C)p(x_2 = C|x_1 = S) \\
&= \eta \times 0.3 \times 0.2 = \eta \times 0.06 \\
p(x_2|x_1 = R, z_2 = S) &= \eta p(z_2 = S|x_2 = R)p(x_2 = R|x_1 = S) \\
&= 0 \\
\text{After normalization, } \eta &= 0.54 \\
p(x_2|x_1 = S, z_2 = S) &= \text{bel}(x_2) = [0.89, 0.11, 0]
\end{aligned}
$$

Day 3:

$$
\begin{aligned}
p(x_3|x_1, z_2, z_3) &= p(x_3|x_1, z_3) = \text{bel}(x_3) \\
\text{bel}(x_3) &= \eta p(z_3 = S|x_3) \sum_{x_2} p(x_3|x_2)\text{bel}(x_2) \\
&= \eta \begin{bmatrix} 0.6 & 0.3 & 0 \end{bmatrix} \times \begin{bmatrix} 0.89 \\ 0.11 \\ 0 \end{bmatrix} \\
&= \eta \begin{bmatrix} 0.4536 \\ 0.0666 \\ 0 \end{bmatrix} \\
\text{After normalization, } \eta &= 0.872 \\
\text{bel}(x_3) &= \begin{bmatrix} 0.872 \\ 0.128 \\ 0 \end{bmatrix}
\end{aligned}
$$

Day 4:

$$p(x_4|x_1, z_2, z_3, z_4) = p(x_4|x_1, z_4) = \text{bel}(x_4)$$

$$\text{bel}(x_4) = \eta p(z_4 = R|x_4) \sum_{x_3} p(x_4|x_3)\text{bel}(x_3)$$

$$= \eta \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.872 \\ 0.128 \\ 0 \end{bmatrix}$$

$$= \eta \begin{bmatrix} 0 \\ 0 \\ 0.0256 \end{bmatrix}$$

After normalization, $\eta = 1$

$$\text{bel}(x_4) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We can summarise the above data in the following table:

$\mathbf{bel(x_i)}$	$\mathbf{x_i}$ =**Sunny**	$\mathbf{x_i}$ =**Cloudy**	$\mathbf{x_i}$ =**Rainy**
$\mathbf{bel(x_2)}$	0.889	0.111	0
$\mathbf{bel(x_3)}$	0.872	0.128	0
$\mathbf{bel(x_4)}$	0	0	1

Table 1: Belief about weather from previous day, and current sensor readings

ii. Hindsight; future data also available

Considering the fact that data from the future is also available, then our probability for each day will be as follows:
Day 2: $P(x_2 \mid z_2, z_3, z_4)$
Day 3: $P(x_3 \mid z_2, z_3, z_4)$
Day 4: $P(x_4 \mid z_2, z_3, z_4)$

However, we would need to calculate the probabilities backwards, starting from Day 4 uptil Day 2. We already calculated the probability for Day 4 in the previous part, which was $P(x_4 \mid z_2, z_3, z_4) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

Day 3: $\mathbf{P(x_3 \mid z_2, z_3, z_4)}$

$$P(x_3 \mid z_2, z_3, z_4) = P(z_4|x_3, z_3, z_2)P(x_3 \mid z_2, z_3) \quad \textit{Bayes Rule}$$
$$P(x_3 \mid z_2, z_3, z_4) = P(z_4 \mid x_3)P(x_3 \mid z_2, z_3) \quad \textit{Markov Assumption}$$
$$P(x_3 \mid z_2, z_3, z_4) = \sum_{x_4}(P(z_4, x_4 \mid x_3))\ \ P(x_3 \mid z_2, z_3)$$
$$P(x_3 \mid z_2, z_3, z_4) = \sum_{x_4}(P(z_4 \mid x_4, x_3)P(x_4 \mid x_3))\ \ P(x_3 \mid z_2, z_3) \quad \textit{Bayes Rule on Sum}$$
$$P(x_3 \mid z_2, z_3, z_4) = \sum_{x_4}(P(z_4 \mid x_4)P(x_4 \mid x_3))\ \ P(x_3 \mid z_2, z_3) \quad \textit{Markov Assumption}$$

Then our equation becomes

$$P(x_3 \mid z_2, z_3, z_4) = \sum_{x_4}(P(z_4 \mid x_4)P(x_4 \mid x_3))\ \ P(x_3 \mid z_2, z_3)$$

for which we already have the values calculated from the previous part. So we can substitute the values and get the following results:

$P(x_3 = S \mid z_2, z_3, z_4) = 0,\ \ P(x_3 = C \mid z_2, z_3, z_4) = 1,\ \ P(x_3 = R \mid z_2, z_3, z_4) = 0$

Day 2: $\mathbf{P(x_2 \mid z_2, z_3, z_4)}$

$$P(x_2 \mid z_2, z_3, z_4) = P(z_4, z_3 \mid x_2, z_2)P(x_2 \mid z_2) \quad \textit{Bayes Rule}$$
$$P(x_2 \mid z_2, z_3, z_4) = P(z_4, z_3 \mid x_2)P(x_2 \mid z_2) \quad \textit{Markov Assumption}$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}(P(z_4, z_3, x_4 \mid x_2))\ \ P(x_2 \mid z_2)$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}(P(z_4 \mid x_4, z_3, x_2)P(x_4, z_3 \mid x_2))\ \ P(x_2 \mid z_2) \quad \textit{Bayes Rule on Sum}$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}(P(z_4 \mid x_4)P(x_4, z_3 \mid x_2))\ \ P(x_2 \mid z_2) \quad \textit{Markov Assumption}$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}\left[P(z_4 \mid x_4)\sum_{x_3}P(x_4, z_3, x_3 \mid x_2)\right]\ \ P(x_2 \mid z_2)$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}\left[P(z_4 \mid x_4)\sum_{x3}P(x_4 \mid z_3, x_3, x_2)P(z_3, x_3 \mid x_2)\right]P(x_2 \mid z_2) \quad \textit{Bayes Rule}$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}\left[P(z_4 \mid x_4)\sum_{x3}P(x_4 \mid x_3)P(z_3, x_3 \mid x_2)\right]P(x_2 \mid z_2) \quad \textit{Markov Assumption}$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}\left[P(z_4 \mid x_4)\sum_{x3}P(x_4 \mid x_3)P(z_3 \mid x_3, x_2)P(x_3 \mid x_2)\right]P(x_2 \mid z_2) \quad \textit{Bayes Rule}$$
$$P(x_2 \mid z_2, z_3, z_4) = \sum_{x_4}\left[P(z_4 \mid x_4)\sum_{x3}P(x_4 \mid x_3)P(z_3 \mid x_3)P(x_3 \mid x_2)\right]P(x_2 \mid z_2) \quad \textit{Markov Assumption}$$

Pluggin in the values, we get the following results:
$P(x_2 = S \mid z_2, z_3, z_4) = 0.6,\ \ P(x_2 = C \mid z_2, z_3, z_4) = 0.4,\ \ P(x_2 = R \mid z_2, z_3, z_4) = 0$

(b) Since we have three possibilities of weather on each day, then from Day 2 to Day 4, we have $3^3 = 27$ possible sequences of weather. Then we can find the probability through:

$$P(x_2, x_3, x_4 \mid x_1, z_2, z_3, z_4)$$

where x_1 is the weather on Day 1, z_2 is the observation on Day 2, z_3 is the observation on Day 3, and z_4 is the observation on Day 4.

**Note: When today's weather is sunny, tomorrow's weather is never rainy. Sensor also detects rainy perfectly, so all possibilities where day 4 is not rainy are 0.*

1. **SSS:** $P(x_2 = S, x_3 = S, x_4 = S \mid x_1 = S, z_2 = S, z_3 = S, z_4 = R)$
   $P(SSS \mid SSSR) = \eta(0.48)(0.6)(0.8)(0.48)(0) = 0$
2. **SSC:** $P(x_2 = S, x_3 = S, x_4 = C \mid x_1 = S, z_2 = S, z_3 = S, z_4 = R) = 0$
3. **SCC:** $P(x_2 = S, x_3 = C, x_4 = C \mid x_1 = S, z_2 = S, z_3 = S, z_4 = R) = 0$
4. **SRS:** $P(x_2 = S, x_3 = R, x_4 = S \mid x_1 = S, z_2 = S, z_3 = S, z_4 = R) = 0$

5...25. **= 0** These are the following cases: **(SRS), (SRC), (SRR), (CCC), (CSC), (CSS), (CRC), (CRS), (CCS), (CRR), (RRS), (RRC), (RSC), (RSS), (RCS), (RRR), (RSR), (RSC), (RCR), (RCC), (SSR), (CSR)**

26. **CCR:** $P(x_2 = C, x_3 = C, x_4 = R \mid x_1 = S, z_2 = S, z_3 = S, z_4 = R)$
    $P(CCR \mid SSSR) = \eta P(x_{2:4} \mid x_1, z_{2:4}) P(x_{2:4} \mid x_1) P(x_{2:4} \mid x_1 z_{2:4})$
    $bel(x_2 = C) = 0.06\eta, \;\; bel(x_3 = C) = 0.0072\eta, \;\; bel(x_4 = R) = 0.00144\eta$
27. **SCR:** $P(x_2 = S, x_3 = C, x_4 = R \mid x_1 = S, z_2 = S, z_3 = S, z_4 = R)$
    $P(SCR \mid SSSR) = \eta P(x_{2:4} \mid x_1, z_{2:4}) P(x_{2:4} \mid x_1) P(x_{2:4} \mid x_1 z_{2:4})$
    $bel(x_2 = S) = 0.48\eta, \;\; bel(x_3 = C) = 0.0288\eta, \;\; bel(x_4 = R) = 0.00576\eta$

After normalizing, we get the following probabilities for the two cases in which the probabilities did not end up being 0: **CCR = 0.2**, and **SCR = 0.8**

So the most likely sequence is *Sunny, Cloudy, Rainy* with a probability of 0.8.

Problem 4 [20 Points]

Solution: The completed `incrementLocalization` function is as follos:

```
function [x_posterori, P_posterori, xpriori] =
    incrementalLocalization(x, P, u, S, M, params, k, g, b)
% returns the a posterori estimate of the state and its
    covariance, given the previous state estimate, control
    inputs, laser measurements and the map

C_TR = diag([repmat(0.1^2, 1, size(S, 2)) repmat(0.1^2, 1,
    size(S, 2))]);

[z, R, ~] = extractLinesPolar(S(1,:), S(2,:), C_TR, params);

figure(2), cla, hold on;
z_prior = zeros(size(M));
for k = 1:size(M,2)
    z_prior(:,k) = measurementFunction(x, M(:,k));
end
plot(z(1,:), z(2,:),'bo');
plot(z_prior(1,:), z_prior(2,:),'rx');
xlabel('angle [rad]'); ylabel('distance [m]')
legend('measurement','prior')
drawnow

% estimate robot pose
[x_posterori, P_posterori, xpriori] = filterStep(x, P, u, z, R
    , M, k, g, b);
```

Listing 1: incrementLocalization

As per the `turtlebotEKFLocalization` function, an additional output, `xpriori`, is added to the function. This is the prior estimate of the robot pose.

The `extractLinesPolar` wasn't changed, but the `measurementFunction` was implemented as follows:

```
function [h, H_x] = measurementFunction(x, m)
% returns the predicted measurement given a state x and a
    single map entry m. H_x denotes the Jacobian of the
    measurement function with respect to the state evaluated at
     the state provided.
h = [...
    m(1) - x(3)
    m(2) - (x(1)*cos(m(1)) + x(2)*sin(m(1)))
    ];

H_x = [...
```

```
    0,          0,          -1
    -cos(m(1)), -sin(m(1)),  0
    ];

[h(1), h(2), isRNegated] = normalizeLineParameters(h(1), h(2))
    ;

if isRNegated
    H_x(2, :) = - H_x(2, :);
end
```

Listing 2: measurementFunction

The `filterStep` function was implemented as follows:

```
function [x_posteriori, P_posteriori, x_priori] = filterStep(x
    , P, u, Z, R, M, k, g, b)
% returns an a posteriori estimate of the state and its
    covariance

% propagate the state
Q = k*diag(abs(u));

[x_priori, F_x, F_u] = transitionFunction(x, u, b);
P_priori = F_x * P * F_x' + F_u * Q * F_u';

if size(Z,2) == 0
    x_posteriori = x_priori;
    P_posteriori = P_priori;
    return;
end

[v, H, R] = associateMeasurements(x_priori, P_priori, Z, R, M,
    g);

y = reshape(v, [], 1);
H = reshape(permute(H, [1,3,2]), [], 3);
R = blockDiagonal(R);

% update state estimates
S = H * P_priori * H' + R;
K = P_priori * (H' / S);

P_posteriori = (eye(size(P_priori)) - K*H) * P_priori;
x_posteriori = x_priori + K * y;
```

Listing 3: filterStep

The `transitionFunction` was implemented as follows:

```
function [f, F_x, F_u] = transitionFunction(x,u, b)
% predicts the state x at time t given the state at time t-1
    and the input u at time t. F_x denotes the Jacobian of the
    state transition function with respect to the state
    evaluated at the state and input provided. F_u denotes the
    Jacobian of the state transition function with respect to
    the input evaluated at the state and input provided.

f = x + [ ...
    (u(1)+u(2))/2 * cos(x(3) + (u(2)-u(1))/(2*b) )
    (u(1)+u(2))/2 * sin(x(3) + (u(2)-u(1))/(2*b) )
    (u(2)-u(1))/b ...
    ];

F_x = [...
    1, 0, -sin(x(3) - (u(1) - u(2))/(2*b))*(u(1)/2 + u(2)/2)
    0, 1,  cos(x(3) - (u(1) - u(2))/(2*b))*(u(1)/2 + u(2)/2)
    0, 0,  1 ...
    ];

F_u = [...
    cos(x(3) - (u(1) - u(2))/(2*b))/2 + (sin(x(3) - (u(1) - u
        (2))/(2*b))*(u(1)/2 + u(2)/2))/(2*b), cos(x(3) - (u(1)
        - u(2))/(2*b))/2 - (sin(x(3) - (u(1) - u(2))/(2*b))*(u
        (1)/2 + u(2)/2))/(2*b)
    sin(x(3) - (u(1) - u(2))/(2*b))/2 - (cos(x(3) - (u(1) - u
        (2))/(2*b))*(u(1)/2 + u(2)/2))/(2*b), sin(x(3) - (u(1)
        - u(2))/(2*b))/2 + (cos(x(3) - (u(1) - u(2))/(2*b))*(u
        (1)/2 + u(2)/2))/(2*b)
    -1/b,

        1/b ...
    ];
```

Listing 4: transitionFunction

The `associateMeasurements` function was implemented as follows:

```
function [v, H, R] = associateMeasurements(x, P, Z, R, M, g)
% returns a set of innovation vectors and associated jacobians
     and measurement covariances by matching line features by
    Mahalanobis distance.

% evaluate distances of measurements to the map
nMeasurements = size(Z,2); nMapEntries = size(M,2);
d = zeros(nMeasurements, nMapEntries);
v = zeros(2, nMeasurements * nMapEntries);
```

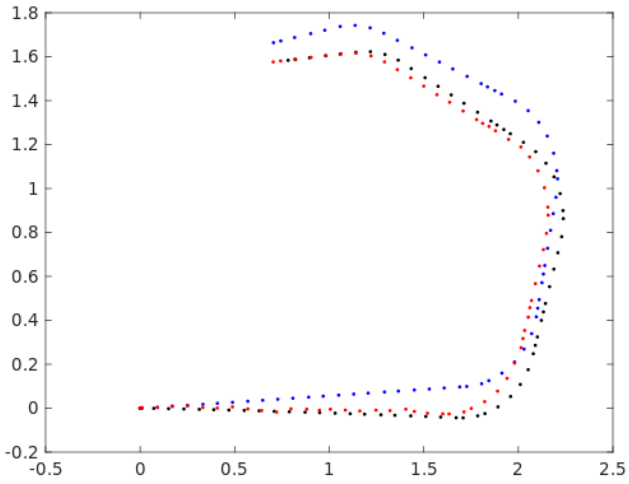
```
H = zeros(2, 3, nMeasurements * nMapEntries);
for i = 1 : nMeasurements
    for j = 1 : nMapEntries
        [z_priori, H(:, :, j + (i-1) * nMapEntries)] =
            measurementFunction(x, M(:,j));
        v(:,j + (i-1) * nMapEntries) = Z(:,i) - z_priori;
        W = H(:, :, j + (i-1) * nMapEntries) * P * H(:, :, j +
             (i-1) * nMapEntries)' + R(:,:,i);
        d(i,j) = v(:,j + (i-1) * nMapEntries)' * inv(W) * v(:,
            j + (i-1) * nMapEntries);
    end
end

% line feature matching; association of each measurement to
    the map point with minimal distance
[minima, map_index] = min(d');
[measurement_index] = find(minima < g^2);
map_index = map_index(measurement_index);

v = v(:, map_index + (measurement_index-1)*nMapEntries);
H = H(:, :, map_index + (measurement_index-1)*nMapEntries);
R = R(:, :, measurement_index);
```

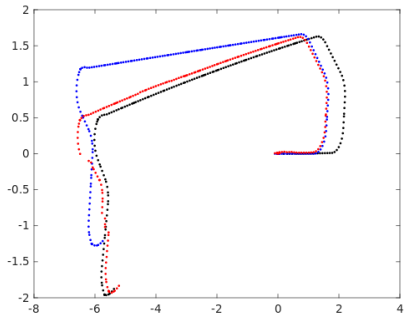
Listing 5: associateMeasurements

Running the `turtlebotEKFLocalization` function for the default of 30 seconds gives the following plots:

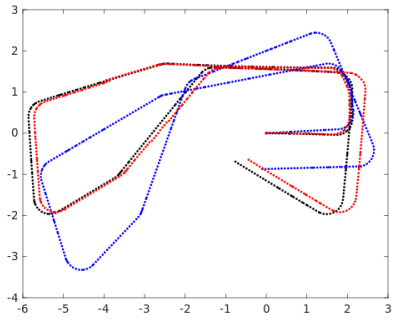


Here, the black dots show the ground truth, red dots show the EKF pose estimate, and blue dots are the odometric calculations resulting from ground truth.

On increasing the times to 100, and 200 seconds, we get the following plots:



(a) 100 seconds



(b) 200 seconds

Again, the black dots show the ground truth, red dots show the EKF pose estimate, and blue dots are the odometric calculations resulting from ground truth. It is evident that for a shorter time duration, the odometric calculations are closer to the ground truth, but as the time duration increases, a significant drift is observed. The EKF pose estimate, however, is much closer to the ground truth for shorter, and larger time durations. We do observe some drift in the EKF pose estimate, but it is much less than the odometric calculations, and it eventaully converges to the ground truth again.

Problem 5 [20 Points]

Solution:

Lyeba Abid

(a) I spent approximately 7 hours working on this homework. We engaged in group discussions to collaboratively tackle the problems.

(b) I took responsibility for solving questions 2 and 3. The group collectively discussed all the problems, ensuring a comprehensive understanding.

(c) My advice for future students working on this homework is to initiate the task early. Starting early provides ample time for thorough exploration and understanding of the material, ensuring a more comprehensive and well-thought-out submission.

(d) Reflecting on my learning, I delved into the intricacies of the Kalman filter and Markov chain localization. This knowledge significantly contributes to my understanding of robotics. I have successfully achieved the outcomes set at the beginning of the assignment. However, I acknowledge that there are still aspects I'd like to explore further. To achieve this, I plan to conduct additional research and seek clarification on any lingering questions. Engaging in more practical applications of these concepts will enhance my skills in robot development. The collaborative effort within the group was beneficial, and I intend to continue participating in such group discussions to expand my knowledge base.

Ali Muhammad

(a) I spent about 6-7 hours on this assignment.

(b) We both went over the assignment together and discussed the problems and solutions. I took Problem 1, and 4.

(c) Start the homework as early as you can, it'll take time and it will be confusing a lot. Go to the instructor to clear the confusions. You will question your own self a lot, but don't give up.

(d) Reflecting on my own learning, I used basic high school physics equations to develop a discrete time motion model for a robot in a 1D plane, and then use those to develop a matrix to calculate the covariance of the robot's position and velocity. In addition to that, I found the uncertainty after one time step, and saw how that covariance / uncertainty would accumulate over time for large values of time (more steps) and how that would affect our uncertainty in the robot's position and velocity, based on some small variance in the commanded acceleration. I aim to explore this further for a 2D plane. Moreover, I was able to implement the EKF-based landmark localization.I aim to explore this further with the Unscented Kalman Filer as well.