EECS 568 (Problem Set-1).

TASK-1.

A.) $p(x/y,z) = \frac{p(y/x,z)\,p(x/z)}{p(y/z)}$.

using Bayes rule, we know that

$p(x/y) = \frac{p(y/x)\times p(x)}{p(y)} = \frac{p(x,y)}{p(y)}$.

$p(y/x) = \frac{p(x/y)p(y)}{p(x)} = \frac{p(x,y)}{p(x)}$.

similarly, for 3 variables, we get

$p(x,y,z) = p(x/y,z)\times p(y,z)$ — (i).

$= p(y/x,z) \times p(x,z)$ — (ii).

equating (i) & (ii), we get

$p(x/y,z)\times p(y,z) = p(y/x,z)\times p(x,z)$ — (iii)

$p(y,z) = p(y/z)\times p(z)$ — (4)

$p(x,z) = p(x/z)\times p(z)$ — (5).

using (4) & (5), in (3), we get.

$p(x/y,z)\,p(y/z)\,p(z) = p(y/x,z)\,p(x/z)\,p(z)$

$\Rightarrow p(x/y,z) = p(y/x,z)\times p(x/z)/p(y/z)$.

B) Given that $P(C) = \frac{1}{100}$ $P(NC) = \frac{99}{100}$.

20% false positive.
10% false negative.

$P(P/NC) = \frac{20}{100}$, $P(N/NC) = \frac{80}{100}$.

$P(N/C) = 10/100$ $P(P/C) = 90/100$.

Since one tested positive, lets find the probability of actual cancer, given the test resulted on true on P.

$P(C/P) = ?$

using Bayes rule & total probability, we find that

$$P(C/P) = \frac{P(P/C) \times P(C)}{P(P)}.$$

$$P(P) = P(P/C)\,P(C) + P(P/NC) \times P(NC).$$

Thus,

$$P(P) = \frac{20}{100} \times \frac{99}{100} + \frac{90}{100} \times \frac{1}{100} = 0.207.$$

Thus,

$$P(C/P) = \frac{9/10 \times 1/100}{0.207} = 0.0434 \Rightarrow 4.3\%.$$

3.

Let us denote $P(P/Nc) = a$ & $P(N/c) = b$.

Thus, we have

$$P(P) = a \times \frac{99}{100} + (1-b) \times \frac{1}{100}$$

$$P(C|P) = \frac{(1-b) \times 1/100}{\frac{99a}{100} + (1-b)\frac{1}{100}} = \frac{(1-b)}{99a + (1-b)}.$$

Taking partial derivative of $P(C|P)$, w.r.t a, b, we get

$$\frac{\partial P(C|P)}{\partial a} = \frac{(1-b)}{99a + (1-b)} \times 99 \quad \text{— (i)}.$$

$$\frac{\partial p(C|P)}{\partial b} = \frac{(1-b)}{99a + (1-b)} \times (-1) \quad \text{— (ii)}$$

We see that (i) changes more w.r.t "a" when compared to b.

Thus, investment must be made to reduce "a" ie false positive.

4

c.) It is given that:

$$p\left(x_{t+1} = \text{white} \,/\, \begin{matrix} x_t = \text{blank} \\ u_t = \text{paint} \end{matrix}\right) = 0.9.$$

$$p\left(x_{t+1} = \text{blank} \,/\, \begin{matrix} x_t = \text{blank} \\ u_t = \text{paint} \end{matrix}\right) = 1 - 0.9 = 0.1$$

$p(z = \text{wh} \,/\, x_t = \text{blank}) = 0.2$

$p(z = \text{wh} \,/\, x_t = \text{wh}) = 0.7$

$p(z = \text{blank} \,/\, x_t = \text{blank}) = 0.8$

$p(z = \text{blank} \,/\, x_t = \text{wh}) = 0.3$

Assume the prior probability of the state to be uniform

$p(x_t = \text{blank}) = 0.5 \quad p(x_t = \text{white}) = 0.5$.

Since the manipulate has already decided to paint the door, we see that using Bayes filter, we get

for $x_{t+1} = \text{blank}$ & $x_t = \text{blank}$:

$$\overline{bel}(x_{t+1} = \text{blank}) = \sum p(x_{t+1} = \text{blank} \mid u_t = \text{paint}) \times bel(x_t)$$

$$= \{\text{for } x_t = \text{blank} : 0.1 \times 0.5\} + \{\text{for } x_t = \text{white} : 0 \times 0.5\}.$$

$$= 0.1 \times 0.5$$

5

$$\overline{bel}(x_{t+1} = \text{color}) = \sum_{x_t} p(x_{t+1} = \text{col} \mid u_t = \text{paint}) \times bel(x_t)$$

$$= \{\text{for } x_t = \text{blank} : 0.9 \times 0.5\} + \{\text{for } x_t = \text{colored} : 1.0 \times 0.5\}.$$

$$= 0.95$$

Applying the update stage of Bayes filter, we get

$$bel(x_{t+1} = \text{blank}) = \eta \times p(z_t = \text{col} \mid x_{t+1} = \text{blank}) \times \bar{p}(\text{blank}).$$

$$= \eta \times 0.2 \times 0.05$$

$$bel\ \eta = \frac{1}{0.2 \times 0.05 + 0.7 \times 0.95} = 1.48148.$$

Thus, $bel(x_{t+1} = \text{blank}) = \eta \times 0.2 \times 0.05$

$$= 0.0148$$

$$= 1.48\%.$$

TASK -2

B.) Given that $\mu = \begin{bmatrix} 10 \\ 0 \end{bmatrix}$ & $\Sigma = \begin{bmatrix} 0.5^2 & 0 \\ 0 & 0.25^2 \end{bmatrix}$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix}.$$

If we represent the above as shown below, we get

$$\begin{bmatrix} x \\ y \end{bmatrix} = g\left(\begin{bmatrix} r \\ \theta \end{bmatrix}\right)$$

Linearising g at μ, we get.

$$\begin{bmatrix} x \\ y \end{bmatrix} \approx g\left(\begin{bmatrix} \mu_r \\ \mu_\theta \end{bmatrix}\right) + \frac{\partial g}{\partial \begin{bmatrix} r \\ \theta \end{bmatrix}} \begin{bmatrix} r-\mu_r \\ \theta-\mu_\theta \end{bmatrix}.$$

$$= g\left(\begin{bmatrix} 10 \\ 0 \end{bmatrix}\right) + \frac{\partial g}{\partial \begin{bmatrix} r \\ \theta \end{bmatrix}}$$

$$\frac{\partial g}{\partial \begin{bmatrix} r \\ \theta \end{bmatrix}} = \begin{bmatrix} \frac{\partial g[0]}{\partial r} & \frac{\partial g[0]}{\partial \theta} \\ \frac{\partial g[1]}{\partial r} & \frac{\partial g[1]}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{bmatrix}.$$

$\Rightarrow$ we get

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix} + \begin{bmatrix} \cos\theta\,(r-\mu_r) & -r\sin\theta\,(\theta-\mu_\theta) \\ \sin\theta\,(r-\mu_r) & +r\cos\theta\,(\theta-\mu_\theta) \end{bmatrix}$$

taking r, θ, as $10, 0$ for $\frac{\partial g}{\partial \begin{bmatrix} r \\ \theta \end{bmatrix}}$, we get

$$\frac{\partial g}{\partial \begin{bmatrix} r \\ \theta \end{bmatrix}} = \begin{bmatrix} \cos(0) & -0 \times \sin 0 \\ \sin(0) & \cos(0) \times 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}.$$

Thus, we get

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix} \begin{bmatrix} r - 10 \\ \theta - 0 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix} + \begin{bmatrix} r - 10 \\ 10\theta \end{bmatrix} = \begin{bmatrix} r \\ 10\theta \end{bmatrix}.$$

Thus, $x = r$ linearisation

$y = 10\theta$.

Thus, $\Sigma_x = \Sigma_r = 0.5^2$

$\Sigma_y = \Sigma_{10\theta} = 10^2 \times \Sigma_\theta = 100 \times 0.25^2$.

Thus, linearised covariance matrix is

$$\Sigma_{xy} = \begin{bmatrix} 0.5^2 & 0 \\ 0 & 100 \times 0.25^2 \end{bmatrix}.$$

TASK 1

D. CODE:

```
close all;
class_1= [0, 3, 6];
class_2= [1, 2, 4, 5, 7, 8, 9];

num_locations = size(class_1, 2) + size(class_2, 2);

landmark_numbers = class_1;

p_landmark_at_class_1 = 0.8;
np_landmark_at_class_1 = 0.2;

p_landmark_at_class_2 = 0.4;
np_landmark_at_class_2 = 0.6;

control_input = [0, 3, 4];
observations = [1, 1, 0];

belief = ones(1, num_locations)*1/num_locations;

num_plots = length(control_input)*2;

for index1 = 1:length(control_input)
    control = control_input(index1);
    observe = observations(index1);

    belief_bar = zeros(1, num_locations);
    for index2 = 1:length(belief)
        belief_bar(mod((index2-1) + control, num_locations)+1) = belief(index2);
    end
```

```
    pl = subplot(num_plots, 1, 2*(index1-1)+1);
    bar(pl, 0:num_locations-1, belief_bar, 'stacked');
    for index2 = 1:length(belief)
        if observe
            if ismember(index2-1, landmark_numbers)
                belief_bar(index2) = p_landmark_at_class_1*belief_bar(index2);
            else
                belief_bar(index2) = p_landmark_at_class_2*belief_bar(index2);
            end
        else
            if ismember(index2-1, landmark_numbers)
                belief_bar(index2) = np_landmark_at_class_1*belief_bar(index2);
            else
                belief_bar(index2) = np_landmark_at_class_2*belief_bar(index2);
            end
        end
    end
    belief = belief_bar/sum(belief_bar);
    pl = subplot(num_plots, 1, 2*(index1-1)+2);
    bar(pl, 0:num_locations-1, belief, 'stacked');
end

figure;
bar(0:num_locations-1, belief);
```

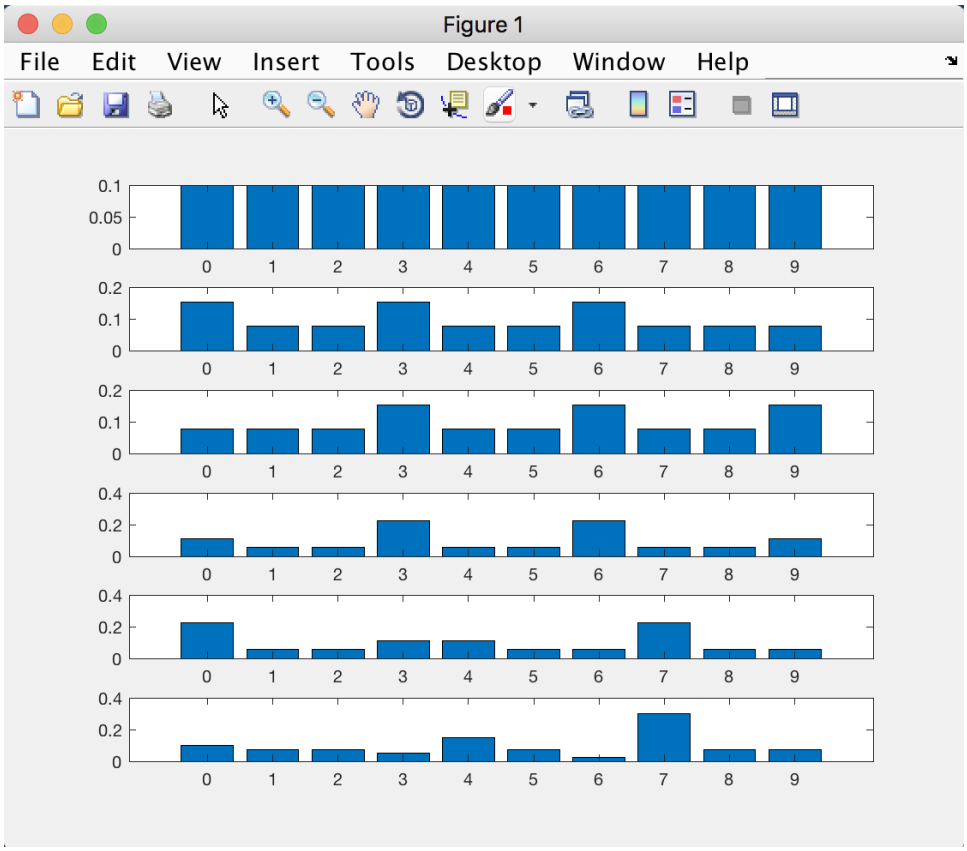


Figure 1
File Edit View Insert Tools Desktop Window Help
0.1
0.05
0
0 1 2 3 4 5 6 7 8 9
0.2
0.1
0
0 1 2 3 4 5 6 7 8 9
0.2
0.1
0
0 1 2 3 4 5 6 7 8 9
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9

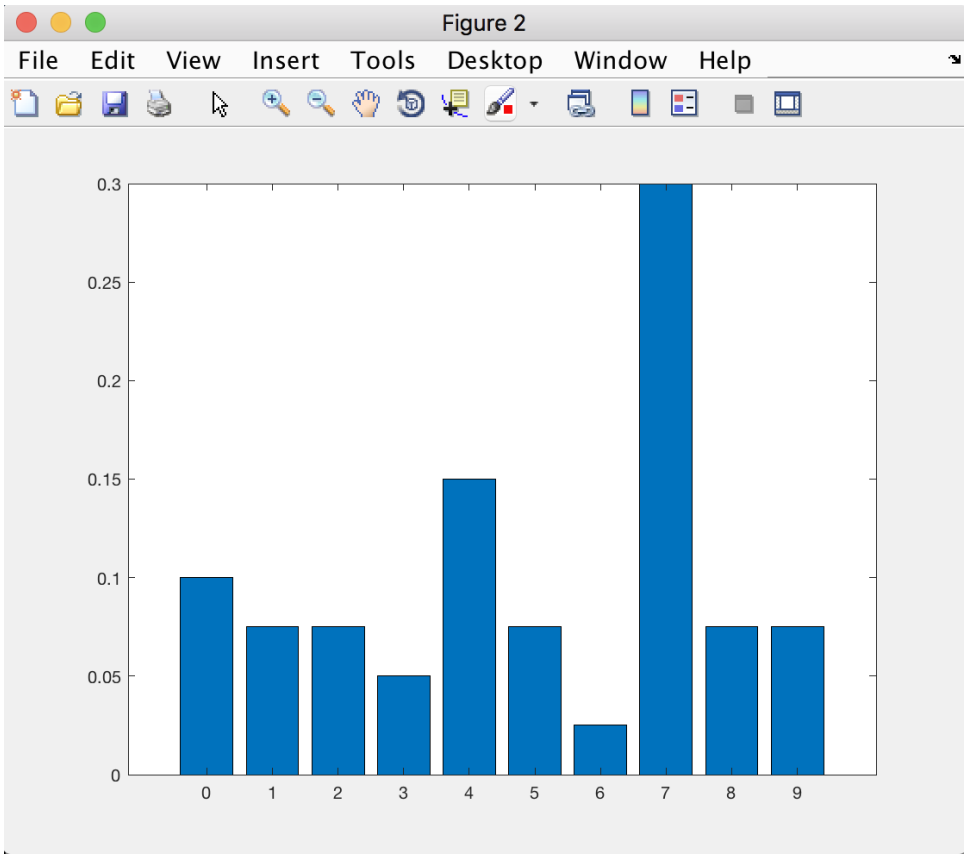


Figure 2
File Edit View Insert Tools Desktop Window Help
0.3
0.25
0.2
0.15
0.1
0.05
0
0 1 2 3 4 5 6 7 8 9

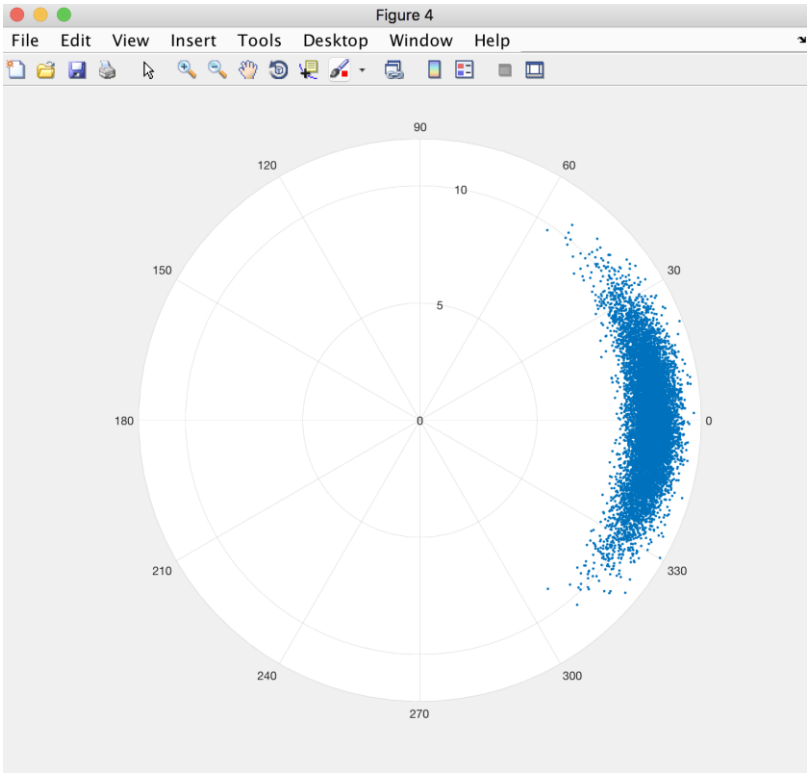
TASK 2

A. CODE:

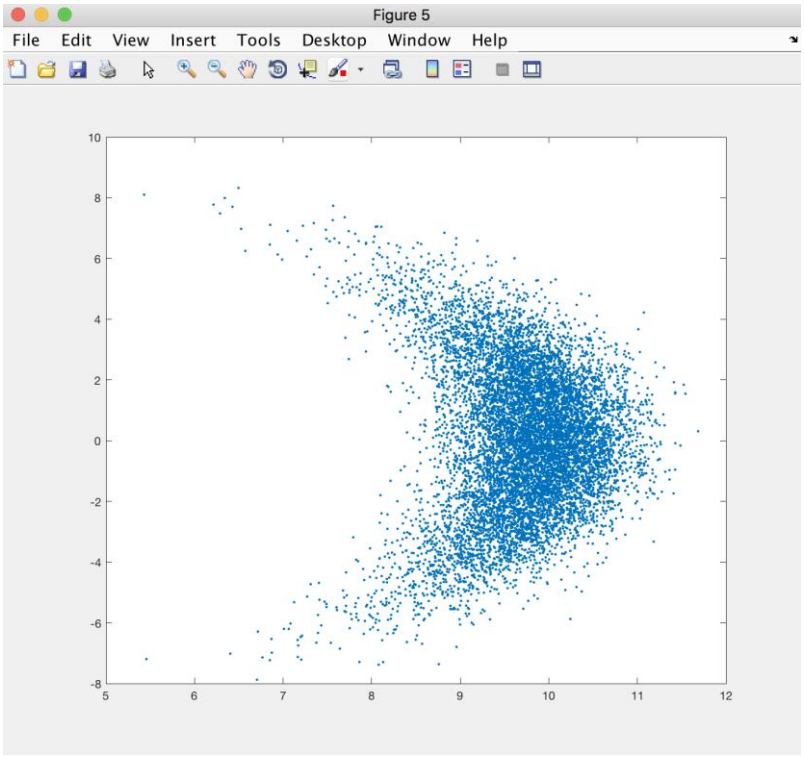
```
% (0, 0)
mean_r = 10.0;
mean_theta = 0.0;
num_points = 10000;
sd_r = 0.5;
sd_theta = 0.25;
r_values = mean_r + sd_r*randn(num_points, 1);
theta_values = mean_theta + sd_theta*randn(num_points, 1);
figure;
polarscatter(theta_values, r_values, '.');
x= r_values.*cos(theta_values);
y = r_values.*sin(theta_values);
figure; plot(x, y, '.');
```

The corresponding plots are as shown below:

The sensor frame space (r, theta)



The cartesian coordiante space(x,y)



B. CODE:

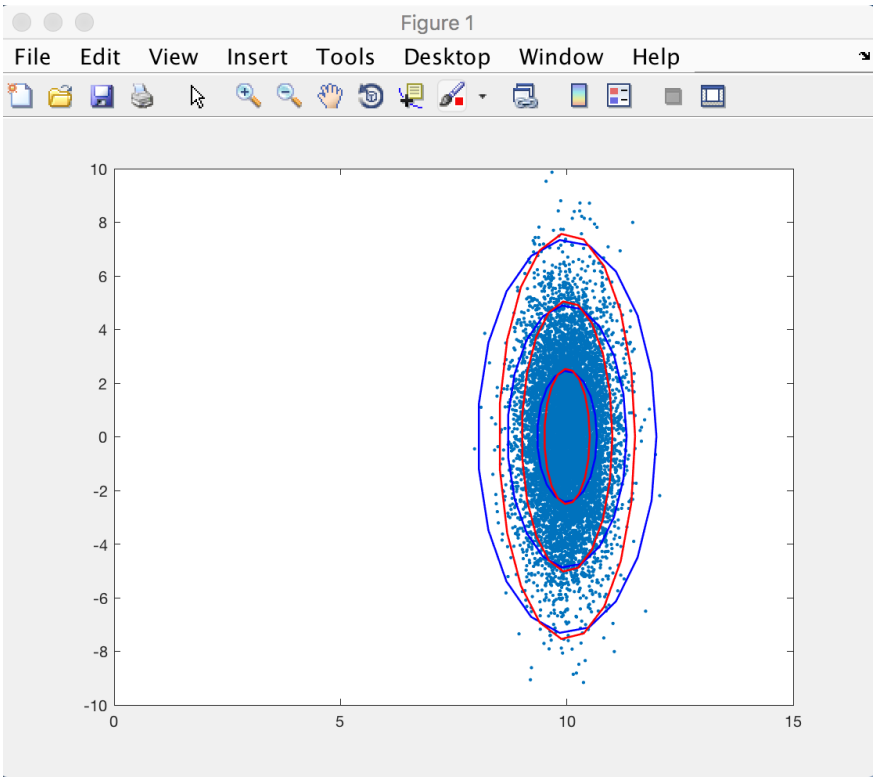
```
%(0,0)

function [x_y_mean, x_y_cov] = mapping(r_theta_mean, r_theta_cov, r_theta_point)

non_linear_at_mean = [r_theta_mean(1)*cos(r_theta_mean(2)); r_theta_mean(1)*sin(r_theta_mean(2))];

% Jacobian of the domain mapping from r-theta to x-y domain

derivative_at_point = [cos(r_theta_mean(2)) -1*r_theta_mean(1)*sin(r_theta_mean(2)); sin(r_theta_mean(2))
r_theta_mean(1)*cos(r_theta_mean(2))];

delta_point = r_theta_point - r_theta_mean;

x_y_mean = non_linear_at_mean + derivative_at_point*delta_point;

x_y_cov = derivative_at_point*r_theta_cov*derivative_at_point';

end

function task2_b

r_theta_mean = [10; 0];

r_theta_cov = [0.5^2, 0; 0, 0.25^2];

[linear_x_y_mean, linear_x_y_cov] = mapping(r_theta_mean, r_theta_cov, r_theta_mean);

display(linear_x_y_cov);

end
```

C. CODE:

```
clear all;

close all;

mean_r = 10.0;

mean_theta = 0.0;

num_points = 10000;

sd_r = 0.5;

sd_theta = 0.25;

for index=1:num_points

   r_values(index) = sample(mean_r, sd_r^2);

   theta_values(index) = sample(mean_theta, sd_theta^2);

end

actual_x_values = r_values.*cos(theta_values);
```

```
actual_y_values = r_values.*sin(theta_values);

actual_var_x = var(actual_x_values);

actual_var_y = var(actual_y_values);

actual_sigma = [actual_var_x, 0; 0, actual_var_y];

for index=1:num_points

   [linear_x_y_mean, linear_x_y_cov] = mapping([mean_r; mean_theta], [sd_r^2 0; 0, sd_theta^2],
[r_values(index); theta_values(index)]);

   x_y_values(:, index) = linear_x_y_mean;

end

linearized_var_x = var(x_y_values(1, :));

linearized_var_y = var(x_y_values(2, :));

linearized_sigma = [linearized_var_x, 0; 0, linearized_var_y];

linearized_mean_x = mean(x_y_values(1, :));

linearized_mean_y = mean(x_y_values(2, :));

figure;

plot(x_y_values(1,:), x_y_values(2,:), '.');

xlim([0 15]);

ylim([-10 10]);

hold on;

draw_ellipse([linearized_mean_x; linearized_mean_y], [actual_var_x, 0; 0, actual_var_y], 1,
'color','blue','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [actual_var_x, 0; 0, actual_var_y], 4,
'color','blue','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [actual_var_x, 0; 0, actual_var_y], 9,
'color','blue','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 1,
'color','red','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 4,
'color','red','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 9,
'color','red','linewidth',1.5);
```

The plot is as shown below:



They do not agree with each other because of linearization of the points about the mean of the polar values. The linearization part moves the points to linear in nature instead of being non linear and this affects the covariance and thus we see the two of them do not agree with each other.

D. CODE:

```
clear all;

close all;

mean_r = 10.0;

mean_theta = 0.0;

num_points = 10000;

sd_r = 0.5;

sd_theta = 0.25;

for index=1:num_points

    r_values(index) = sample(mean_r, sd_r^2);

    theta_values(index) = sample(mean_theta, sd_theta^2);

end

actual_x_values = r_values.*cos(theta_values);

actual_y_values = r_values.*sin(theta_values);

actual_var_x = var(actual_x_values);
```

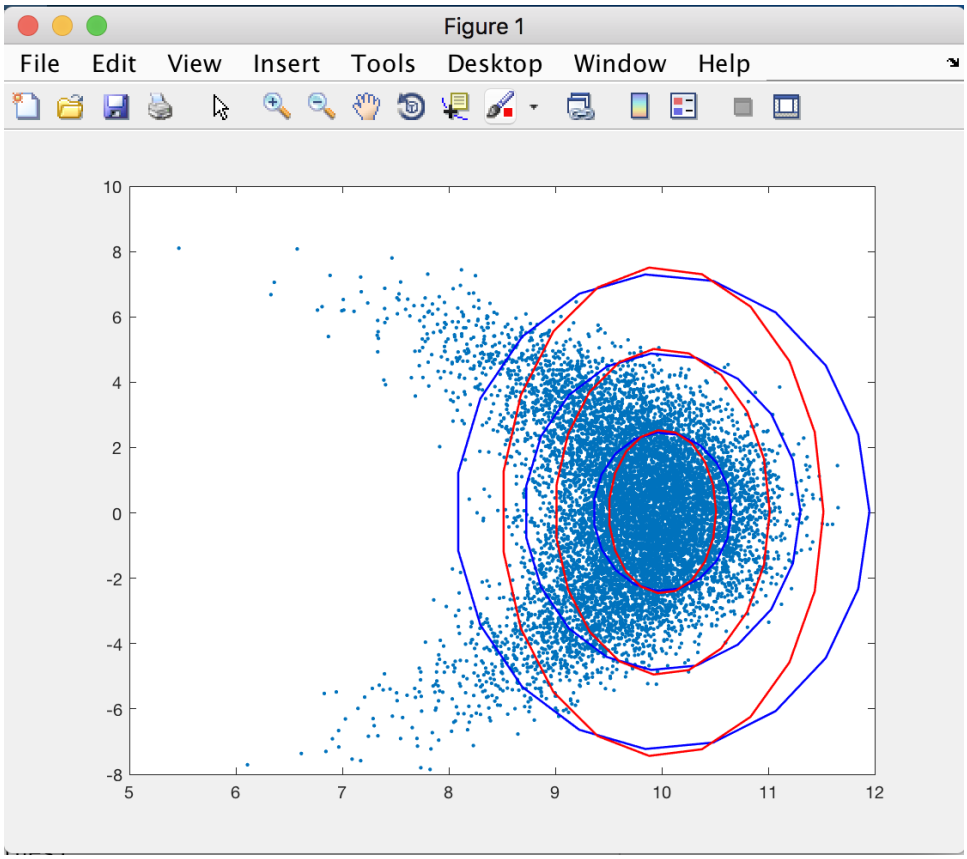
```
actual_var_y = var(actual_y_values);

actual_sigma = [actual_var_x, 0; 0, actual_var_y];

for index=1:num_points

    [linear_x_y_mean, linear_x_y_cov] = mapping([mean_r; mean_theta], [sd_r^2 0; 0, sd_theta^2],
[r_values(index); theta_values(index)]);

    x_y_values(:, index) = linear_x_y_mean;

end

linearized_var_x = var(x_y_values(1, :));

linearized_var_y = var(x_y_values(2, :));

linearized_sigma = [linearized_var_x, 0; 0, linearized_var_y];

linearized_mean_x = mean(x_y_values(1, :));

linearized_mean_y = mean(x_y_values(2, :));

figure;

plot(actual_x_values, actual_y_values, '.');

hold on;

draw_ellipse([linearized_mean_x; linearized_mean_y], [actual_var_x, 0; 0, actual_var_y], 1,
'color','blue','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [actual_var_x, 0; 0, actual_var_y], 4,
'color','blue','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [actual_var_x, 0; 0, actual_var_y], 9,
'color','blue','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 1,
'color','red','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 4,
'color','red','linewidth',1.5);

draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 9,
'color','red','linewidth',1.5);


% Task 2 D

sigma_1_count = 0;

sigma_2_count = 0;

sigma_3_count = 0;

for index = 1:num_points

    distance = mahalanobis_distance(x_y_values(:,index), [linearized_mean_x; linearized_mean_y],
linearized_sigma);
```

```
    if distance <= 1
        sigma_1_count = sigma_1_count + 1;
        sigma_2_count = sigma_2_count + 1;
        sigma_3_count = sigma_3_count + 1;
    elseif distance <= 2
        sigma_2_count = sigma_2_count + 1;
        sigma_3_count = sigma_3_count + 1;
    elseif distance <= 3
        sigma_3_count = sigma_3_count + 1;
    end
end
sigma_1_count = sigma_1_count/num_points;
sigma_2_count = sigma_2_count/num_points;
sigma_3_count = sigma_3_count/num_points;
display(sprintf("Actual 1 sigma values = %f, computed percent = %f", chi2cdf(1,2), sigma_1_count));
display(sprintf("Actual 2 sigma values = %f, computed percent = %f", chi2cdf(4,2), sigma_2_count));
display(sprintf("Actual 3 sigma values = %f, computed percent = %f", chi2cdf(9,2), sigma_3_count));
```

Function for Mahalanobis distance:

```
function distance = mahalanobis_distance(point, mean, variance)
distance = sqrt((point-mean)'*inv(variance)*(point-mean));
end
```

The plot is as shown below:



"Actual 1 sigma values = 0.393469, computed percent = 0.392800"

"Actual 2 sigma values = 0.864665, computed percent = 0.862900"

"Actual 3 sigma values = 0.988891, computed percent = 0.988700"

E.

It is observed that the count matches on low variance and the performance degrades as the value of variance increases. Tested for the value of variance ranging from 0.1 to 0.9.

F. CODE:

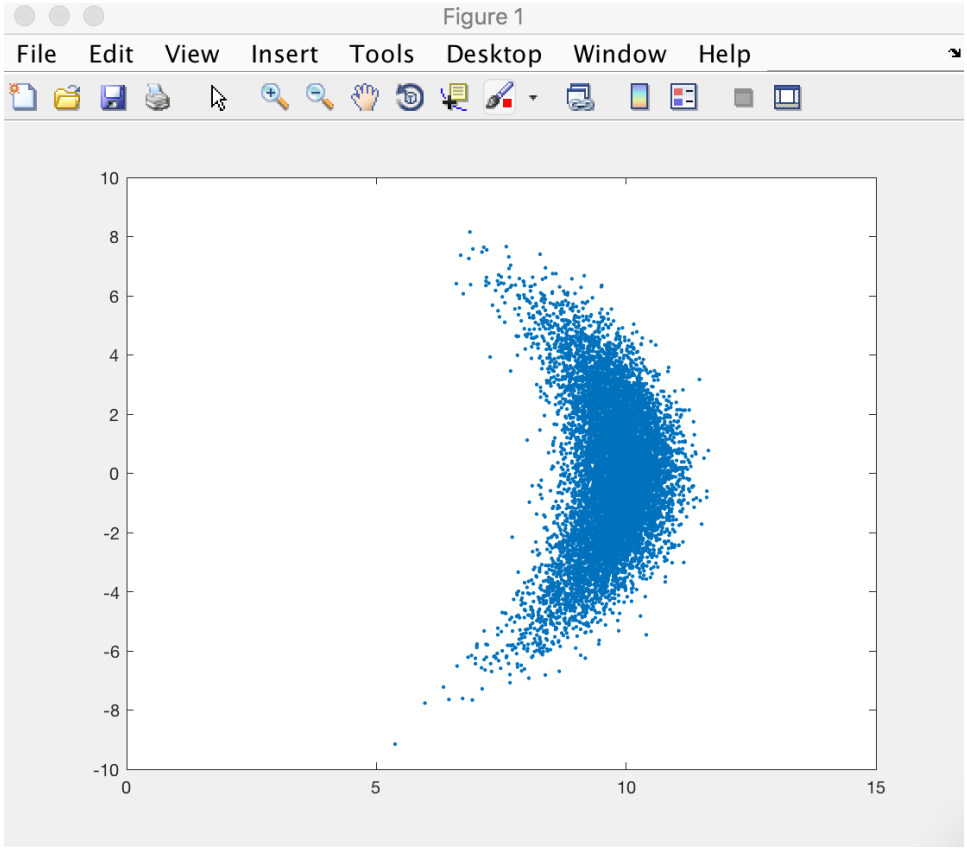
```
% (0,0)

clear all;

close all;

corr_coefficient = [0.1, 0.5, 0.9];

sd_r = 0.5;

sd_theta = 0.25;

for corr_coeff = corr_coefficient
```

```
mean_r = 10.0;
mean_theta = 0.0;
num_points = 10000;
sigma_r_theta = [ sd_r^2, sd_r*sd_theta*corr_coeff; sd_r*sd_theta*corr_coeff, sd_theta^2];
mean_r_theta = [mean_r; mean_theta];
for index=1:num_points
    actual_r_theta_values(:, index) = mvnrnd(mean_r_theta, sigma_r_theta);
end
actual_x_values = actual_r_theta_values(1,:).*cos(actual_r_theta_values(2, :));
actual_y_values = actual_r_theta_values(1,:).*sin(actual_r_theta_values(2, :));
figure;
plot(actual_x_values, actual_y_values, '.');
xlim([0, 15]);
ylim([-10, 10]);
hold on;
actual_mean = [mean(actual_x_values); mean(actual_y_values)];
actual_var_x = var(actual_x_values);
actual_var_y = var(actual_y_values);
actual_sigma = cov(actual_x_values, actual_y_values);
x_y_values = [];
for index=1:num_points
    [linear_x_y_mean, linear_x_y_cov] = mapping([mean_r; mean_theta], sigma_r_theta, actual_r_theta_values(:,
index));
    x_y_values(:, index) = linear_x_y_mean;
end
linearized_var_x = var(x_y_values(1, :));
linearized_var_y = var(x_y_values(2, :));
linearized_sigma = [linearized_var_x, 0; 0, linearized_var_y];


linearized_mean_x = mean(x_y_values(1, :));
linearized_mean_y = mean(x_y_values(2, :));
figure;
```

```
    plot(x_y_values(1,:), x_y_values(2,:), '.');
    hold on;
    xlim([0, 15]);
    ylim([-10, 10]);
    draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 1, 'color','red','linewidth',1.5);
    draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 4, 'color','red','linewidth',1.5);
    draw_ellipse([linearized_mean_x; linearized_mean_y], [linearized_var_x, 0; 0, linearized_var_y], 9, 'color','red','linewidth',1.5);
    draw_ellipse([actual_mean(1); actual_mean(2)], actual_sigma, 1, 'color','blue','linewidth',1.5);
    draw_ellipse([actual_mean(1); actual_mean(2)], actual_sigma, 4, 'color','blue','linewidth',1.5);
    draw_ellipse([actual_mean(1); actual_mean(2)], actual_sigma, 9, 'color','blue','linewidth',1.5);
end
```

The plots are as shown below:

Part 1



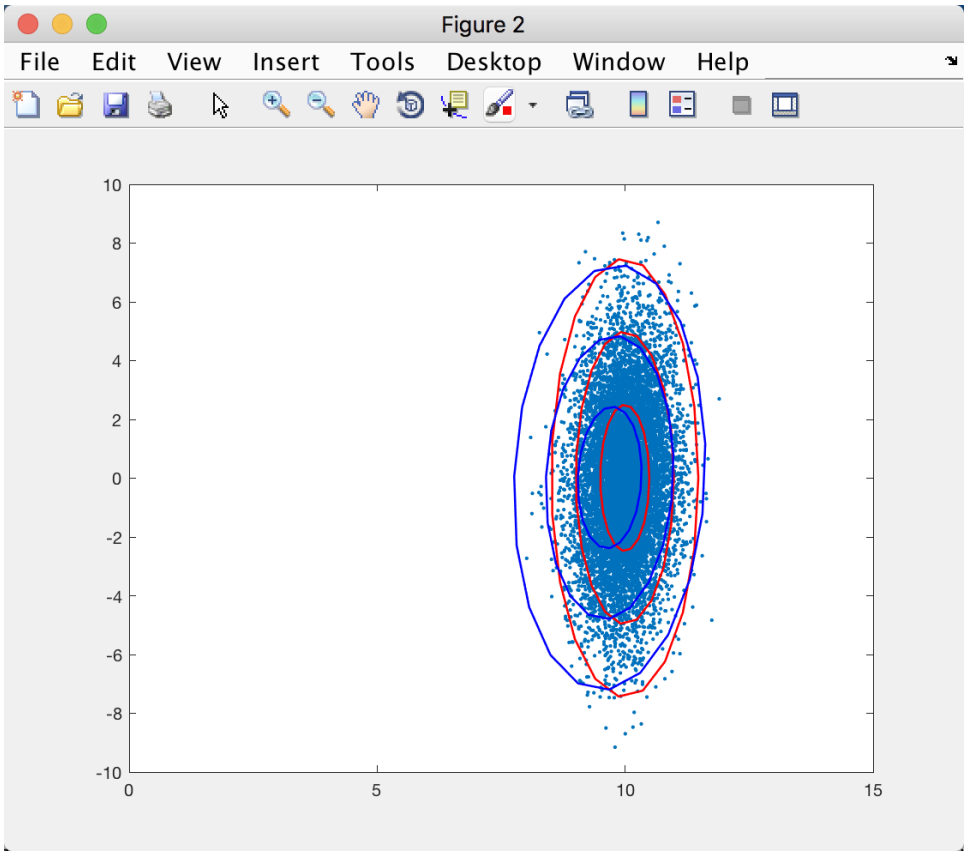


Figure 2
File Edit View Insert Tools Desktop Window Help
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
5
10
15

Part 2

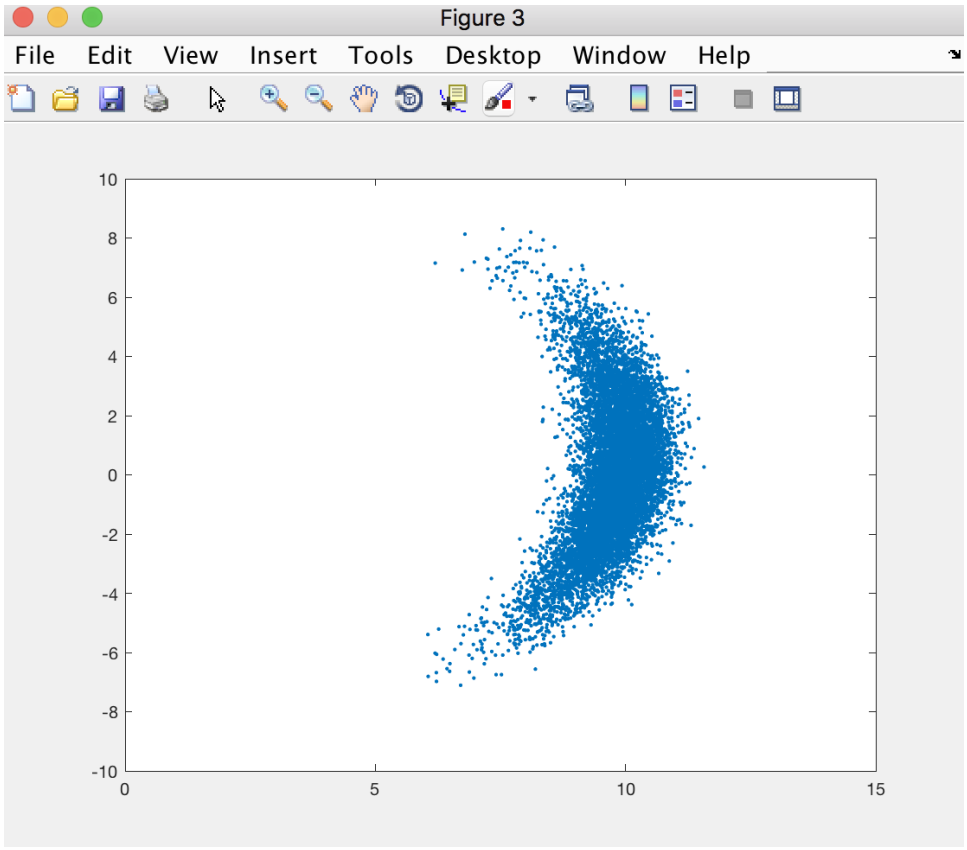


Figure 3
File Edit View Insert Tools Desktop Window Help
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
5
10
15

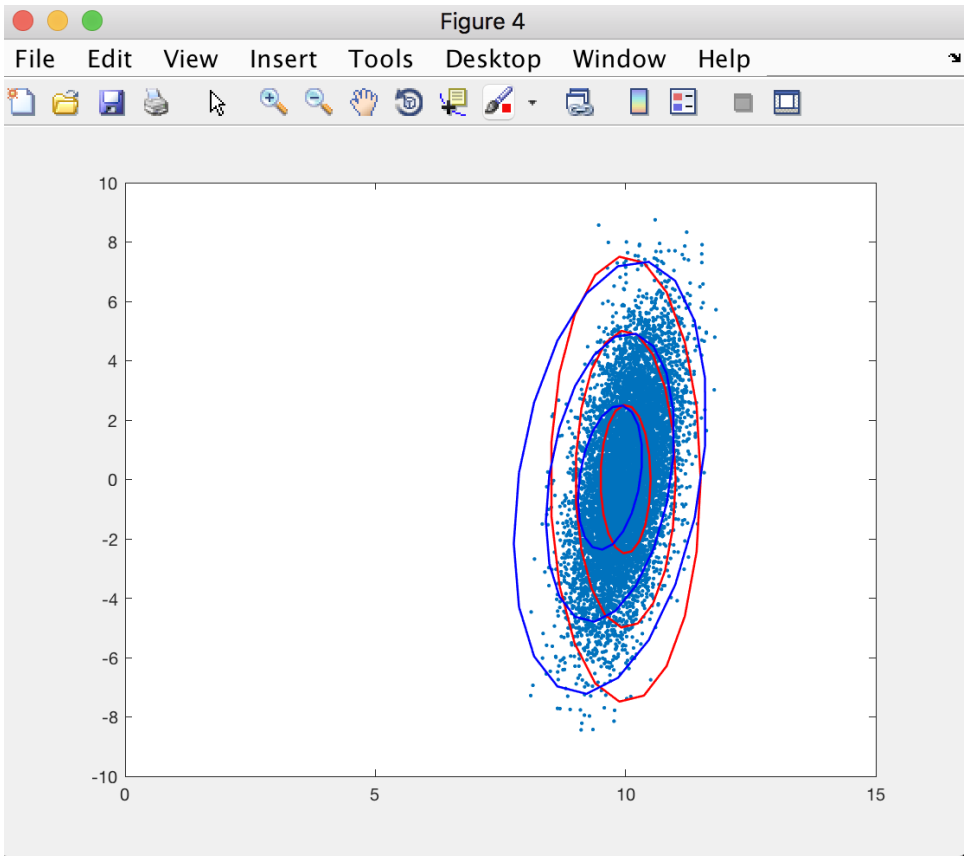


Figure 4
File Edit View Insert Tools Desktop Window Help
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
5
10
15

Part 3

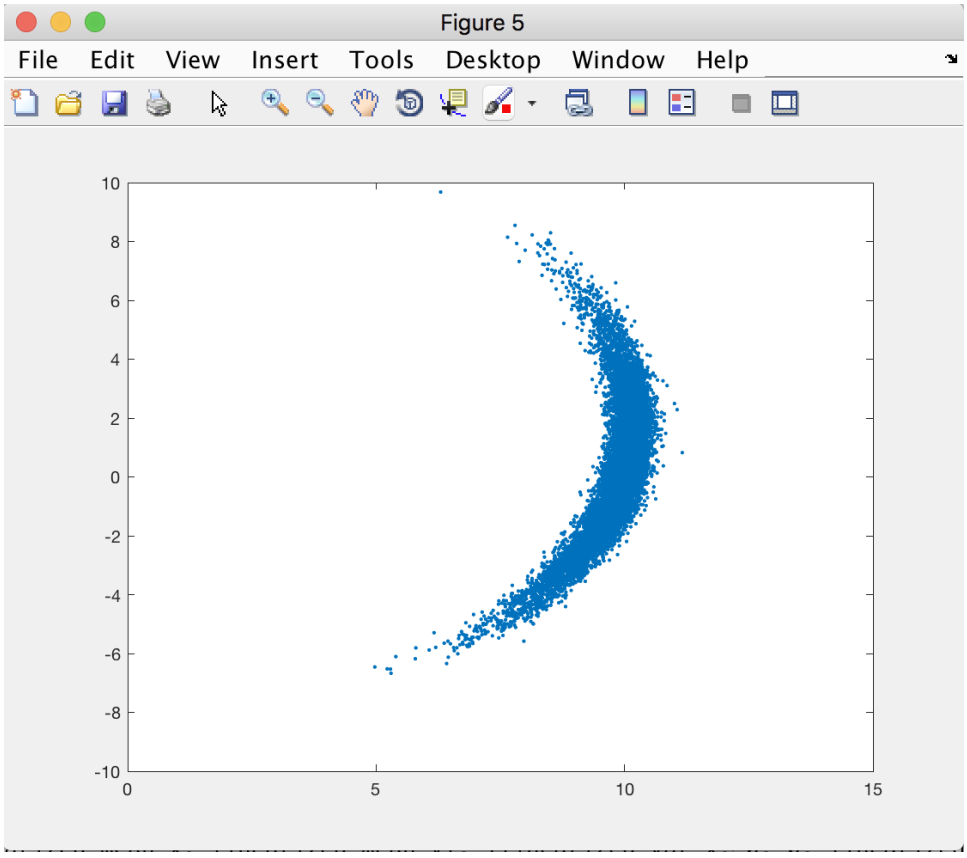


Figure 5
File Edit View Insert Tools Desktop Window Help
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
5
10
15

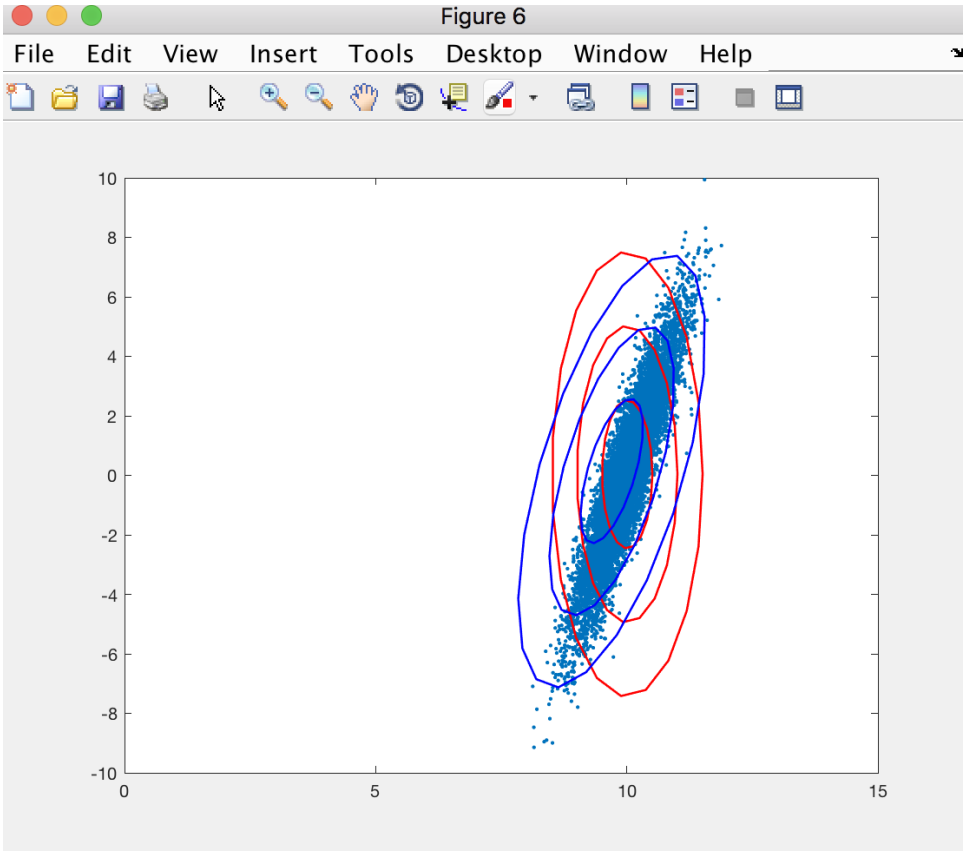
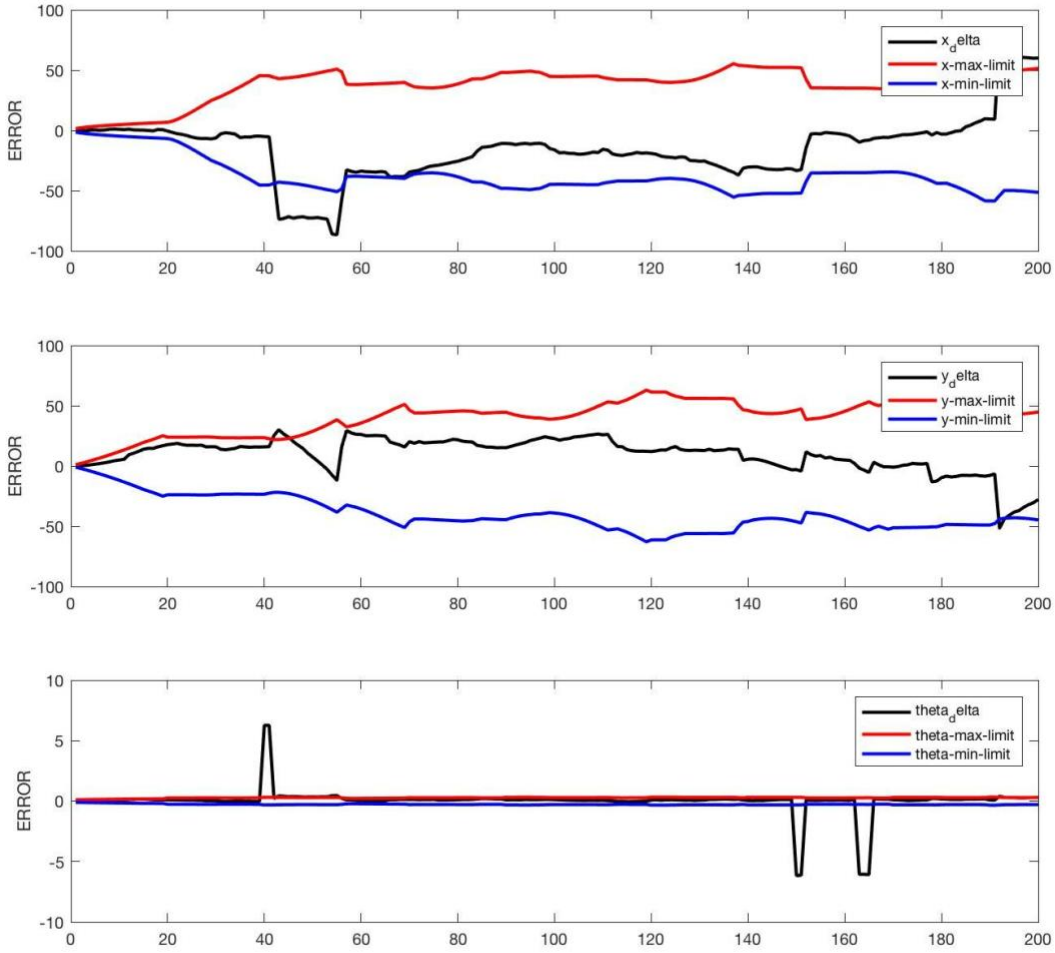


Figure 6
File Edit View Insert Tools Desktop Window Help
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
5
10
15

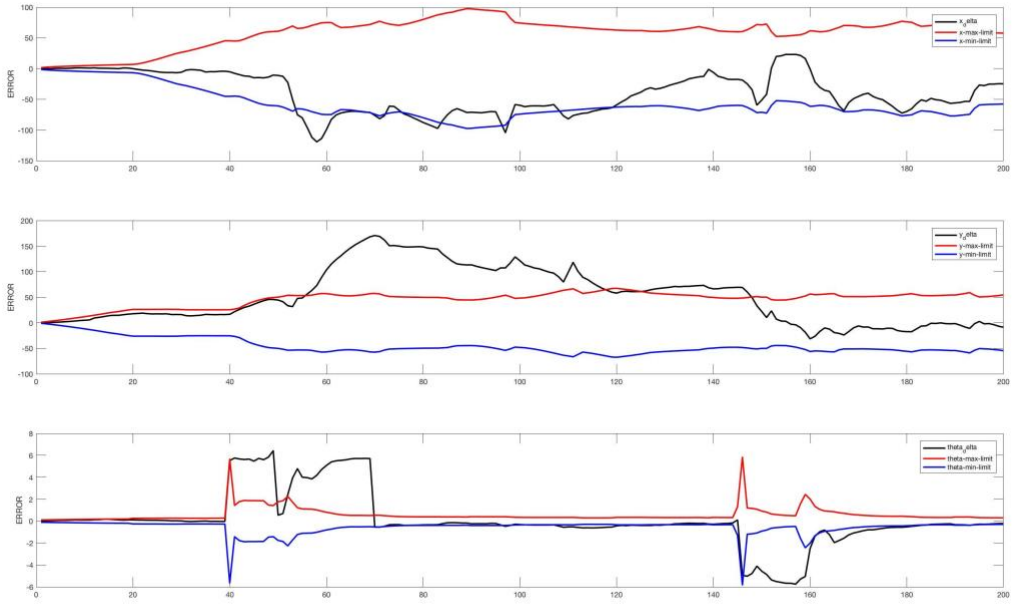
Task #3

1. Check attached videos for demo

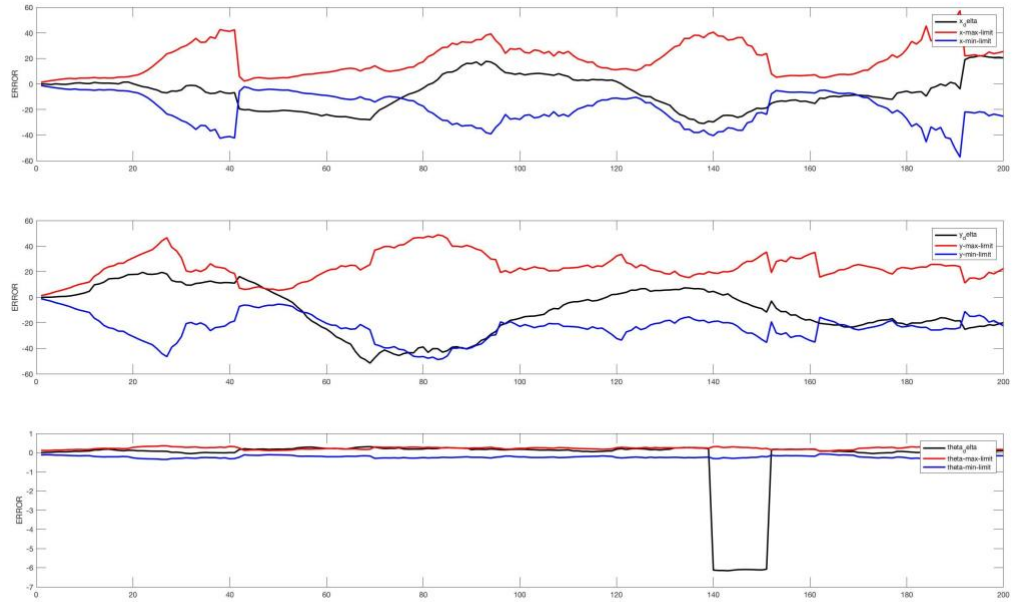
2. The below diagrams show the delta error in x, y and theta for different algorithms EKF:



UKF:


ERROR
x_delta
x-max-limit
x-min-limit
ERROR
y_delta
y-max-limit
y-min-limit
ERROR
theta_delta
theta-max-limit
theta-min-limit

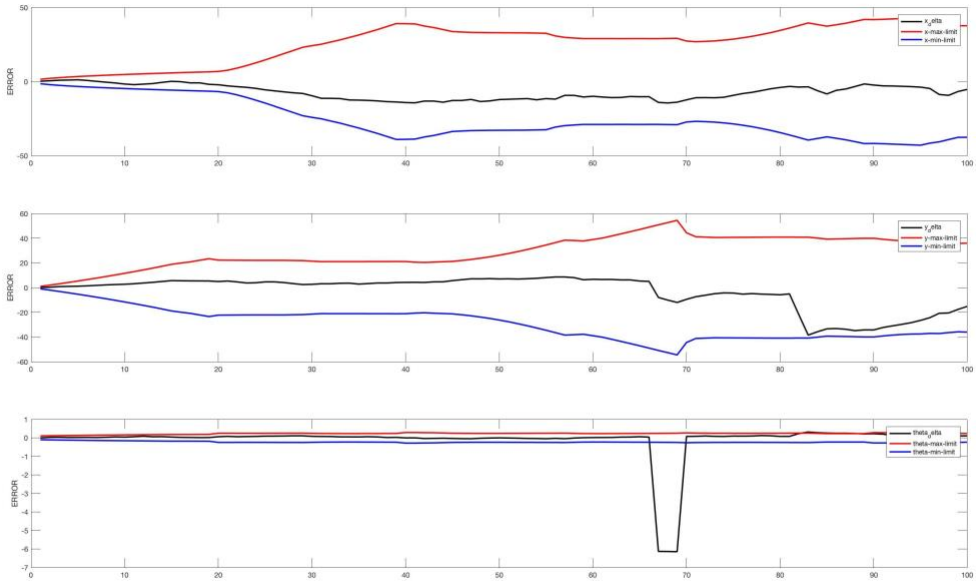
PF:


ERROR
x_delta
x-max-limit
x-min-limit
ERROR
y_delta
y-max-limit
y-min-limit
ERROR
theta_delta
theta-max-limit
theta-min-limit

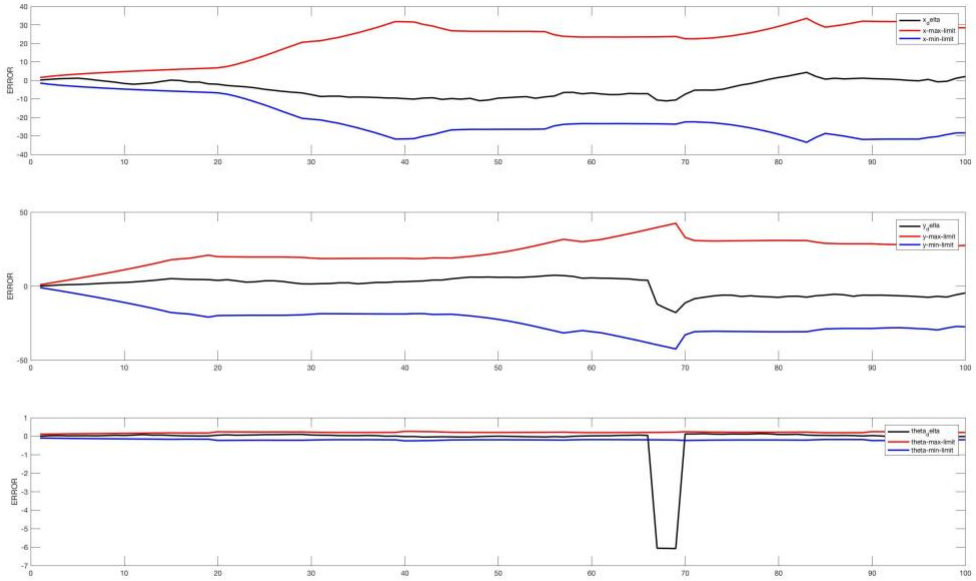
3. The below graphs show the trends as requested in the questions

a. As the motion and sensor noise reduces, we get better and better actual estimates with lower covariance's.
It can also be seen that sensor noise affects the actual estimates much more than motion noise. i.e reducing sensor noise improves faster than reducing motion noise.

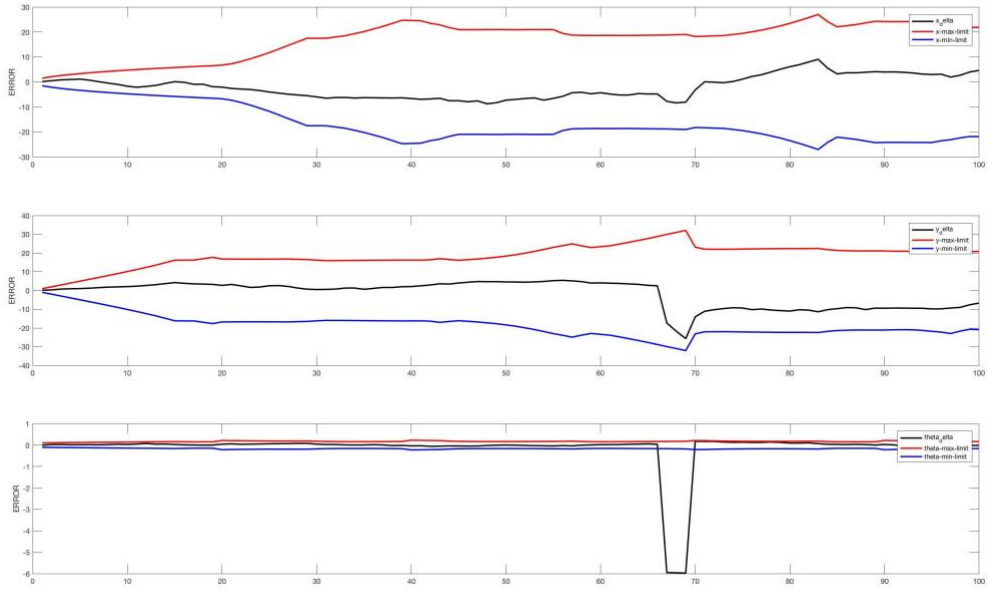
EKF: Using beta = beta/2



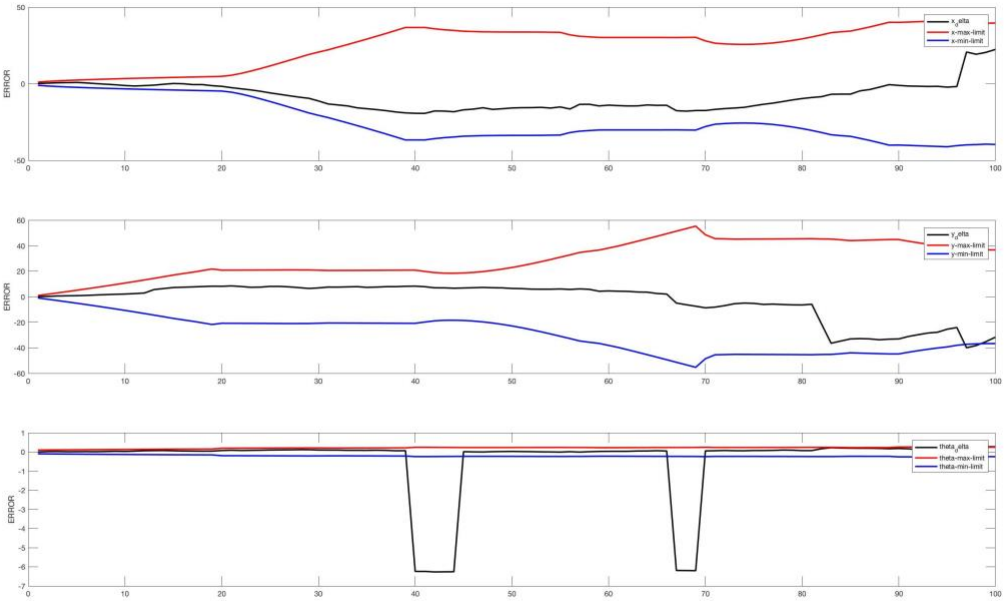
EKF: Using beta = beta/4



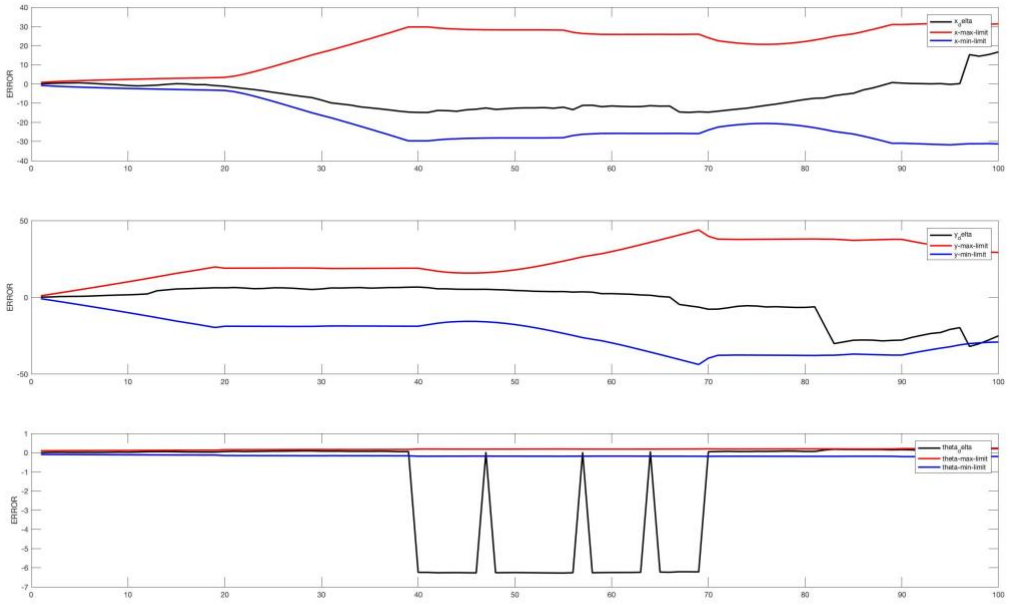
EKF: Using beta = beta/8



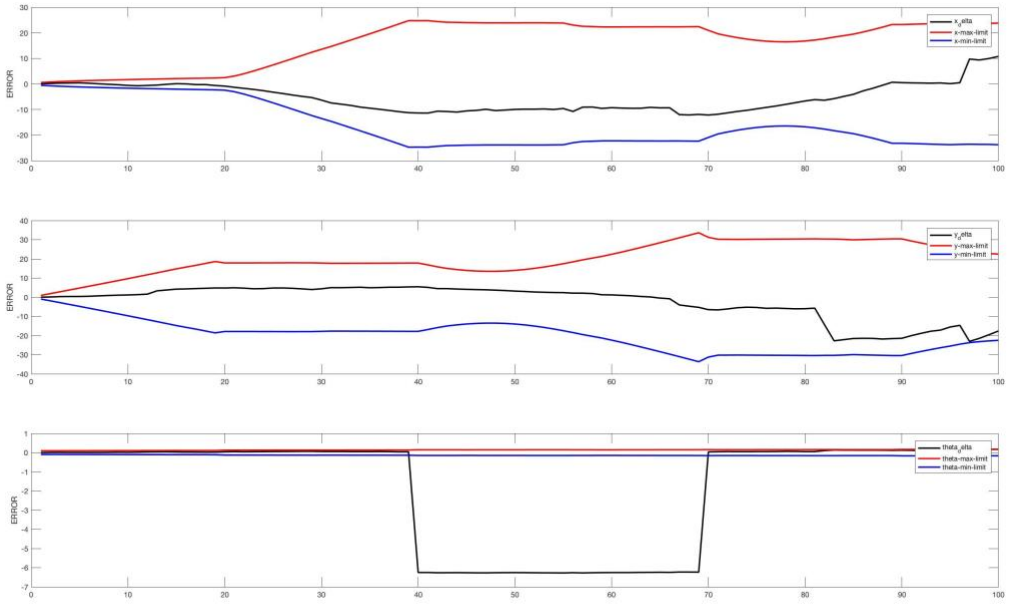
EKF: Using alphas = alphas/2



EKF: Using alphas = alphas/4

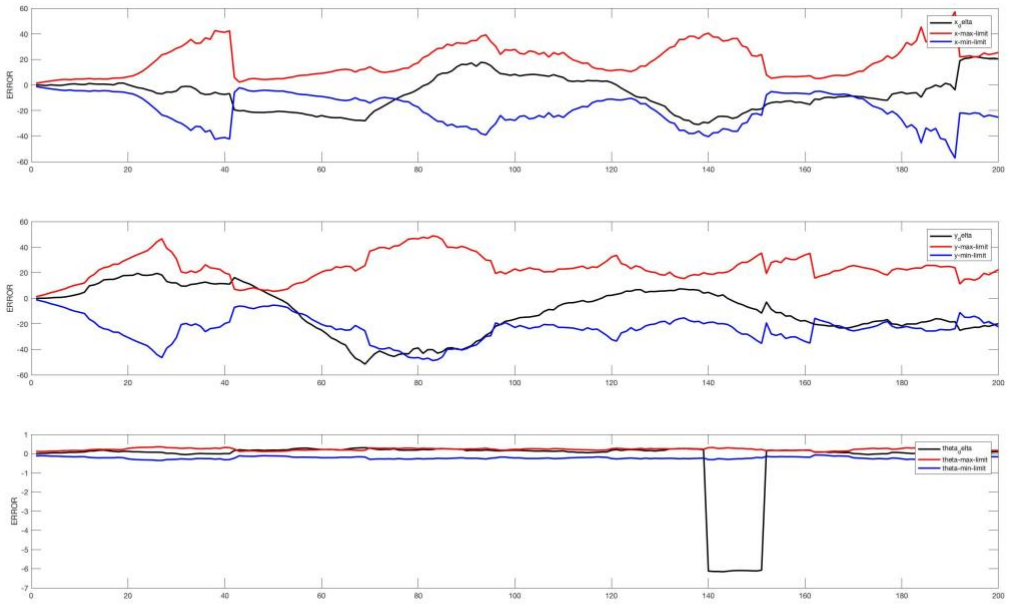


EKF: Using alphas = alphas/8

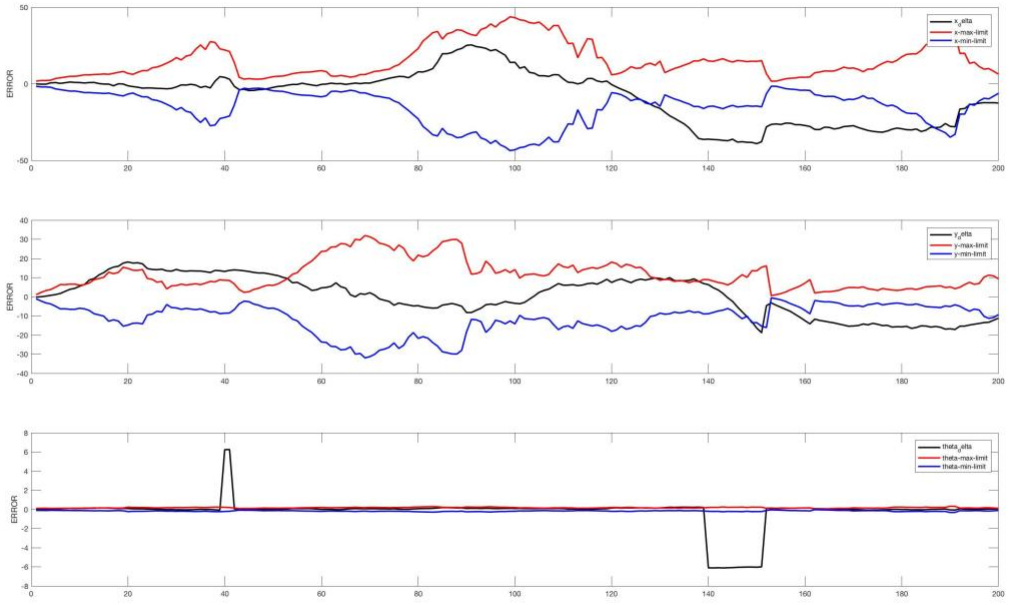


b. Reducing the number of particles increases the variance of the estimate as well as error in the estimated mean. But it can be seen that it significantly improves the running time of the filter.

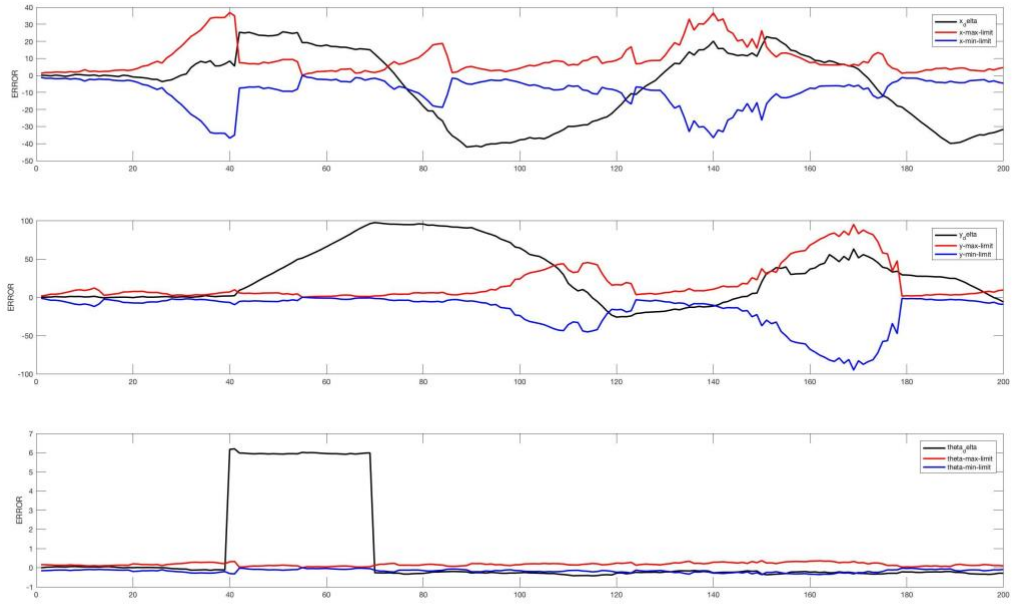
PKF: Using num_particles = 100



PKF: Using num_particles = 50

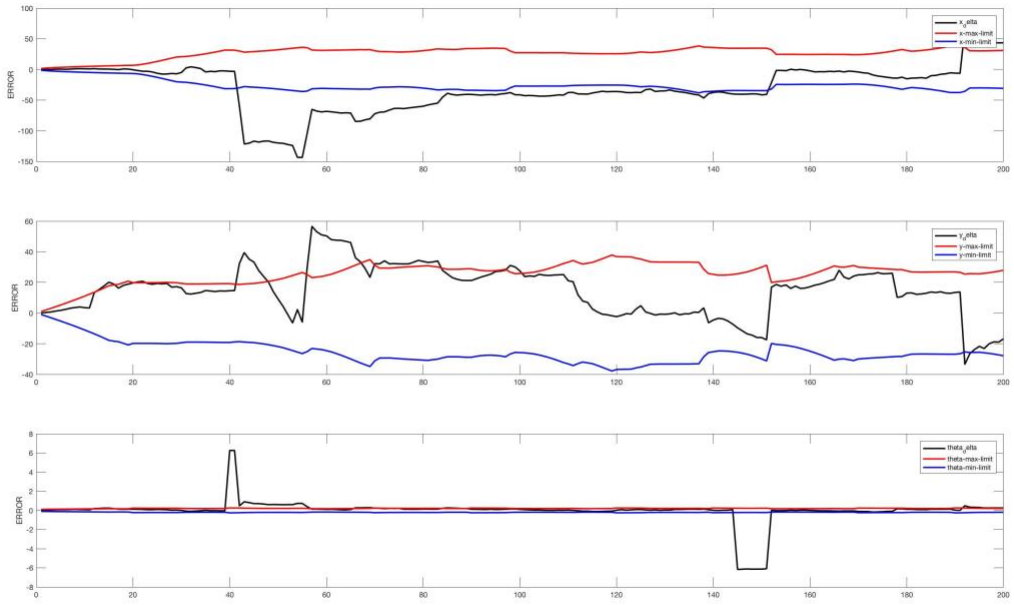


PKF: Using num_particles = 25

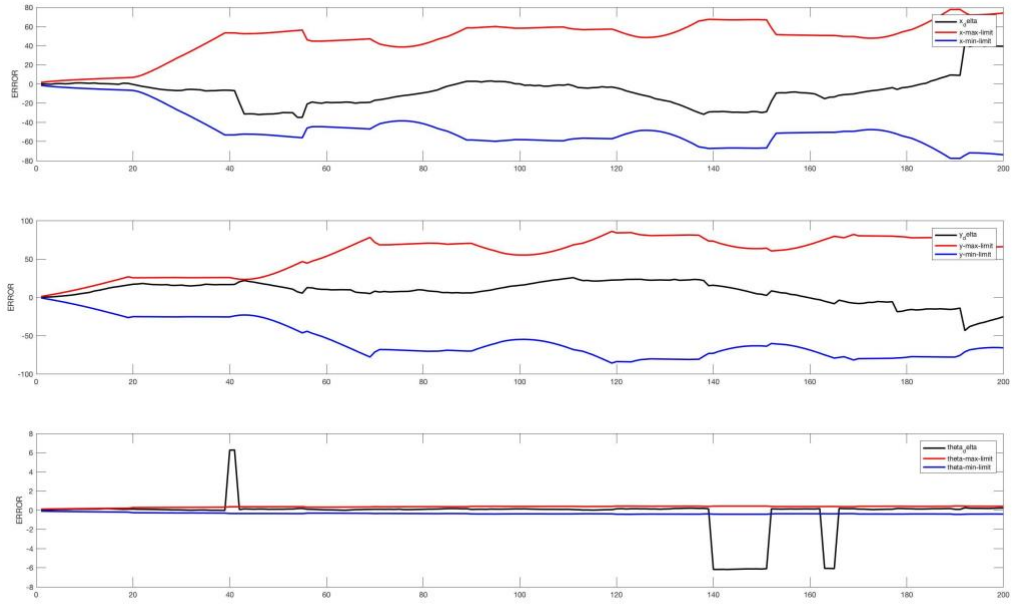


c. Underestimating sensor noise produces worse results when compared to overestimating the sensor noise.

EKF: With underestimated beta



EKF: With overestimated beta



Extra:

Check videos attached for kidnapped robot issue.

We can see that the algorithms handle global localization pretty perfectly, however they are unable to correct for kidnapped robot problem even when noise is redu