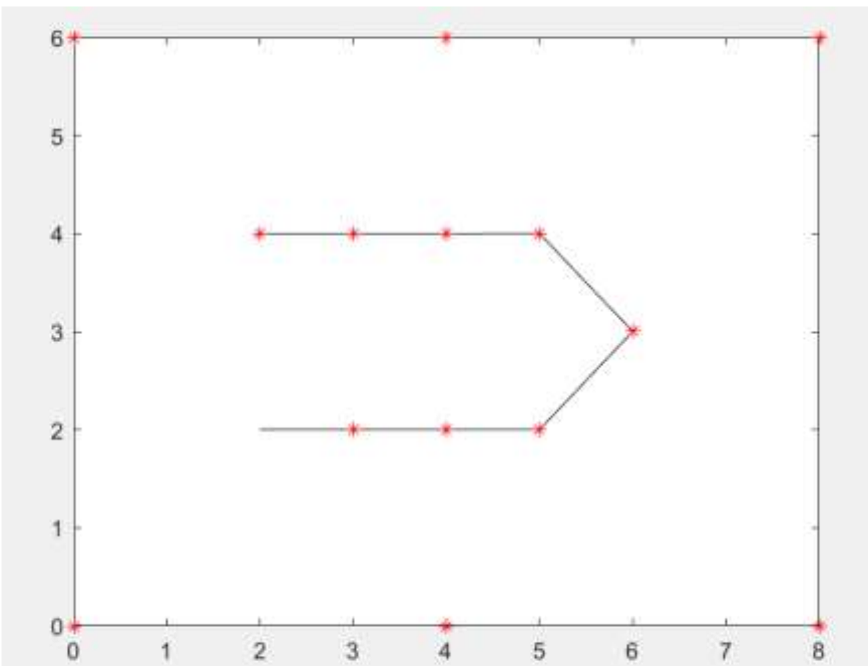
Project Assignment 1

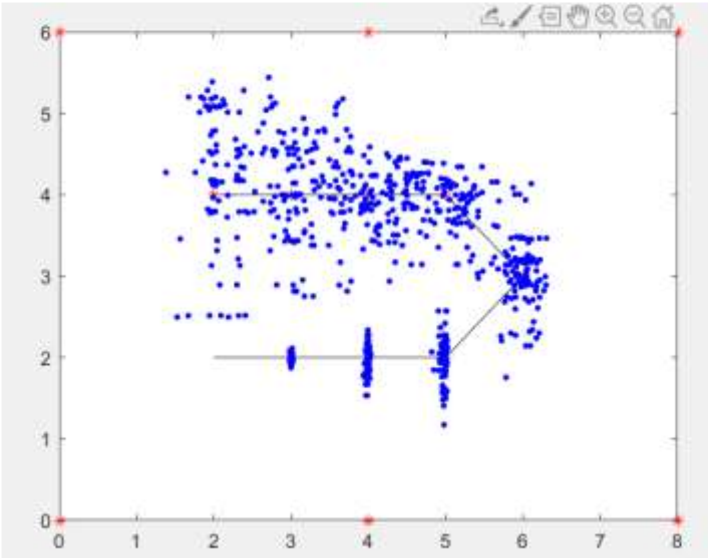
Particle filter:

We start with a robot at position (2, 2, 0). Since we are absolutely certain about this position all the particles are crowded at this point. In a noiseless motion the position of robot at increasing time steps along with landmarks is as shown.

1. The red marks at corners show landmarks
2. The black shows the nominal path of the robot
3. The red on the this path mark the position for a noiseless motion



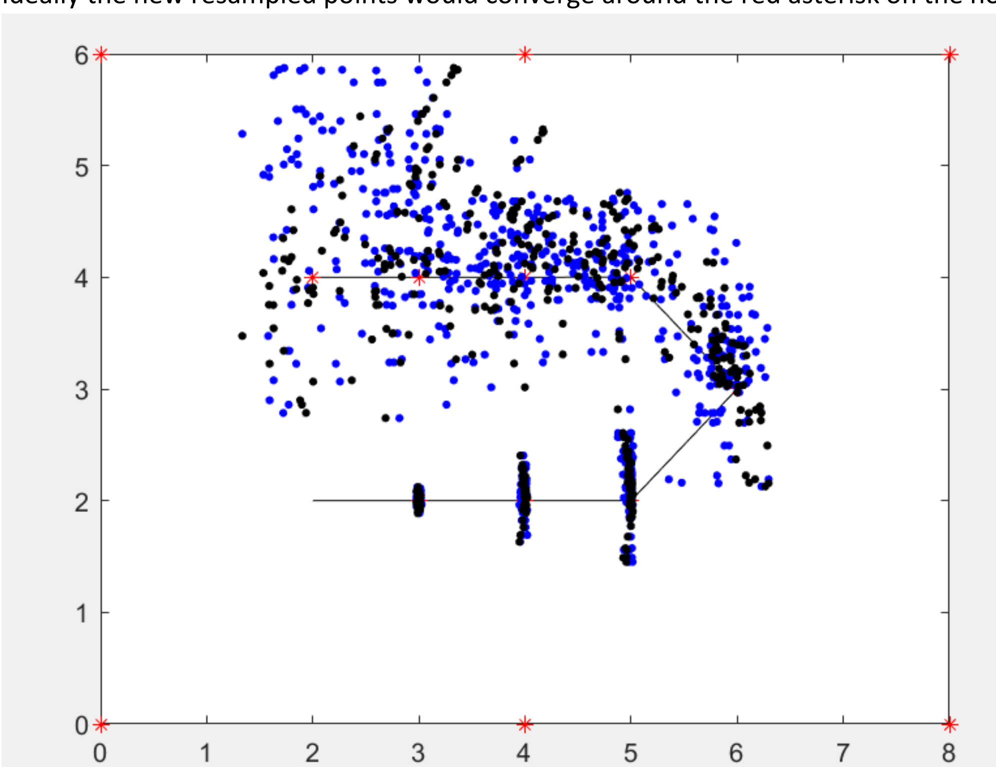
But, in real life, this is not what happens, the robot has errors in input and therefore continuously loses its sense of position, and deviates from the nominal. Fig 2 shows the particle samples at different time steps. There are M = 100 particles at every time step.



Next, we see how the robot can be localized better using the landmark info which assigns a weight to every particle. This weight is the probability of that position. Unfortunately, here I was unable to gather the weight quantity as I ended up getting 0 weights.

Therefore, I did the resampling by taking those particles that had their normalized x values greater than some randomly generated probability. This is general resampling.

Ideally the new resampled points would converge around the red asterisk on the nominal motion graph.

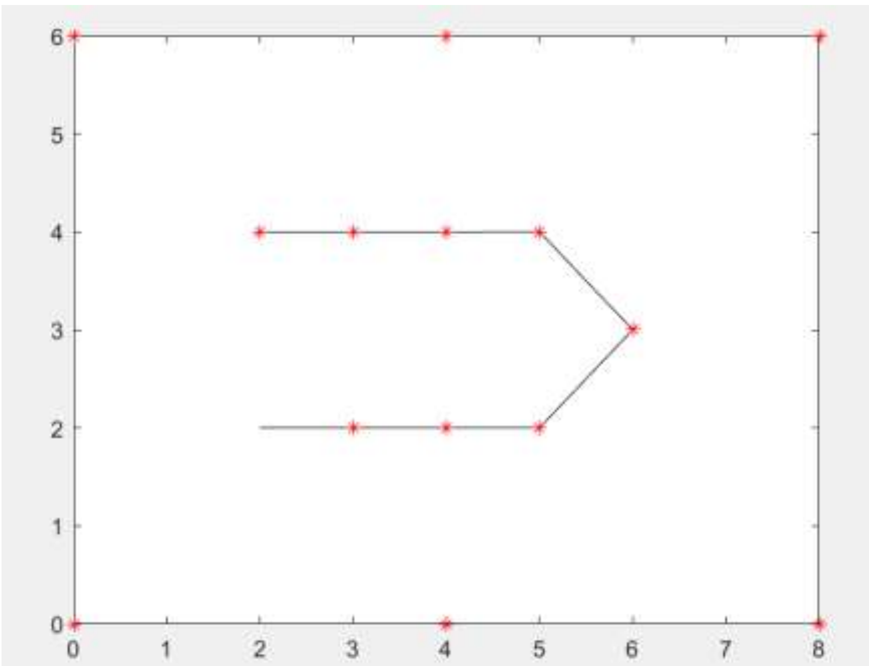


EKF Filter:

We start with a robot at position (2, 2, 0). This is the mean value, and because we are very certain, the robot has a covariance matrix of type [10^-9 0 0; 0 10^-9   0;   0 0 10^-9] .

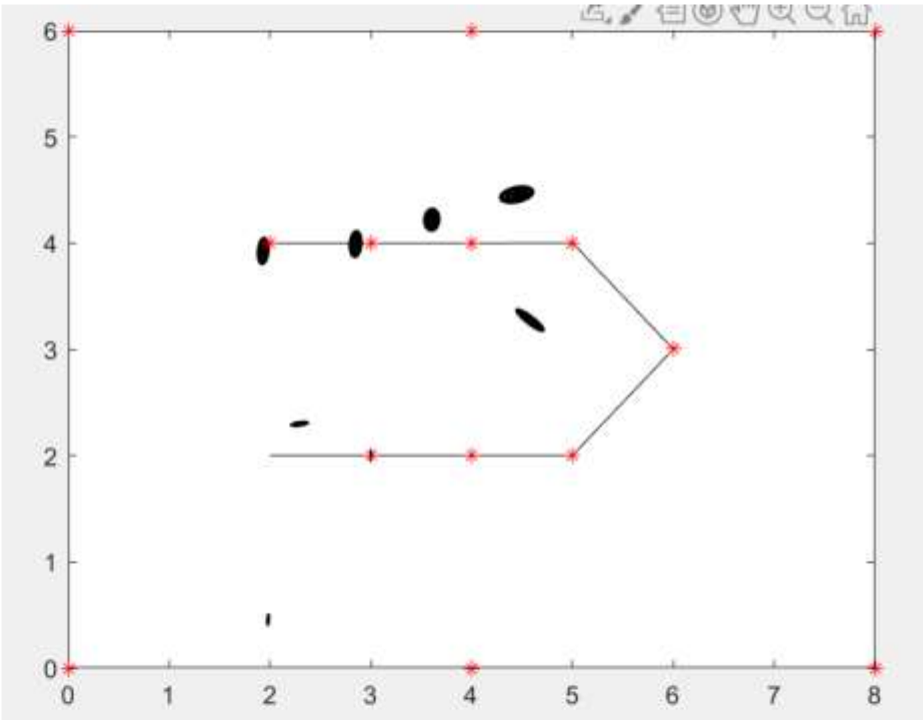
Nominal motion with no motion errors would look something like this:

1. The red marks at corners show landmarks
2. The black shows the nominal path of the robot
3. The red on the this path mark the position for a noiseless motion

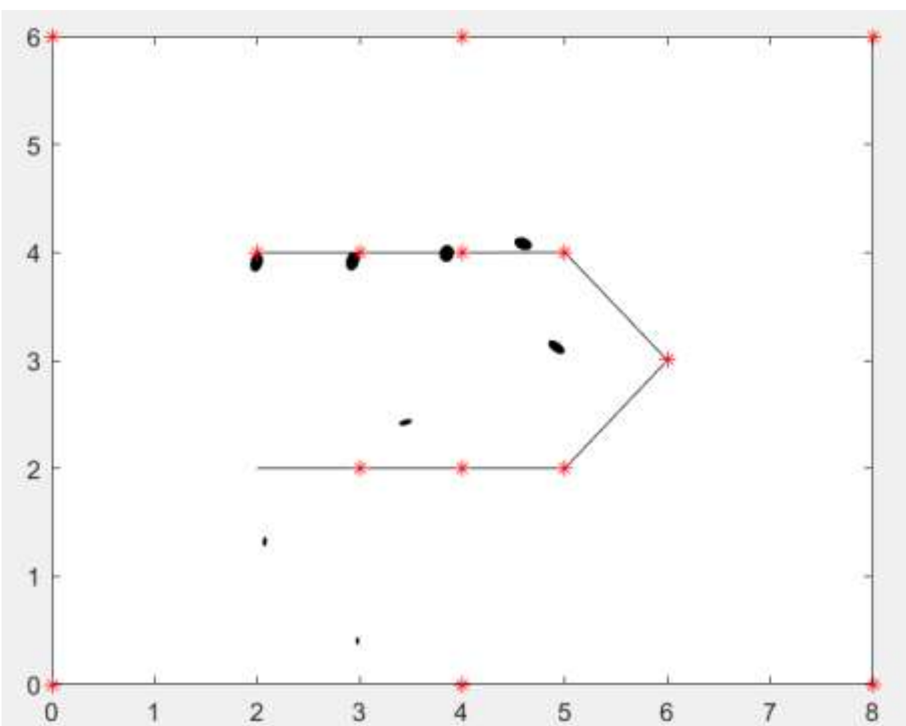


However, the true trajectory is not this. It is different. These ellipses give an idea about how the motion would have been if the inputs had noise and there were no landmarks available, that were observed.

These ellipses are made using the third part library : plot_gaussian_ellipsoid



When landmarks are factored in, the motion aligns to that of the nominal but not exactly. Notice how the covariances are reduced.



Particle filter code :

```
% EECS 499 - Homework 3
% Question 1.2
% Author: Krashagi Gupta
% Case ID: kxg360
clear;
clc;

% initialise
%% 1.2
global M
M = 100;
x0 = [2, 2, 0]; % vector inital pose
X0 = repmat(x0, M, 1);

Ut = [[1,0.0001]; [1,0.0001];[1,0.0001]; [pi/2,
pi/2]; [pi/2, pi/2];[1,0.0001];
[1,0.0001];[1,0.0001]];

Zt = [[2.276, 5.249, 2];[4.321, 5.834, 3];[3.418,
5.869, 3];[3.774, 5.911, 4];
    [2.631, 5.140, 5];[4.770,5.791,6];[3.828,
5.742, 6];[3.153, 5.739, 6]];

m1 = [0, 0];
m2 = [4, 0];
m3 = [8, 0];
m4 = [8, 6];
m5 = [4, 6];
m6 = [0, 6];
map = [m1; m2; m3; m4; m5; m6];
h = figure;
for i = 1:6
    plot(map(i,1),map(i,2),'r*')
    hold on ;
end
Xr = zeros(8 * M,4);
Xbar = zeros(8 * M,4);
```

```
step = 0;
tempX = zeros(M,3);
tempX = X0;

for q = 1 : 8
     [Xr(1+ M*(q - 1):q*M,:),Xbar(1+ M*(q -
1):q*M,:)]= mcl_algo(tempX, Ut(q,:), Zt(q,:), map);
%%%% problem is here for 2nd one
     tempX  = Xr(1+ M*(q - 1):q * M,:);
end

plot_everything(x0, Ut ,Xr, Xbar)

%% mcl algorithm
function [Xt, Xt_bar] = mcl_algo(Xt_1, ut, zt, map)

   global M
   alpha = [0.0001, 0.0001, 0.01, 0.0001, 0.0001,
0.0001 ];

    % line 2
    Xt_bar =  zeros(M,4);
    Xt = zeros(M,4);
    tempX = zeros(1,3);

    % line 3
    for m = 1: M


     tempX = sample_motion_model_velocity1(ut,
Xt_1(m,:))';

%
%    line 5
       tempWt = measurement_model(zt, tempX, map);
%%%% have to solve
%    line 6

        Xt_bar(m,:)= [tempX tempWt] ;
    end
```

```
     % rearrange by wt

         Xt_bar = sortrows(Xt_bar,1);
%          Xt_bar(m,:)
%        line 7
        Xt = resample(Xt_bar);


end
%% sample_motion_model

function [xn]= sample_motion_model(ut0, xp, alpha)


        xn =[0,0,0];
        xo = xp(1);
        yo = xp(2);
        thetao = xp(3);
        delT = 1 ;

    %      building an error array
        errorArray = sample_error(ut0,alpha);

        vcp  = ut0(1) + errorArray(1,1);
        wcp = ut0(2) + errorArray(2,1);
        ycp = errorArray(3,1);


        xn(1,1) = xo  + (-vcp/wcp)* sin(thetao) +
(vcp/wcp)*sin(thetao + (wcp* delT));
        xn(1,2) = yo  + (vcp/wcp)* cos(thetao) -
(vcp/wcp)*cos(thetao + (wcp* delT));
        xn(1,3) = thetao  +  wcp*delT  + ycp*delT ;

%     end
end

%% sample error
```

```
function[errorArray] = sample_error(ut, alp)

errorArray = [0; 0; 0;];
v = ut(1);
w = ut(2);
vcpE = sample(v, w, alp(1), alp(2));
wcpE = sample(v, w, alp(3), alp(4));
ycpE =  sample(v, w, alp(5), alp(6));
errorArray(1,1)= vcpE;
errorArray(2,1)= wcpE;
errorArray(3,1)= ycpE;

end
%% sample
function[err] = sample(v, w, alfF, alfS)
varnc = alfF*v + alfS*w;
mu = 0;
sigma = sqrt(varnc);
err = normrnd(mu,sigma);
end
%% sample2
function
[xt]=sample_motion_model_velocity1(ut,xtm1)
N = 1;
alpha=[0.0001;0.0001;0.01;0.0001;0.0001;0.0001];
vv=ut(1)+normrnd(0,(alpha(1)*ut(1)^2+alpha(2)*ut(2)
^2)^0.5,[N,1]);
ww=ut(2)+normrnd(0,(alpha(3)*ut(1)^2+alpha(4)*ut(2)
^2)^0.5,[N,1]);
rr=normrnd(0,(alpha(5)*ut(1)^2+alpha(6)*ut(2)^2)^0.
5,[N,1]);
    for i=1:N
    xt(1,:)=xtm1(1)-
vv./ww.*sin(xtm1(3))+vv./ww.*sin(xtm1(3)+ww.*1);
    xt(2,:)=xtm1(2)+vv./ww.*cos(xtm1(3))-
vv./ww.*cos(xtm1(3)+ww.*1);
    xt(3,:)=xtm1(3)+ww.*1+rr.*1;
    end
```

```
end

%%
function [reqProb] = measurement_model(z,X,m)
      % input received right
%     input form
%     z -> [d,a]
      d = z(1);
      a = z(2);
      s = z(3);
%     X -> [x, y, theta]
      x = X(1);
      y = X(2);
      theta = X(3);
%     m -> [mx, my]
      mx = m(s,1);
      my = m(s,2);
      % noise parameters
      sigmaR = 0.1;
      sigmaPhi = 0.09;

    dCp = sqrt((mx - x)^2 + (my - y)^2);
    alphaCp =  atan2(my - y,mx - x) - theta;
%     pDet = prob(dCp - d, sigmaR^2)* prob(alphaCp
- a, sigmaPhi^2);
    pDet = prob(dCp - d, sigmaR^2)* prob(alphaCp -
a, sigmaPhi^2);

    Zdet = 1;
    reqProb = Zdet * pDet;
%     1/(2*pi*sigmar^2)^0.5* exp(-0.5*(zt(1)-
rhat)^2/sigmar^2)


end
%% prob function
function [ansr] = prob(a, var)
%UNTITLED4 Summary of this function goes here
```

```
%   Detailed explanation goes here
ansr =  (1/(2*pi*var)^0.5)* exp(-0.5*(a)^2/var);
end

%% plot nominal motion
function [] = plot_everything(x0,Ut, Xr, Xbar)
%     x = 2;
%     y = 2;
%     theta = 0 ;
%     v = pi/2;
%     w = pi/2;
    X = zeros(8, 3);
    global M;
    global delta_t;
    delta_t = 1;
    old_location = x0;
    for t = 1: 8
        temp_X = move(old_location, Ut(t,:),
delta_t);
        line([old_location(1), temp_X(1)],
[old_location(2), temp_X(2)], 'Color','black');
        hold on ;

        X(t,:) = temp_X;
        old_location = temp_X;
    end

    for t = 1:8
        plot(X(t,1),X(t,2),'r*');
        hold on ;
    end
    for g = 1 : 8 * M
        scatter(Xbar(g,1),Xr(g,2),10,'b','filled');
        hold on;
    end
    for g = 1 : 8 * M
        scatter(Xr(g,1),Xr(g,2),10,'k','filled');
        hold on;
```

```
    end


end

%% functions
function [new_posi] = move(old_posi, miu, delta_t)
    new_posi = old_posi + [-miu(1)/miu(2) *
sin(old_posi(3)) + miu(1)/miu(2) * sin(old_posi(3)
+ miu(2) * delta_t), miu(1)/miu(2) *
cos(old_posi(3)) - miu(1)/miu(2) * cos(old_posi(3)
+ miu(2) * delta_t), miu(2) * delta_t];
end

%% resample
function [Xt] = resample(Xt_bar)
global M
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

    dim = size(Xt_bar);
    Xt = zeros(dim); % ensuring output has same
dimensions as input
    Xt_bar_norm =  Xt_bar./sum(Xt_bar);
    cumSum =  cumsum(Xt_bar_norm);
    for j = 1: M
         chance = rand(1, 1);
         % works right
         for k = 1 : M
            if(cumSum(k,1) >= chance)
                 Xt(j,:) = Xt_bar(k,:);
                 break;
            end
         end
    end
end

 %% shhd
```

```
function
[q]=landmark_model_known_correspondence(zt,ct,xt,m)
j=ct;
rhat=((m(1,j)-xt(1))^2+(m(2,j)-xt(2))^2)^0.5;
phihat=atan2(m(2,j)-xt(2),m(1,j)-xt(1))-xt(3);
q=1/(2*pi*0.01)^0.5*exp(-0.5*(zt(1)-
rhat)^2/0.01)*1/(2*pi*0.09^2)^0.5*exp(-0.5*(zt(2)-
phihat)^2/0.09^2);

end
```

Ekf code :

```
% EECS 499 - Homework 3
% Question 1.2
% Author: Krashagi Gupta
% Case ID: kxg360
clear;
clc;

% initialise
%% 1.1
global alpha
global sigmaR
global sigmaPhi
sigmaR = 0.1;
sigmaPhi = 0.09;
alpha = [0.0001, 0.0001, 0.01, 0.0001, 0.0001,
0.0001 ];
Ut = [[1,0.0001]; [1,0.0001];[1,0.0001]; [pi/2,
pi/2]; [pi/2, pi/2];[1,0.0001];
[1,0.0001];[1,0.0001]];
Zt = [[2.276, 5.249, 2];[4.321, 5.834, 3];[3.418,
5.869, 3];[3.774, 5.911, 4];
    [2.631, 5.140, 5];[4.770,5.791,6];[3.828,
5.742, 6];[3.153, 5.739, 6]];
m1 = [0, 0];
m2 = [4, 0];
m3 = [8, 0];
m4 = [8, 6];
m5 = [4, 6];
m6 = [0, 6];
map = [m1; m2; m3; m4; m5; m6];
for i = 1:6
    plot(map(i,1),map(i,2),'r*')
    hold on ;
end

mu0 = [2; 2; 0];
zeta0 =  [10^-9 0 0; 0 10^-9 0;0 0 10^-9 ];
```

```
c0 = Zt(1,3);

mu_t = [0 ; 0 ; 0];
zeta_t = zeros(3,3);
temp_mu = mu0 ;
temp_zeta  =zeta0 ;
mu_collec = zeros(8, 3);
mu_bar_collec = zeros(8, 3);
zeta_collec = zeros(24, 3);
zeta_bar_collec = zeros(24, 3);

for k = 1: 8
    [temp_mu ,temp_zeta, mu_t_bar,zeta_t_bar ] =
ekf_algo(temp_mu, temp_zeta, Ut(k,:), Zt(k,:),
Zt(k,3), map);
    mu_collec(k,:) = temp_mu' ;
    zeta_collec(1 + 3*(k -1) : 3*k,:)  = temp_zeta
;
    mu_bar_collec(k,:) = mu_t_bar' ;
    zeta_bar_collec(1 + 3*(k -1) : 3*k,:)  =
zeta_t_bar ;

end
%  h1 = plot_gaussian_ellipsoid([1 1], [1 0.5; 0.5
1]);
%  h2 = plot_gaussian_ellipsoid([2 1.5], [1 -0.7; -
0.7 1]);
%  h3 = plot_gaussian_ellipsoid([0 0], [1 0; 0 1]);
%  set(h2,'color','r');
%  set(h3,'color','g');

plot_everything(mu0', Ut, mu_collec, zeta_collec,
mu_bar_collec, zeta_bar_collec )
% plot_gaussian_ellipsoid(m, C, sdwidth, npts, axh)
%% ekf function

function [mu_t ,zeta_t, mu_t_bar,zeta_t_bar] =
ekf_algo(mu_t_1, zeta_t_1, ut, zt, ct, map)
%    test dimension
```

```
%    zt
%    ct
   map;
    global alpha
    global sigmaR
    global sigmaPhi
     del_t = 1;
     vt = ut(1,1);
     wt = ut(1,2);
%     line 2
    theta = mu_t_1(3,1);

%    line 3
%    iniialisation
    gt13 = (-vt/wt)* cos(theta) + (vt/wt)*
cos(theta + (wt * del_t));
    gt23 =  (-vt/wt)* sin(theta) + (vt/wt)*
sin(theta + (wt * del_t));
    Gt = [1 0 gt13 ; 0 1 gt23 ; 0 0 1]; %% 3* 3
matrix

%     line 4
%     initialisation
    vt11  = (-sin(theta)+ sin(theta +
(wt*del_t)))/wt ;
    vt12 = (vt/wt^2)*(sin(theta)- sin(theta + (wt *
del_t)))  +   (vt * del_t/wt)*(cos(theta + (wt *
del_t)));
    vt21 = (cos(theta)- cos(theta + (wt*del_t)))/wt
;
    vt22 = (-vt/wt^2)*(cos(theta)- cos(theta + (wt
* del_t)))  +   (vt * del_t/wt)*(sin(theta + (wt *
del_t)));
    Vt = [vt11 vt12 ; vt21 vt22; 0 del_t]; %% 3*2
matrix

%     line 5
```

```
    Mt = [alpha(1)*vt^2 + alpha(2)*wt^2  0; 0
alpha(3)*vt^2 + alpha(4)*wt^2 ];% 2*2 matrix

%     line 6
    mu_t_bar = mu_t_1 + [gt23 ; -gt13 ; wt* del_t];
% 3*1 matrix

%     line 7
    zeta_t_bar = (Gt * zeta_t_1 * Gt') + (Vt * Mt *
Vt');


%    line 8
    Qt = [sigmaR^2 0; 0 sigmaPhi^2]; %% 2*2 matrix

%     line 9 - > at one time instant only 1
observation
%       line 10
    j = ct;
%      line 11

    q = (map(j,1)- mu_t_bar(1,1))^2 + (map(j,2) -
mu_t_bar(2,1))^2 ;% single value

%     line 12

    zt_cap = [sqrt(q);atan2(map(j,2) -
mu_t_bar(2,1),map(j,1) - mu_t_bar(1,1)) -
mu_t_bar(3,1)];% 2*1 %% looks wrong the theta value
%     zt

%     line 13
    Ht = [-(map(j,1)- mu_t_bar(1,1))/sqrt(q)    -
(map(j,2) - mu_t_bar(2,1))/sqrt(q)   0;
            (map(j,2) - mu_t_bar(2,1))/q    -
(map(j,1)- mu_t_bar(1,1))/q    -1 ];  % 2*3 matrix

%         line 14
    St = Ht * zeta_t_bar * Ht' + Qt ; % 2*2
```

```
%     line 15
    %%% how to find inverse of matrix in matab
%     Y = inv(X) matrix must be square
    Kt = zeta_t_bar *   Ht' * inv(St) ;% 3*2

%     line 16

%      zt   - zt_cap
    mu_t = mu_t_bar + Kt * (zt(:,1:2)' - zt_cap); %
address concern % 3*3

%     line 17
%     how to make identity matrix of a given size
in matlab
%     I = eye(n)
    zeta_t = (eye(3) - Kt*Ht)* zeta_t_bar;
end
%% plot everything
function [] = plot_everything(x0,Ut,
mu_collec,zeta_collec,mu_bar_collec,
zeta_bar_collec)
x0
Ut
mu_collec
zeta_collec
mu_bar_collec
zeta_bar_collec
%     x = 2;
%     y = 2;
%     theta = 0 ;
%     v = pi/2;
%     w = pi/2;
    X = zeros(8, 3)
    global M;
    global delta_t;
    delta_t = 1;
    old_location = x0;
    for t = 1: 8
```

```
        temp_X = move(old_location, Ut(t,:),
delta_t);
        line([old_location(1), temp_X(1)],
[old_location(2), temp_X(2)], 'Color','black');
        hold on ;

        X(t,:) = temp_X;
        old_location = temp_X;
    end

    for t = 1:8
         plot(X(t,1),X(t,2),'r*');
        hold on ;
    end

%     for k = 1 : 1
%          plot_gaussian_ellipsoid(
mu_bar_collec(k,:) , zeta_bar_collec(1 + 3*(k -1) :
3 * k,:))
%     end
    %  h1 = plot_gaussian_ellipsoid([1 1], [1 0.5;
0.5 1]);
%  h2 = plot_gaussian_ellipsoid([2 1.5], [1 -0.7; -
0.7 1]);
%  h3 = plot_gaussian_ellipsoid([0 0], [1 0; 0 1]);
%  set(h3,'color','g');

    for k = 1 : 8
        plot_gaussian_ellipsoid( mu_collec(k,:) ,
zeta_collec(1 + 3*(k -1) : 3 * k,:));
    end

end
%% functions
function [new_posi] = move(old_posi, miu, delta_t)
    new_posi = old_posi + [-miu(1)/miu(2) *
sin(old_posi(3)) + miu(1)/miu(2) * sin(old_posi(3)
+ miu(2) * delta_t), miu(1)/miu(2) *
```

```
cos(old_posi(3)) - miu(1)/miu(2) * cos(old_posi(3)
+ miu(2) * delta_t), miu(2) * delta_t];
end
```

Third party library code –

```
function h = plot_gaussian_ellipsoid(m, C, sdwidth,
npts, axh)
% PLOT_GAUSSIAN_ELLIPSOIDS plots 2-d and 3-d
Gaussian distributions
%
% H = PLOT_GAUSSIAN_ELLIPSOIDS(M, C) plots the
distribution specified by
%  mean M and covariance C. The distribution is
plotted as an ellipse (in
%  2-d) or an ellipsoid (in 3-d).  By default, the
distributions are
%  plotted in the current axes. H is the graphics
handle to the plotted
%  ellipse or ellipsoid.
%
% PLOT_GAUSSIAN_ELLIPSOIDS(M, C, SD) uses SD as the
standard deviation
%  along the major and minor axes (larger SD =>
larger ellipse). By
%  default, SD = 1. Note:
%  * For 2-d distributions, SD=1.0 and SD=2.0 cover
~ 39% and 86%
%     of the total probability mass, respectively.
%  * For 3-d distributions, SD=1.0 and SD=2.0 cover
~ 19% and 73%
%     of the total probability mass, respectively.
%
% PLOT_GAUSSIAN_ELLIPSOIDS(M, C, SD, NPTS) plots
the ellipse or
%  ellipsoid with a resolution of NPTS (ellipsoids
are generated
%  on an NPTS x NPTS mesh; see SPHERE for more
details). By
%  default, NPTS = 50 for ellipses, and 20 for
ellipsoids.
%
```

```
% PLOT_GAUSSIAN_ELLIPSOIDS(M, C, SD, NPTS, AX) adds
the plot to the
%  axes specified by the axis handle AX.
%
% Examples:
% -------------------------------------------
%  % Plot three 2-d Gaussians
%  figure;
%  h1 = plot_gaussian_ellipsoid([1 1], [1 0.5; 0.5
1]);
%  h2 = plot_gaussian_ellipsoid([2 1.5], [1 -0.7; -
0.7 1]);
%  h3 = plot_gaussian_ellipsoid([0 0], [1 0; 0 1]);
%  set(h2,'color','r');
%  set(h3,'color','g');
%
%  % "Contour map" of a 2-d Gaussian
%  figure;
%  for sd = [0.3:0.4:4],
%    h = plot_gaussian_ellipsoid([0 0], [1 0.8; 0.8
1], sd);
%  end
%
%  % Plot three 3-d Gaussians
%  figure;
%  h1 = plot_gaussian_ellipsoid([1 1  0], [1 0.5
0.2; 0.5 1 0.4; 0.2 0.4 1]);
%  h2 = plot_gaussian_ellipsoid([1.5 1 .5], [1 -0.7
0.6; -0.7 1 0; 0.6 0 1]);
%  h3 = plot_gaussian_ellipsoid([1 2 2], [0.5 0 0;
0 0.5 0; 0 0 0.5]);
%  set(h2,'facealpha',0.6);
%  view(129,36); set(gca,'proj','perspective');
grid on;
%  grid on; axis equal; axis tight;
% -------------------------------------------
%
```

```
%  Gautam Vallabha, Sep-23-2007,
Gautam.Vallabha@mathworks.com
%  Revision 1.0, Sep-23-2007
%    - File created
%  Revision 1.1, 26-Sep-2007
%    - NARGOUT==0 check added.
%    - Help added on NPTS for ellipsoids
if ~exist('sdwidth', 'var'), sdwidth = 1; end
if ~exist('npts', 'var'), npts = []; end
if ~exist('axh', 'var'), axh = gca; end
if numel(m) ~= length(m),
    error('M must be a vector');
end
if ~( all(numel(m) == size(C)) )
    error('Dimensionality of M and C must match');
end
if ~(isscalar(axh) && ishandle(axh) &&
strcmp(get(axh,'type'), 'axes'))
    error('Invalid axes handle');
end
set(axh, 'nextplot', 'add');
switch numel(m)
   case 2, h=show2d(m(:),C,sdwidth,npts,axh);
   case 3, h=show3d(m(:),C,sdwidth,npts,axh);
   otherwise
      error('Unsupported dimensionality');
end
if nargout==0,
    clear h;
end
%-----------------------------
function h = show2d(means, C, sdwidth, npts, axh)
if isempty(npts), npts=50; end
% plot the gaussian fits
tt=linspace(0,2*pi,npts)';
x = cos(tt); y=sin(tt);
ap = [x(:) y(:)]';
[v,d]=eig(C);
```

```
d = sdwidth * sqrt(d); % convert variance to
sdwidth*sd
bp = (v*d*ap) + repmat(means, 1, size(ap,2));
h = plot(bp(1,:), bp(2,:), '-', 'parent', axh);
%-----------------------------
function h = show3d(means, C, sdwidth, npts, axh)
if isempty(npts), npts=20; end
[x,y,z] = sphere(npts);
ap = [x(:) y(:) z(:)]';
[v,d]=eig(C);
if any(d(:) < 0)
   fprintf('warning: negative eigenvalues\n');
   d = max(d,0);
end
d = sdwidth * sqrt(d); % convert variance to
sdwidth*sd
bp = (v*d*ap) + repmat(means, 1, size(ap,2));
xp = reshape(bp(1,:), size(x));
yp = reshape(bp(2,:), size(y));
zp = reshape(bp(3,:), size(z));
h = surf(axh, xp,yp,zp);
```