```
% Matlab script for a simple EKF localization simulation

%Plot the ground-truth world-map locs of the landmarks as dark green disks
%Plot the ground-truth world-map locs of the viapoints as red disks
%Plot the robot ground-truth position as a dark cyan triangle
%Plot the estimated landmark locations as light green disks
%Plot the estimated viapoints as yellow disks
%Plot the estimated robot pose as a cyan filled triangle


% Kalman filters are highly robust to gaussian nose.
% In fact they are amazingly tenacious.
% This simulation exhibits that quality

% sensor noise covariance scale factor for landmark poses
snsf = 2;
% EKF correction-step noise covariance, Q, scale factor
qnsf = 1;
% odometry noise scale factor
onsf = 1.5;
% heading noise scale factor:
% relative sensitivity of robot heading w.r.t. fwd & rot vel odometry noise
hnsf = 0.001;

% Temporal Parameters

% Number of iterations before time-out
NIters = 512;
% waiting time between iterations (to make it as fast as possible set it to
0)
% PauseTime = 0.25;
PauseTime = 0;
% Sampling time interval
DeltaT = 1;

% Robot Parameters

% Initial pose of robot
StartingPose = [150;0;90*pi/180]; % [cm;cm;radians];
% Robot's forward velocity cm/s
v = 10;
% Robot's angular velocity, w, is computed below from
% the location of the next via point.
% Robot's angular velocity fudge factor -- a global adjustment
% to increase or decrease the computed angular velocity control
avff = 1.0;

% Noise Parameters

% SNoiseCov is a 3x3 covariance matrix for the noise to be added to
% the sensor estimate of the pose of a landmark w.r.t. the robot's
% own estimated (noisy) pose.
% I.e. w.r.t. each landmark's x position, y position, and bearing angle
SNoiseCov = snsf*[1 0 0 ; 0 1 0 ; 0 0 1];
% Measurement noise covariance used in EKF correction step
```

```
EstSNCov = qnsf*[1;1;1];
% Odometery noise parameters
a1 = onsf*0.05; % translational velocity (TV) to estimated TV
a2 = onsf*0.05; % Rotational velocity (RV) to estimated TV
a3 = onsf*0.05; % TV to est RV
a4 = onsf*0.05; % RV to est RV
% a1=0;a2=0;a3=0;a4=0; % noise-free case

% INITIALIZE LANDMARKS AND VIAPOINTS

% Landmarks is a 2x12 matrix of 12 (x,y) coordinate vectors (World Frame)
% ViaPoints is a 4x12 matrix such that for i=1:12 (order of crossing)
%     ViaPoints(1:2,i) are the map coordinates of the ith viapoint,
%     ViaPoints(3,i) is the id (index) number of the landmark to the left,
%     ViaPoints(4,i) is the id (index) number of the landmark to the right.
% fig1 is the matlab graphics handle of the figure window
% to be used for the plots.
% The function plots the ground truth landmarks (green) and viapoints
% (orange) and their id numbers.
% The world map extends from (x,y)=(cols,rows)=(-50,-50) to (450,650)
[Landmarks,ViaPoints,fig1] = SetUpLandmarks;
% [current viapoint #; left landmark #; right landmark #]
vp = [1 ; ViaPoints(3) ; ViaPoints(4)];

% INITIALIZE ROBOT

% Robot pose vector before EKF estimation [x;y;heading(radians)] w.r.t.
% world map coordinates
x0 = StartingPose;   % starting pose of robot
% Dislay it as an open cyan colored triangle pointing in dir of heading
PlotRobot(fig1,x0,[0 0.5 0.5],[])
%
% Initialize the statistics
% Initial system state transition covariance
sigma0 = eye(3);
% "Actual" sensor noise covariance used in the estimation
%  of the bearing angles tolandmarks
Cov = SNoiseCov;
% Kalman filter's sensor noise covariance, Q, for correction step
sigmaZ = EstSNCov;
% Odometery noise parameters
alphas = [a1 a2 a3 a4]';
%
% Initialize the robot controls
d = norm(ViaPoints(1:2,1)-x0(1:2)); % distance from here to viapoint 1
% Compute angular velocity control, w, in radians/s as the bearing to
% the current estimated viapoint / (distance / forward velocity)
% adjusted by avff (usually == 1) to globally increase or decrease the
% angular velocity so computed.
w = avff*(atan2((ViaPoints(2,1)-x0(2)),(ViaPoints(1,1)-x0(1))) -
x0(3))/(d/v);
u = [v w]';     % control vector
dt = DeltaT;    % delta time
% Initialize the working variables
VPEst = ViaPoints; % this will hold the EKF estimate of the via point
locations
```

```
mu0 = x0;       % start with estimate = ground truth
x1  = x0;       % set up for immediate swap

% At each time step record:
%     the actual and estimated robot poses,
%     the minimal distance between true via points
%     the system covariances, the control velocities,
%     the distances and bearings to the landmarks,
%     and the via point being approached
Actual = zeros(3,NIters);
Estimate = zeros(3,NIters);
Minimal = zeros(2,NIters);
Sigmas = zeros(3,3,NIters);
Vels = zeros(2,NIters);
LMdb = zeros(3,size(Landmarks,2),NIters);
VPs   = zeros(4,NIters);

% number of steps taken from previous via point to
% arrive at current via point. (reset at each via point)
nStepsVP = 0;
% number of steps taken to pass through all via points so far
% The value is updated when a new via point is reached and held until
% the next one is reached.
lStepsVP = 0;

rdcf = 180/pi;  % rad -> deg conversion factor
fig2 = figure; hold on;xlabel('iteration');ylabel('heading (degrees)');
fig2.MenuBar = 'none';
fig2.OuterPosition = [628 540 500 540];
ax = gca;
ax.Title.String = 'Heading Angles';
WinOnTop(fig2,true);


fig3 = figure; hold on;xlabel('iteration');ylabel('distance (cm)');
fig3.MenuBar = 'none';
fig3.OuterPosition = [1192 540 500 540];
ax = gca;
ax.Title.String = 'Path Difference';
WinOnTop(fig3,true);

% uncomment pause to wait before beginning iterations
% pause;

% MAIN LOOP

for k=1:NIters

    % Record the pose states and the system covariance
    Actual(:,k) = x0; %
    Estimate(:,k) = mu0;
    Sigmas(:,:,k) = sigma0;
    % figure(h2), stairs ([Actual(3,1:k)*rdcf;Estimate(3,1:k)*rdcf]')

    % COMPUTE GROUND TRUTH
```

```
    % distance to current viapoint as estimated by the EKF
    d = norm(VPEst(1:2,vp(1))-mu0(1:2));

    % To set robot's heading, test for crossing of current viapoint
    LeftLM = Landmarks(:,vp(2));     % left landmark location
    RightLM = Landmarks(:,vp(3));    % right landmark location
    Side = SideOfLine(mu0,LeftLM,RightLM);
    % If the robot is to the left of the viapoint line, or it is within 15 cm
if
    % the via point, it has arrived there
    % If that was the final final via point, stop.
    % otherwise, get the next via point.
    if (d < 15) || (Side(1) == 'L')
        % when a new via point is reached, sample the straight-line
        % distance between it and the previous via point.
        % one sample for each time step (each iteration of the EKF)
        if vp(1) == 1
            % distance at first via point is w.r.t. the robot's staring
position
            deltaVP = (ViaPoints(1:2,1)-StartingPose(1:2,1))/nStepsVP;
            Minimal(:,lStepsVP+1:k) = ...
                repmat(StartingPose(1:2,1),[1 nStepsVP+1]) + ...
                (0:nStepsVP).*repmat(deltaVP,[1 nStepsVP+1]);
        else
            % at all other via points the distance is computed between it
            % and the previous one
            deltaVP = (ViaPoints(1:2,vp(1))-ViaPoints(1:2,vp(1)-
1))/(nStepsVP);
            Minimal(:,lStepsVP+1:k) = ...
                repmat(ViaPoints(1:2,vp(1)-1),[1 nStepsVP]) + ...
                ((0:nStepsVP-1).*repmat(deltaVP,[1 nStepsVP]));
        end
        % number of steps so far.
        % Hold this value until the next via point is reached
        lStepsVP = k;
        % reset number of steps taken to pass through the next via point
        nStepsVP = 1;
        vp(1) = vp(1)+1; % via point number
        % Get the L & R landmark labels and positions
        vp(2:3) = [VPEst(3,vp(1));VPEst(4,vp(1))]; % L,R landmark
        % distance from here to next via point
        d = norm(VPEst(1:2,vp(1))-mu0(1:2));
        % if we are at the last via point, break out of the loop,
        % collect and display statistics, and plot results
        if vp(1) == size(ViaPoints,2)
            % draw the robot here before breaking out
            PlotRobot(fig1,x0,[0 0 0],'fill');
            break;
        end
    else
        nStepsVP = nStepsVP + 1;
    end

    % VELOCITY CONTROL
```

```
    % bearing to next viapoint as estimated by the EKF
    th = atan2((VPEst(2,vp(1))-mu0(2)),(VPEst(1,vp(1))-mu0(1))) - mu0(3);
    % Correct the bearing for branch cut
    th = mod(th+pi,2*pi)-pi;
    % check for runaway condition (arbitrarily set to 150 cm away from next
via point)
    if (d<150) % no runaway
        % compute angular velocity control, w,  as the bearing to the next
        % viapoint / (distance / forward velocity)
        w = th/(d/u(1));
    else % possible runaway
        disp(['Warning: Possible runaway condition! Viapoint target: '
num2str(vp(1))])
        % set the rotational velocity directly to the viapoint bearing
        w = th;
    end
    % scale the rotational velocity control by the "angular velocity fudge
factor"
    % which is usually 1
    u(2) = avff*w;

    % Record the control velocities
    Vels(:,k) = u;

    % Return x1 = [x1;y1;theta1], the 2D pose of point, x0 = [x0;y0;theta0],
    % after the application of [translational;Rotational] velocity vector,
    % u = [v;w] for dt seconds.
    % I.e. move the robot exactly (from where it actually is to where
    % it will be by following the controls estimated via the EKF)
    % using the system's motion model.
    x1 = TRVdtToXYTh(x0,u,dt);

    % SENSE NEW POSITION

    % Add odometry noise to the ground truth location
    xs = (a1*(v.^2)+a2*(w.^2));
    ys = (a3*(v.^2)+a4*(w.^2));
    ts = sqrt(xs+ys)*hnsf;  % hnsf heading noise scale factor set at
beginning
    M1 = [xs 0 0; % motion noise covariance
          0 ys 0;
          0 0 ts];
    % Add multivariate normally distributed random noise with
    % mean 0, cov M1 to the robot pose
    if sum(M1(:)) ~= 0
        MN = mvnrnd([0;0;0],M1)'; % requires a non zero covariance matrix
        x1 = x1 + MN;
    end
    % Correct the heading for > 2*pi branch cut
    x1(3) = mod(x1(3)+pi,2*pi)-pi;
    % Now the robot's pose estimate is incorrect.

    % The landmark poses are are computed from where the robot is according
    % to its noisy motion model, not where the EKF thinks it is.
    % Those poses are then used by the EKF to update its estimate
    % of where the robot is.
```

```
    %
    % Get estimated landmark poses w.r.t. moved robot's noisy pose.
    % Motion noise with covariance M1 was added to the robot's pose above.
    % The LandMarkPose function adds noise with covariance Cov to the
    % landmark pose estimate.
    [dist,bearing,index]=LandmarkPose(x1,Landmarks,Cov);
    % Now the robot's estimates of the poses of the landmarks are incorrect

    % Sensor matrix z's column i is [dist;bearing;index]
    % of landmark(index(i)) sorted by distance from robot
    z = [dist;bearing;index];
    % Record the distances and bearings to all 12 landmarks
    LMdb(:,:,k) = z;

    % Estimate the robot's pose using the EKF based on its estimated
    % previous pose and using the landmark info provided by
    % the robot after the motion control was applied.
    [ mu1, sigma1 ] = EKFLoc( mu0, sigma0, alphas, u, dt, z, sigmaZ, index,
[Landmarks;1:12]);

    % Convert landmark distances and orientations w.r.t EKF estimated Robot
    % pose to x, y map coordinates and orientations w.r.t world x-axis ...
    [x,y,~] = DBwrtRtoMapXY(dist,bearing,mu1,index);
    %   ... so we can compute viapoint locations w.r.t. the estimated
    %   landmark locations.
    % Compute viapoints in traversal order.
    VPEst = ComputeVPs(x,y);
    % Record the viapoints
    VPs(:,k) = VPEst(:,vp(1));

    % PLOTTING

    figure(fig1);clf; % clear figure
    % Plot the ground-truth world-map locs of the landmarks as dark green
disks
    PlotCircles(fig1,Landmarks(1,:),Landmarks(2,:),400,[0.0 0.5 0.0],'fill');
    % Plot the ground-truth world-map locs of the viapoints as red disks
    PlotCircles(fig1,ViaPoints(1,:),ViaPoints(2,:),120,[1.0 0.0 0.0],'fill');
    % Plot the robot ground-truth position as a dark cyan triangle
    PlotRobot(fig1,x1,[0 0.4 0.4],'fill');

    % Plot the estimated landmark locations as light green disks
    PlotCircles(fig1,x,y,400,[0.0 0.8 0.0],'fill');
    % Plot the estimated viapoints as yellow disks
    PlotCircles(fig1,VPEst(1,:),VPEst(2,:),120,[1 0.9 0],'fill');
    % Plot the estimated robot pose as a cyan filled triangle
    PlotRobot(fig1,mu1,'c','fill');

    % Plot actual and estimated heading angles in degrees
    figure(fig2)
    hold on
    stairs(Actual(3,1:k)*rdcf,'b');
    stairs(Estimate(3,1:k)*rdcf,'g');
    hold off
```

```matlab
    % plot position difference
    figure(fig3)
    hold on
    stairs(ColNorm(Estimate(1:2,1:k)-Actual(1:2,1:k)));
    hold off

    drawnow

    disp(['iteration: ' int2str(k)])
    disp(['act: x = ' num2str(x1(1)) ' y = ' num2str(x1(2)) ' hdg = '
num2str(x1(3)*rdcf)])
    disp(['est: x = ' num2str(mu1(1)) ' y = ' num2str(mu1(2)) ' hdg = '
num2str(mu1(3)*rdcf)])
    adif = mod(mu1(3)-x1(3)+pi,2*pi)-pi;
    disp(['dif: x = ' num2str(mu1(1)-x1(1)) ' y = ' num2str(mu1(2)-x1(2)) '
hdg = ' num2str(adif*rdcf)])
    disp(['vp: ' num2str(vp(1)) ' dist = ' num2str(d) ' brng = '
num2str(th*rdcf) ])
    disp(['ctrl: fwd = ' num2str(u(1)) ' rot = ' num2str(u(2)*rdcf)])
    disp('  ')

    % make new values current
    x0 = x1;
    mu0 = mu1;
    sigma0 = sigma1;

    pause(PauseTime);
%     pause;
end

if k == NIters
    disp(['Timed out after ' num2str(k) ' iterations.']);
end

Actual = Actual(:,1:k);
Estimate = Estimate(:,1:k);
Minimal = Minimal(:,1:k);
Sigmas = Sigmas(:,1:k);
Vels = Vels(:,1:k);
LMdb = LMdb(:,1:k);
VPs  = VPs(:,1:k);

TimeStamp = datestr(clock,'_yyyy-mm-dd_HH.MM.SS');

PrfPathLength = sum(ColNorm(diff([StartingPose(1:2)
ViaPoints(1:2,1:13)],[],2)));
display(['Minimum path length through all viapoints: '
num2str(PrfPathLength)]);
ActPathLength = sum(ColNorm(diff(Actual(1:2,:),[],2)));
display(['Length of path actually traversed by robot: '
num2str(ActPathLength)]);
EstPathLength = sum(ColNorm(diff(Estimate(1:2,:),[],2)));
display(['Length of EKF estimated path: ' num2str(EstPathLength)]);
perg = sum((ColNorm(Actual(1:2,:)-Minimal(1:2,:))).^2);
```

```
display(['Sum of squared difference between actual and minimal paths: ',
num2str(perg)]);
aerg = sum((mod(Estimate(3,:)-Actual(3,:)+pi,2*pi)-pi).^2);
display(['Sum of squared difference in headings: ', num2str(aerg)]);

figure(fig2)
legend('Ground Truth','UKF Estimate','Location', 'SouthWest');
% savefig(fig2,['Heading_Angles_' TimeStamp '.fig']);

% savefig(fig3,['Position_Difference_' TimeStamp '.fig']);

fig4=figure;
hold on
PlotCircles(fig4,Landmarks(1,:),Landmarks(2,:),400,[0.0 0.8 0.0],'fill');
PlotLandmarks(fig4,ViaPoints(1,:),ViaPoints(2,:),120,[1 0.9 0],'fill')
line(Minimal(1,:),Minimal(2,:),'color','k');
line(Estimate(1,:),Estimate(2,:),'color','r');
title('Robot Trajectories: minimal (black), estimated (red)')
fig4.MenuBar = 'none';
fig4.OuterPosition = [64 32 500 540];
WinOnTop(fig4,true);
hold off
% savefig(fig4,['EKF_Trajectories' TimeStamp '.fig']);

fig5=figure;
hold on
PlotCircles(fig5,Landmarks(1,:),Landmarks(2,:),400,[0.0 0.8 0.0],'fill');
PlotCircles(fig5,ViaPoints(1,:),ViaPoints(2,:),120,[1 0.9 0],'fill')
npts = size(Estimate,2);
PlotArrows(fig5,Estimate(1:2,:),10*ones(1,npts),Estimate(3,:),'b');
title('EKF Heading Estimates')
fig5.MenuBar = 'none';
fig5.OuterPosition = [628 32 500 540];
WinOnTop(fig5,true);
hold off
% savefig(fig5,['EKF_Headings' TimeStamp '.fig']);

fig6=figure;
hold on
PlotCircles(fig6,Landmarks(1,:),Landmarks(2,:),400,[0.0 0.8 0.0],'fill');
PlotCircles(fig6,ViaPoints(1,:),ViaPoints(2,:),120,[1 0.9 0],'fill')
lgths = 10*ColNorm(Estimate(1:2,:)-Actual(1:2,:));
adif = mod(Estimate(3,:)-Actual(3,:)+pi,2*pi)-pi;
PlotArrows(fig6,Actual(1:2,:),lgths,adif,'b');
title('Difference between actual and EKF Heading Estimates')
fig6.MenuBar = 'none';
fig6.OuterPosition = [1192 32 500 540];
WinOnTop(fig6,true);
hold off
% savefig(fig6,['Heading_Differential' TimeStamp '.fig']);

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% references: Prof. Richard Alan Peters (Intelligent systems and robotics
class)
```