OPTIMAL ESTIMATION IN DYNAMIC SYSTEMS

EXCERCISE 8: SLAM

Nimish Shrenik Shah — *s2088894*
n.s.shah@student.utwente.nl

MSc – EE, RaM
Faculty of Electrical Engineering, Mathematics and Computer Science
Course Code: 2018-191210920-2B

Supervised by:
dr.ir. F. van der Heijden

UNIVERSITY OF TWENTE.

MAY 12, 2020

Problem Description

The purpose of this assignment is to use SLAM to improve the estimate of the track using the observations of the landmarks, and at the same time to estimate the (static) positions of these landmarks.

Asumptions **Problem Description**

1. A vehicle is moving around in an unexplored area.
2. The motion of the vehicle is monitored in terms of heading and speed. The system equation is:

$$\begin{bmatrix} x(i+1) \\ y(i+1) \end{bmatrix} = \begin{bmatrix} x(i) \\ y(i) \end{bmatrix} + \Delta \cdot (v(i) + \tilde{v}(i)) \begin{bmatrix} \cos(\phi(i) + \tilde{\phi}(i)) \\ \sin(\phi(i) + \tilde{\phi}(i)) \end{bmatrix}$$

   Where, $\Delta = 0.25\,\mathrm{s}$ is the sampling period . The random variables $\tilde{v}(i)$ and $\tilde{\phi}(i)$ represent the uncertainty due to the measurement noise with $\sigma_v = 5\,\mathrm{m\,s^{-1}}$ and $\sigma_\phi = 15°$. Both are uncorrelated with each other in time,
3. The track of the vehicle can be estimated by means of the heading and the speed, but errors will accumulate. Therefore, this estimate will become inaccurate in due course of time.
4. The space is provided with a number of landmarks. These are identifiable and static points. Beforehand, the number of landmarks and their positions are fully unknown.
5. The vehicle is equipped with a sensor system that is able to locate the positions of the landmarks relative to its own position.
6. Only landmarks that are nearby the vehicle can be measured. That is, the landmarks should be within the visible range of the sensor system.

Part I: Prediction Only

Q1.Determine the system matrix F.

As mentioned in the introduction of this assignment that landmarks are *static*, that is, they do not move with time, $\mathbf{p}(i+1) = \mathbf{p}(i)$ if the noise is disregarded. Hence the system matrix,

$$\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Q2. Give the covariance matrix $\mathbf{C}_{\tilde{\mathbf{w}}}$ of $\begin{bmatrix} \tilde{v}(i) & \tilde{\phi}(i) \end{bmatrix}^T$, Jacobian matrix $\mathbf{G}(\mathbf{u}(i))$ and the covariance matrix $\mathbf{C}_{\mathbf{w}}$.

$$\mathbf{C}_{\tilde{\mathbf{w}}} = \mathrm{diag}\left(\begin{bmatrix} \sigma_v^2 & \sigma_\phi^2 \end{bmatrix}\right) = \begin{bmatrix} 25 & 0 \\ 0 & 0.0685 \end{bmatrix}$$

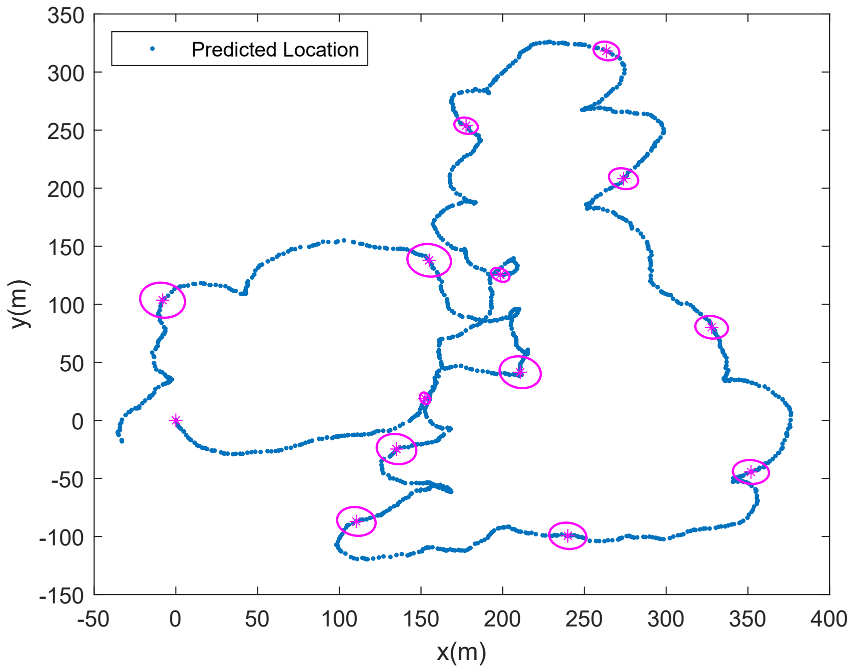


Figure 1: Prediction only estimation

From the system equation,

$$\frac{\partial x(i)}{\partial v(i)} = \Delta \cdot \cos(\phi(i)); \quad \frac{\partial x(i)}{\partial \phi(i)} = -\Delta \cdot v(i) \cdot \sin(\phi(i)); \quad \frac{\partial y(i)}{\partial v(i)} = \Delta \cdot \sin(\phi(i)); \quad \frac{\partial y(i)}{\partial v(i)} = \Delta \cdot v(i) \cdot \cos(\phi(i))$$

Hence

$$\mathbf{G}(\mathbf{u}(i)) = \Delta \cdot \begin{bmatrix} \cos(\phi(i)) & -v(i)\sin(\phi(i)) \\ \sin(\phi(i)) & v(i)\cos(\phi(i)) \end{bmatrix}$$

Covariance matrix $\mathbf{C_w}$ can in terms of $\mathbf{C_{\tilde{w}}}$ and $\mathbf{G}(\mathbf{u}(i))$ can be written as,

$$\mathbf{C_w} = \mathbf{G}(\mathbf{u}(i))\,\mathbf{C_{\tilde{w}}}\,\mathbf{G}(\mathbf{u}(i))^T$$

Q3. Develop a Matlab script that predicts the trajectory. Insert uncertainty regions in your graph at $i = 100, 200, \cdots$.

Refer part 1 of the script `ex8.m` provided in the Appendix.

Part II: Bearing and Distance Measurements

Q4. Suppose that at a particular time i only two landmarks are visible. Suppose that the ids of the two landmarks are 200 and 21 (just an example). Suppose that both landmarks have been seen previously, and that their estimated position is embedded in the state vector at element 3 and 4 (landmark with id=21), and 7 and 8 (landmark with id 200). Define the measurement vector $\mathbf{z}(i)$, measurement matrix $\mathbf{H}(i)$, covariance matrix of the measurement noise $\mathbf{C}_n$ for this particular case. How should the matrix $\mathbf{G}(\mathbf{u}(i))$ be defined so that the process noise is properly modelled?

Suppose, total K landmarks are already in the pool. The state vector will look like:

$$\begin{bmatrix}\mathbf{p}(i) & \mathbf{m}_{200} & \cdot & \mathbf{m}_{21} & \cdots & \text{upto } K^{\text{th}} \text{ landmark}\end{bmatrix}^T$$

The measurement vector contains entries that are only related to landmarks observed at the particular instant. Hence the measurement vector $\mathbf{z}(i) = \begin{bmatrix}\mathbf{x}_{200} & \mathbf{x}_{21}\end{bmatrix}^T \in \mathbb{R}^{4\times 1}$.

The corresponding measurement matrix,

$$\mathbf{H}(i) = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \end{bmatrix} \in \mathbb{R}^{2\times 2+2K}$$

As only two landmarks are observable at i^{th} instant, covariance matrix $\mathbf{C}_n = \begin{bmatrix} 25 & 0 & 0 & 0 \\ 0 & 25 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix}$

The Jacobian matrix $\mathbf{G}(\mathbf{u}(i)) = \begin{bmatrix} \frac{\partial x(i)}{\partial v(i)} & \frac{\partial x(i)}{\partial \phi(i)} \\ \frac{\partial x(i)}{\partial v(i)} & \frac{\partial x(i)}{\partial \phi(i)} \end{bmatrix}$ calculated over every iteration which is only affected by robot pose, in this case particularly, the position.

Q5. Suppose that at another instance of time, j, only one landmark is visible. The landmark has never been seen before. So, it must be a new one. The id assigned to that landmark is id=156. To be able to update the system with this measurements, the state vector must be augmented, and a prediction including this new landmark must be provided; the corresponding prediction covariance matrix must adapted, and the system function needs to be redefined. Describe how (mathematical equation) how these actions must be accomplished.

Suppose, total K landmarks are already seen and landmarks are denoted by $\mathbf{m}$, which means, this newly seen landmark can be denoted by $\mathbf{m}_{K+1}$. Hence, the new state vector will be, $\begin{bmatrix}\mathbf{p}(j) & \mathbf{m_1} & \cdots & \mathbf{m_K} & \mathbf{m_{K+1}}\end{bmatrix}^T$

The new prediction covariance matrix will be,

$$\begin{array}{c} \begin{bmatrix} \mathbf{C_p} & \mathbf{C_{pm_1}} & \mathbf{C_{pm_2}} & \cdots & \mathbf{C_{pm_K}} & \mathbf{C_{pm_{K+1}}} \\ \mathbf{C_{m_1p}} & \mathbf{C_{m_1}} & \mathbf{C_{m_1m_2}} & \cdots & \mathbf{C_{m_1m_K}} & \mathbf{C_{m_1m_{K+1}}} \\ \mathbf{C_{m_2p}} & \mathbf{C_{m_2m_1}} & \mathbf{C_{m_2}} & \cdots & \mathbf{C_{m_2m_K}} & \mathbf{C_{m_2m_{K+1}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{C_{m_Kp}} & \mathbf{C_{m_Km_1}} & \mathbf{C_{m_Km_2}} & \cdots & \mathbf{C_{m_Km_K}} & \mathbf{C_{m_Km_{K+1}}} \\ \mathbf{C_{m_{K+1}p}} & \mathbf{C_{m_{K+1}m_1}} & \mathbf{C_{m_{K+1}m_2}} & \cdots & \mathbf{C_{m_{K+1}m_K}} & \mathbf{C_{m_{K+1}}} \end{bmatrix} \end{array} \begin{array}{l} \mathbb{R}^{2\times 2(K+1)} \\ \mathbb{R}^{2\times 2(K+1)} \\ \mathbb{R}^{2\times 2(K+1)} \\ \vdots \\ \mathbb{R}^{2\times 2(K+1)} \\ \mathbb{R}^{2\times 2(K+1)} \end{array}$$

The state vector will be augmented by adding new $\mathbf{m}$ at the end of the list. Hence, the dimensions of the state vector will be $\mathbb{R}^{2(K+1)\times 1}$ and the dimensions of the prediction covariance matrix will be $\mathbb{R}^{2(K+1)\times 2(K+1)}$. Formally, these operations can be defined as:

Operations | **Augmentation**

- State vector ($\mathbf{x}$) augmentation:

$$\mathbf{x} \leftarrow \mathbf{x} \cup \mathbf{m_{k+1}}$$

- Prediction covariance matrix $\mathbf{C}_{\text{pred}}$ augmentation:

$$\begin{aligned}
&\mathbf{tempC} := \mathbf{0}_{\texttt{dim(x)}\times\texttt{dim(x)}}\\
&n := \texttt{dim}(\mathbf{C}_{\text{pred}})\\
&\mathbf{tempC}(1:n, 1:n) \leftarrow \mathbf{C}_{\text{pred}}\\
&\mathbf{tempC}(n+1:\text{end}, n+1:\text{end}) \leftarrow \mathbf{Inf}\\
&\mathbf{tempC}(1:2, n+1:\text{end}) \leftarrow \mathbf{tempC}(1:2, 1:2)\\
&\mathbf{tempC}(n+1:\text{end}, 1:2) \leftarrow \mathbf{tempC}(1:2, 1:2)\\
&\mathbf{tempC}(3:n, n+1:\text{end}) \leftarrow \mathbf{tempC}(3:n, 1:2)\\
&\mathbf{tempC}(n+1:\text{end}, 3:n) \leftarrow \mathbf{tempC}(1:2, 3:n)\\
&\mathbf{C}_{\text{pred}} \leftarrow \mathbf{tempC}
\end{aligned}$$

Q6. Create pseudocode that shows the main actions that you need to do in each cycle of the SLAM (prediction, updating, embedding of new landmarks, creation of a measurement vector and associated matrices, bookkeeping of the landmarks)

Pseudocode | **SLAM Main Loop**

```
initialize estimates and bookkeeping structure
foreach time instant do
    set house keeping flags
    landmark identification
    if new landmark available then
        augment state vector
        augment prediction covariance matrix
        update bookkeeping structure
    end
    find indices in of current landmarks in state vector
    if ¬no landmark is visible then
        construct measurement matrix H
        perform Kalman update step
    end
    else
        set estimates as current predictions
    end
    perform Kalman predict step
end
```

Q7. Implement the SLAM algorithm. Show the time development of the results of the algorithm in a graphical representation which is expanded and updated in each cycle of the algorithm. That is: in each cycle you add a marker in the plot that

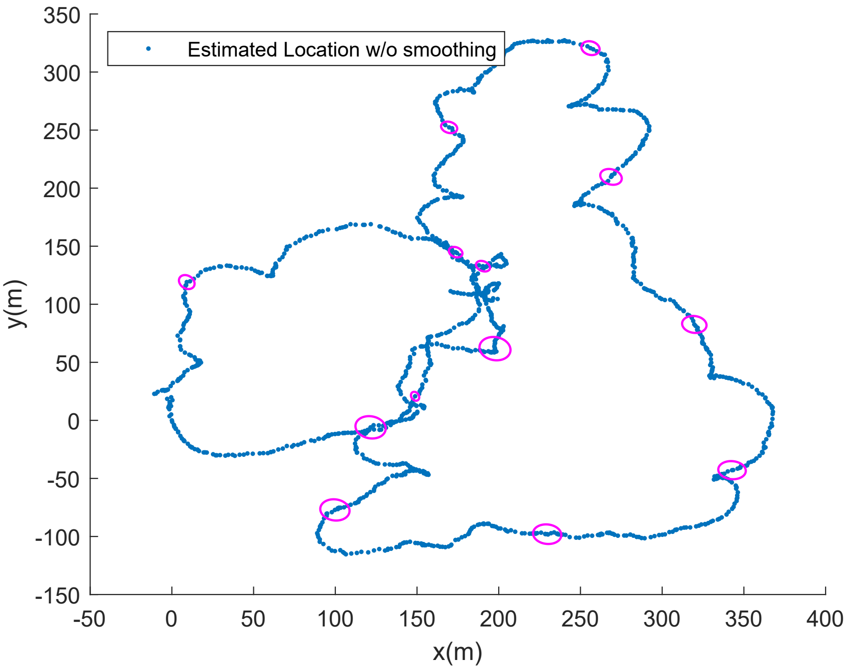


Figure 2: SLAM—w/o smoothing

indicates the current estimated position of the track; you add new landmarks in the plot by means of its uncertainty region; you update the uncertainty regions of already existing landmarks.

Refer part 2 of the script `ex8.m` provided in the Appendix.

Q8. Make a movie of this visualization. See the Matlab functions `getframe` and `writeVideo`. Also create a graph that you include in your report.

Refer fig. 2 and video `without-smoothing.avi` provided as a supplementary material.

Q9. Compare the results with respect to the track to the *prediction only* result.

In fig. 4b we can see the uncertainty of estimation decreases progressively where without application of SLAM technique it increases. This could be attributed to two major factors (i) tracking and updating landmarks as they are seen, and (ii) under the hood implementation of Kalman filter. For trajectory comparison refer fig. 3.

Part III: Smoothing

Q10. Implement the Rauch-Tung-Striebel algorithm. Visualize the results (movie+graph), and compare with the previous results.

Refer fig. 4 and videos `without-smoothing.avi` and `loop-closing.avi` provided as a supplementary material. From fig. 4b it can be seen that, after loop closing, the uncertainty is reduced this can be attributed to the fact that at a specific time instant all measurements are taken in to account to estimate the trajectory where as without loop closing only measurements from current and previous instant are considered to estimate the vehicle trajectory.

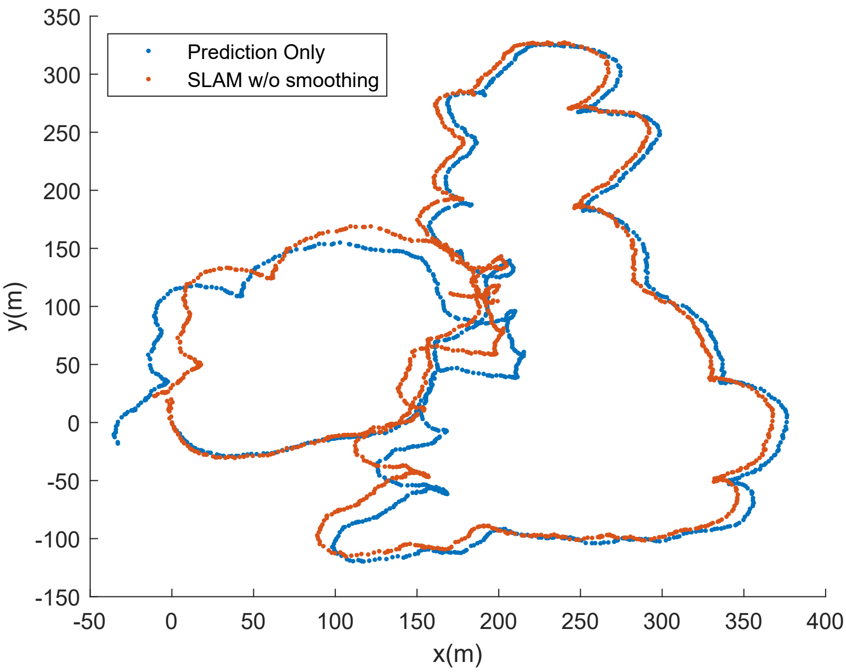


Figure 3: Vehicle trajectory before and after SLAM

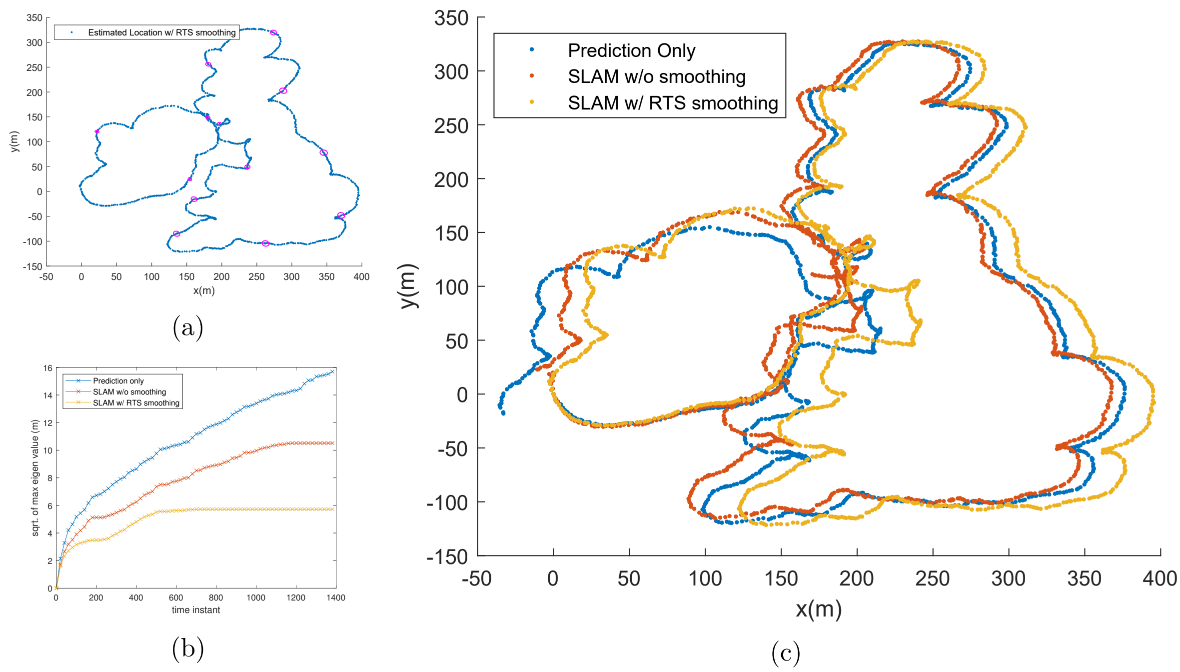


Figure 4: (a) Vehicle trajectory after RTS smoothing; (b) Performance metrics; (c) Comparison of estimated trajectories prediction only, SLAM with and without loop closing

Part IV: Bearing-only measurements

Q.11 Suppose that at a particular time i only two landmarks are visible. Suppose that the ids of the two landmarks are 200 and 21 (just an example). Suppose that both landmarks have been seen previously, and that their estimated position is embedded in the state vector at element 3 and 4 (landmark with id=21), and 7 and 8 (landmark with id 200). How do you define the measurement vector $\mathbf{z}(i)$ for this particular case? How is the corresponding measurement function $\mathbf{h}(\mathbf{x}(i))$ defined? How is the corresponding Jacobian matrix $\mathbf{H}(\mathbf{x}(i))$ defined? How is the covariance matrix of the measurement noise $\mathbf{C}_n$ defined?

Suppose total K landmarks are already seen. Measurement vector $\mathbf{z}(i)$ is defined as $\begin{bmatrix}\theta_{200} & \theta_{21}\end{bmatrix}$ and the corresponding measurement matrix,

$$\mathbf{H}(\mathbf{x}(i)) = \begin{bmatrix} \frac{\partial\theta_{200}}{\partial p_{x,200}} & \frac{\partial\theta_{200}}{\partial p_{y,200}} & \frac{\partial\theta_{200}}{\partial x_{200}} & \frac{\partial\theta_{200}}{\partial y_{200}} & 0 & 0 & 0 & 0 & 0 & \cdots \\ \frac{\partial\theta_{21}}{\partial p_{x,21}} & \frac{\partial\theta_{21}}{\partial p_{y,21}} & 0 & 0 & 0 & 0 & \frac{\partial\theta_{21}}{\partial x_21} & \frac{\partial\theta_{21}}{\partial y_21} & 0 & \cdots \end{bmatrix} \in \mathbb{R}^{2\times 2+K}$$

Covariance matrix of the measurement noise $\mathbf{C}_n = \begin{bmatrix}4 & 0\\ 0 & 4\end{bmatrix}$

Refer script `hmeas_bearing_only.m` provided in the Appendix for implementation of measurement function.

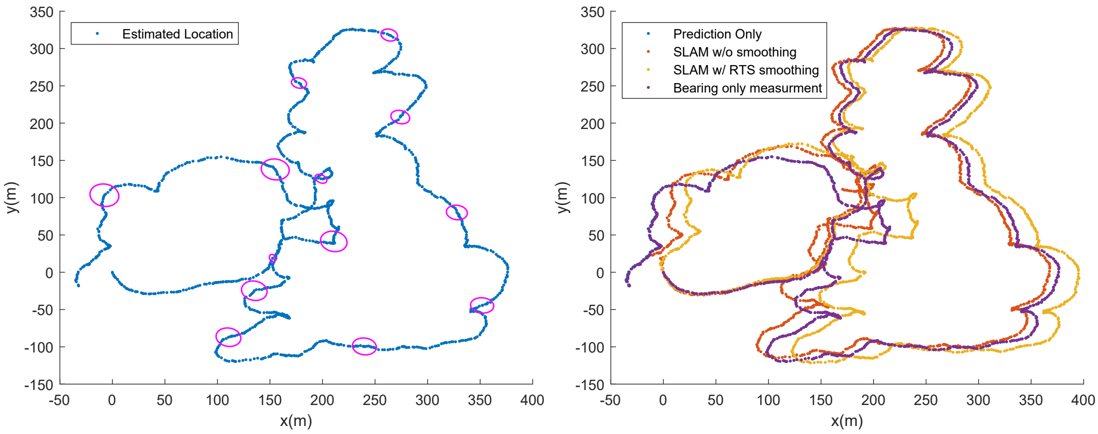
Q12. Implement the SLAM algorithm. Show the development of the algorithm in a graphical representation which is expanded and updated in each cycle of the algorithm. That is in each cycle: you add a marker in the plot that indicates the current estimated position of the track; you add new landmarks in the plot by means of its uncertainty region; you update the uncertainty regions of already existing landmarks. Now and then, you plot uncertainty regions of the track (as was done) in part I.

Refer script `ex8_bearing_only.m` provided in the Appendix for implementation of measurement function.

Q13. Make a movie of this visualization.

Refer the video `bearing-only.avi` provided as a supplementary material.

Q14. Compare the results with respect to the track to the previous results.



(a) (b)

Figure 5: a Trajectory of bearing-only SLAM; b Comparison with other trajectories

Appendix

Matlab implementation of `ex8.m`

```
%%Script Excercise 8 Part 1: Nimish Shah s2088894
close all; clear; clc;

%% load data file
load('SLAM.mat');

%% initializations
v = u(1, :);
phi = u(2, :);

no_iters = size(u, 2);
p = NaN(size(u, 1), no_iters+1);
p(:, 1) = [0; 0];

Cw_tilde = diag([std_velocity^2, std_heading^2]);

Cpredonly = NaN(size(u, 1), size(u, 1), no_iters);
Cpredonly(:, :, 1) = zeros(2);

%% Part I
F = eye(2);
for iter = 2:no_iters
    G = delta * (pi / 180) * [cosd(phi(iter-1)), -v(iter-1) * sind(phi(iter-1)); sind(phi(iter-1)), v(iter-1) * cosd(
        phi(iter-1))];
    p(:, iter) = F * p(:, iter-1) + delta * v(iter-1) * [cosd(phi(iter-1)); sind(phi(iter-1))];
    Cw = G * Cw_tilde * G';
    Cpredonly(:, :, iter) = F * Cpredonly(:, :, iter-1) * F' + Cw;
end

plot(p(1, :), p(2, :), '.', 'Color', '#0072BD');
hold on

% create animation showing evaluation of cov mat at each 100
for time_instant = 1:100:no_iters
    plot_cov_ellipse(p(:, time_instant), Cpredonly(:, :, time_instant), 'showMean', true);
end
xlabel("x(m)");
ylabel("y(m)");
legend("Predicted Location", 'Location', 'northwest')

%% Part II

%% initialize state vector
xest = cell(no_iters, 1); % pre-allocation of the estimated state vector
Cest = cell(no_iters, 1); % error covariance matrix
xpred = cell(no_iters+1, 1); % predicted state vector
Cpred = cell(no_iters+1, 1); % prediction covariance matrix

xpred{1} = zeros(2, 1);
Cpred{1} = zeros(2);

clear G;
G{1} = zeros(2);
unique_landmarks = [];
for time_instant = 1:no_iters
    unique_landmarks = vertcat(unique_landmarks, Z{time_instant}.id(:));
end
```

```matlab

max_unique_landmarks = length(unique(unique_landmarks));

%% initialize bookkeeping structure

LMBOOK.state_vector_ind = zeros(max_unique_landmarks+2, 1); % index of the landmark in the state vector
LMBOOK.state_vector_dim = 2; % dim of state vector
LMBOOK.total_visible = zeros(no_iters, 1); % no of total landmatks visible at specific instant
LMBOOK.visible = zeros(max_unique_landmarks, no_iters); % visibility of landmarks over time

%% start SLAM loop

for time_instant = 1:no_iters
    current_meas = Z{time_instant};
    LMBOOK.total_visible(time_instant) = length(current_meas.id);
    current_visible_lms = current_meas.id;
    is_new_lm_available = false;
    is_known_lm_available = false;
    is_no_landmark_visible = false;

    %% landmark identification
    % list known landmarks from visible
    [current_known_lms, ~, current_known_lm_ind] = intersect(current_visible_lms, LMBOOK.state_vector_ind, 'stable');
    if ~isempty(current_known_lms)
        is_known_lm_available = true;
    end
    no_current_known_lms = length(current_known_lms);

    % list new landmarks from visible
    if ~is_known_lm_available
        current_new_lms = current_visible_lms;
    else
        current_new_lms = current_visible_lms(~ismember(current_visible_lms, current_known_lms));
    end
    no_current_new_lm = length(current_new_lms);

    if ~isempty(current_new_lms)
        is_new_lm_available = true;
    end

    % no landmarks visible
    if ~(is_new_lm_available || is_known_lm_available)
        is_no_landmark_visible = true;
    end

    %% agumentation
    if is_new_lm_available
        LMBOOK.state_vector_ind(LMBOOK.state_vector_dim/2+1:LMBOOK.state_vector_dim/2+no_current_new_lm) = current_new_lms;
        LMBOOK.state_vector_dim = LMBOOK.state_vector_dim + 2 * no_current_new_lm;
        % augment state vector
        xpred{time_instant} = [xpred{time_instant}; zeros(2*no_current_new_lm, 1)]; %% add sensor readings instead of zeros
        % augment pred covariance matrix
        temp_cpred = zeros(LMBOOK.state_vector_dim);
        old_size = size(Cpred{time_instant}, 1);
        temp_cpred(1:old_size, 1:old_size) = Cpred{time_instant};
        temp_cpred(old_size+1:end, old_size+1:end) = 1000^2 * eye(2*no_current_new_lm); % very high uncertainity about current prediction
        temp_cpred(1:2, old_size+1:end) = repmat(temp_cpred(1:2, 1:2), 1, no_current_new_lm);
        temp_cpred(old_size+1:end, 1:2) = repmat(temp_cpred(1:2, 1:2), no_current_new_lm, 1);
        temp_cpred(3:old_size, old_size+1:end) = repmat(temp_cpred(3:old_size, 1:2), 1, no_current_new_lm);
        temp_cpred(old_size+1:end, 3:old_size) = repmat(temp_cpred(1:2, 3:old_size), no_current_new_lm, 1);
        Cpred{time_instant} = temp_cpred;
    end
    [~, ~, current_visible_lm_state_ind] = intersect(current_visible_lms, LMBOOK.state_vector_ind, 'stable');

    %% update
    if ~is_no_landmark_visible

        %% measurement function and measurement vector
        z = current_meas.zpos;
        H = zeros(2*LMBOOK.total_visible(time_instant), LMBOOK.state_vector_dim);
        H(1:end, 1:2) = repmat(-eye(2), LMBOOK.total_visible(time_instant), 1);
        for lm = 1:LMBOOK.total_visible(time_instant)
            ind = current_visible_lm_state_ind(lm) - 1;
            H(2*lm-1:2*lm, 2*ind+1:2*ind+2) = eye(2);
        end
        Cv = eye(2*LMBOOK.total_visible(time_instant)) * stdn^2;
        zpred = H * xpred{time_instant};
        S = H * Cpred{time_instant} * H' + Cv;
        K = Cpred{time_instant} * H' / S;

        xest{time_instant} = xpred{time_instant} + K * (z(:) - zpred);
        Cest{time_instant} = Cpred{time_instant} - K * S * K';
    else
        xest{time_instant} = xpred{time_instant};
        Cest{time_instant} = Cpred{time_instant};
    end

    %% predict
    xpred{time_instant+1} = xest{time_instant};
    xpred{time_instant+1}(1:2) = F * xest{time_instant}(1:2) + delta * v(time_instant) * [cosd(phi(time_instant)), sind(phi(time_instant))]';
    G{time_instant+1} = delta * (pi / 180) * [cosd(phi(time_instant)), -v(time_instant) * sind(phi(time_instant)); sind(phi(time_instant)), v(time_instant) * cosd(phi(time_instant))];

    Cpred{time_instant+1} = Cest{time_instant};
    Cpred{time_instant+1}(1:2, 1:2) = F * Cest{time_instant}(1:2, 1:2) * F' + G{time_instant} * Cw_tilde * G{
```

```
        time_instant}';

end

%% plot
figure;
hold on
for time_instant = 1:no_iters
    xplotx(time_instant) = xest{time_instant}(1);
end

for time_instant = 1:no_iters
    xploty(time_instant) = xest{time_instant}(2);
end
plot(xplotx, xploty, '.', 'Color', '#0072BD');

for time_instant = 1:100:no_iters
    plot_cov_ellipse([xplotx(time_instant); xploty(time_instant)], Cest{time_instant}(1:2, 1:2))
end
xlabel("x(m)");
ylabel("y(m)");
legend("Estimated Location w/o smoothing", 'Location', 'northwest')

%% Part III
xrts{no_iters, 1} = xest{no_iters};
Crts{no_iters, 1} = Cest{no_iters};

for time_instant = no_iters - 1:-1:1
    Q = size(xest{time_instant+1}, 1);
    P = size(xest{time_instant}, 1);
    del = Q - P;

    F = eye(Q, P);
    F(end-del+1:end, :) = 0;

    C = Cest{time_instant} * F' / Cpred{time_instant+1};
    xrts{time_instant} = xest{time_instant} + C * (xrts{time_instant+1} - xpred{time_instant+1});
    Crts{time_instant} = Cest{time_instant} + C * (Crts{time_instant+1} - Cpred{time_instant+1}) * C';
end

%% plot
figure;
hold on
for time_instant = 2:no_iters
    xplotx_rts(time_instant) = xrts{time_instant}(1);
end

for time_instant = 2:no_iters
    xploty_rts(time_instant) = xrts{time_instant}(2);
end
plot(xplotx_rts, xploty_rts, '.', 'Color', '#0072BD');

for time_instant = 2:100:no_iters
    plot_cov_ellipse([xplotx_rts(time_instant); xploty_rts(time_instant)], Crts{time_instant}(1:2, 1:2))
end
xlabel("x(m)");
ylabel("y(m)");
legend("Estimated Location w/ RTS smoothing", 'Location', 'northwest')

%% Comparisions
% plot predicted + SLAM w/o smoothing
figure;
hold on
plot(p(1, :), p(2, :), '.', 'Color', '#0072BD');
plot(xplotx, xploty, '.', 'Color', '#D95319');
xlabel("x(m)")
ylabel("y(m)")
legend("Prediction Only", "SLAM w/o smoothing", 'Location', 'northwest');

% plot predicted + SLAM w/o smoothing + SLAM w/ smoothing
figure;
hold on
plot(p(1, :), p(2, :), '.', 'Color', '#0072BD');
plot(xplotx, xploty, '.', 'Color', '#D95319');
plot(xplotx_rts, xploty_rts, '.', 'Color', '#EDB120');
xlabel("x(m)")
ylabel("y(m)")
legend("Prediction Only", "SLAM w/o smoothing", "SLAM w/ RTS smoothing", 'Location', 'northwest');


%% performance plots

eigen_predonly = nan(2, no_iters);
for i = 1:no_iters
    eigen_predonly(:, i) = eig(Cpredonly(:, :, i));
    sqrt_max_eigen(i) = sqrt(max(eigen_predonly(:)));
end

eigen_slam = nan(2, no_iters);
for i = 1:no_iters
    eigen_slam(:, i) = eig(Cest{i}(1:2, 1:2));
    sqrt_max_eigen_slam(i) = sqrt(max(eigen_slam(:)));
end

eigen_rts = nan(2, no_iters);
for i = 2:no_iters
    eigen_rts(:, i) = eig(Crts{i}(1:2, 1:2));
    sqrt_max_eigen_rts(i) = sqrt(max(eigen_rts(:)));
end
```

```

figure;
plot(1:20:no_iters, sqrt_max_eigen(1:20:end), '-x')
hold on
plot(1:20:no_iters, sqrt_max_eigen_slam(1:20:end), '-x')
plot(1:20:no_iters, sqrt_max_eigen_rts(1:20:end), '-x')
xlabel("time instant")
ylabel("sqrt. of max eigen value (m)")
legend("Prediction only","SLAM w/o smoothing", "SLAM w/ RTS smoothing","Bearing only predition", 'Location', 'northwest');
%% Videos
% SLAM w/o smoothing
v = VideoWriter('without-smoothing.avi', 'Motion JPEG AVI');
open(v);
figure;

for time_instant=1:no_iters
    clf
    title('Estimated Vehicle Trajectory using KF-SLAM w/o Smoothing')
    ylim([-150,350])
    xlim([-50,450])
    hold on
    plot(xplotx(1:time_instant), xploty(1:time_instant), '.', 'Color', '#0072BD');
    plot_cov_ellipse([xplotx(time_instant); xploty(time_instant)], Cest{time_instant}(1:2, 1:2), 'showMean', true, 'color', 'b', 'labels', ["x (m)", "y (m)"])
    frame = getframe(gcf);
    writeVideo(v,frame);
end
close(v);
% Loop closing only
v = VideoWriter('with-smoothing.avi', 'Motion JPEG AVI');
open(v);
figure;

for time_instant=2:no_iters
    clf
    title('Estimated Vehicle Trajectory using KF-SLAM w/ RTS Smoothing')
    ylim([-150,350])
    xlim([-50,450])
    hold on
    plot(xplotx_rts(1:time_instant), xploty_rts(1:time_instant), '.', 'Color', '#0072BD');
    plot_cov_ellipse([xplotx_rts(time_instant); xploty_rts(time_instant)], Crts{time_instant}(1:2, 1:2), 'showMean', true, 'color', 'b', 'labels', ["x (m)", "y (m)"])
    frame = getframe(gcf);
    writeVideo(v,frame);
end
close(v);

% SLAM+loop closing video
v = VideoWriter('loop-closing.avi', 'Motion JPEG AVI');
open(v);
figure;


for time_instant=1:no_iters
    clf
    title('SLAM Loopclosing')
    ylim([-150,350])
    xlim([-50,450])
    hold on
    plot(xplotx(1:time_instant), xploty(1:time_instant),'.', 'Color', '#0072BD')
    plot_cov_ellipse([xplotx(time_instant); xploty(time_instant)], Cest{time_instant}(1:2, 1:2), 'showMean', true, 'color', 'b', 'labels', ["x (m)", "y (m)"])

    legend("Estimated Location w/o Smoothing")
    frame = getframe(gcf);
    writeVideo(v,frame);
end

for time_instant=no_iters:-1:2
    clf
    hold on
    plot(xplotx, xploty,'.', 'Color', '#0072BD')
    title('SLAM Loopclosing')
    ylim([-150,350])
    xlim([-50,450])

    plot(xplotx_rts(no_iters:-1:time_instant), xploty_rts(no_iters:-1:time_instant), '.', 'Color', '#D95319');
    plot_cov_ellipse([xplotx_rts(time_instant); xploty_rts(time_instant)], Crts{time_instant}(1:2, 1:2), 'showMean', true, 'color', 'b', 'labels', ["x (m)", "y (m)"])

    legend("Estimated Location w/o Smoothing","Estimated Location w/ Smoothing")
    frame = getframe(gcf);
    writeVideo(v,frame);
end

close(v);
```

Matlab implementation of ex8_bearing_only.m

```
%%Script Excercise 8 Part 1: Nimish Shah s2088894
close all; clear; clc;

%% load data file
load('SLAM.mat');

```

```
%% initializations
v = u(1, :);
phi = u(2, :);
no_iters = size(u, 2);
F = eye(2);

xest{1} = zeros(1, 1);
Cest{1} = zeros(1);
G{1} = zeros(2);

std_meas_noise_mtr = 5;
std_new_landmark_mtr = 22;

Cw_tilde = diag([std_velocity^2, std_heading^2]);


xpred = zeros(2, no_iters);
no_visible_landmarks = 0;

%% Part IV

%% initialize state vector
xest = cell(no_iters, 1); % pre-allocation of the estimated state vector
Cest = cell(no_iters, 1); % error covariance matrix
xpred = cell(no_iters+1, 1); % predicted state vector
Cpred = cell(no_iters+1, 1); % prediction covariance matrix

xpred{1} = zeros(2, 1);
Cpred{1} = zeros(2);

clear G;
G{1} = zeros(2);
unique_landmarks = [];
for time_instant = 1:no_iters
    unique_landmarks = vertcat(unique_landmarks, Z{time_instant}.id(:));
end

max_unique_landmarks = length(unique(unique_landmarks));

%% initialize bookkeeping structure

LMBOOK.state_vector_ind = zeros(max_unique_landmarks+2, 1); % index of the landmark in the state vector
LMBOOK.state_vector_dim = 2; % dim of state vector
LMBOOK.total_visible = zeros(no_iters, 1); % no of total landmatks visible at specific instant
LMBOOK.visible = zeros(max_unique_landmarks, no_iters); % visibility of landmarks over time

%% start SLAM loop

for time_instant = 1:no_iters
    current_meas = Z{time_instant};
    LMBOOK.total_visible(time_instant) = length(current_meas.id);
    current_visible_lms = current_meas.id;
    is_new_lm_available = false;
    is_known_lm_available = false;
    is_no_landmark_visible = false;

    %% landmark identification
    % list known landmarks from visible
    [current_known_lms, ~, current_known_lm_ind] = intersect(current_visible_lms, LMBOOK.state_vector_ind, 'stable');
    if ~isempty(current_known_lms)
        is_known_lm_available = true;
    end
    no_current_known_lms = length(current_known_lms);

    % list new landmarks from visible
    if ~is_known_lm_available
        current_new_lms = current_visible_lms;
    else
        current_new_lms = current_visible_lms(~ismember(current_visible_lms, current_known_lms));
    end
    no_current_new_lm = length(current_new_lms);

    if ~isempty(current_new_lms)
        is_new_lm_available = true;
    end

    % no landmarks visible
    if ~(is_new_lm_available || is_known_lm_available)
        is_no_landmark_visible = true;
    end

    %% agumentation
    if is_new_lm_available
        LMBOOK.state_vector_ind(LMBOOK.state_vector_dim/2+1:LMBOOK.state_vector_dim/2+no_current_new_lm) =
            current_new_lms;
        LMBOOK.state_vector_dim = LMBOOK.state_vector_dim + 2 * no_current_new_lm;
        % augment state vector
        r = 22;
        [~, ~, current_new_lms_id] = intersect(current_new_lms, current_meas.id, 'stable');
        for lm = 1: no_current_new_lm
            xpred{time_instant} = [xpred{time_instant}; [r*cosd(current_meas.zbearing(current_new_lms_id(lm))); r*sind
                (current_meas.zbearing(current_new_lms_id(lm)))]+xpred{time_instant}(1:2)];
        end
        % augment pred covariance matrix
        temp_cpred = zeros(LMBOOK.state_vector_dim);
        old_size = size(Cpred{time_instant}, 1);
        temp_cpred(1:old_size, 1:old_size) = Cpred{time_instant};
        temp_cpred(old_size+1:end, old_size+1:end) = 1000^2 * eye(2*no_current_new_lm); % very high uncertainity about
            current prediction
```

```
        temp_cpred(1:2, old_size+1:end) = repmat(temp_cpred(1:2, 1:2), 1, no_current_new_lm);
        temp_cpred(old_size+1:end, 1:2) = repmat(temp_cpred(1:2, 1:2), no_current_new_lm, 1);
        temp_cpred(3:old_size, old_size+1:end) = repmat(temp_cpred(3:old_size, 1:2), 1, no_current_new_lm);
        temp_cpred(old_size+1:end, 3:old_size) = repmat(temp_cpred(1:2, 3:old_size), no_current_new_lm, 1);
        Cpred{time_instant} = temp_cpred;
    end
    [~, ~, current_visible_lm_state_ind] = intersect(current_visible_lms, LMBOOK.state_vector_ind, 'stable');

    %% update
    if ~is_no_landmark_visible

        %% measurement function and measurement vector
        z = current_meas.zbearing;
        Cv = eye(LMBOOK.total_visible(time_instant)) * 2^2;
        [zpred, Hjacobian] = hmeas_bearing_only(xpred{time_instant}, current_visible_lm_state_ind);
        S = Hjacobian * Cpred{time_instant} * Hjacobian' + Cv;
        K = Cpred{time_instant} * Hjacobian' / S;

        xest{time_instant} = xpred{time_instant} + K * wrapTo180(z - zpred);
        Cest{time_instant} = Cpred{time_instant} - K * S * K';
    else
        xest{time_instant} = xpred{time_instant};
        Cest{time_instant} = Cpred{time_instant};
    end

    %% predict
    xpred{time_instant+1} = xest{time_instant};
    xpred{time_instant+1}(1:2) = F * xest{time_instant}(1:2) + delta * v(time_instant) * [cosd(phi(time_instant)), sind(phi(time_instant))]';
    G{time_instant+1} = delta * (pi / 180) * [cosd(phi(time_instant)), -v(time_instant) * sind(phi(time_instant)); sind(phi(time_instant)), v(time_instant) * cosd(phi(time_instant))];

    Cpred{time_instant+1} = Cest{time_instant};
    Cpred{time_instant+1}(1:2, 1:2) = F * Cest{time_instant}(1:2, 1:2) * F' + G{time_instant} * Cw_tilde * G{time_instant}';

end

%% plot
figure;
hold on
for time_instant = 1:no_iters
    xplotxx(time_instant) = xest{time_instant}(1);
end

for time_instant = 1:no_iters
    xplotyy(time_instant) = xest{time_instant}(2);
end
plot(xplotxx, xplotyy, '.', 'Color', '#0072BD');

for time_instant = 1:100:no_iters
    plot_cov_ellipse([xplotxx(time_instant); xplotyy(time_instant)], Cest{time_instant}(1:2, 1:2))
end
xlabel("x(m)");
ylabel("y(m)");
legend("Estimated Location", 'Location', 'northwest')

%% performance plots

eigen_slam = nan(2, no_iters);
for i = 1:no_iters
    eigen_slam(:, i) = eig(Cest{i}(1:2, 1:2));
    sqrt_max_eigen_slam(i) = sqrt(max(eigen_slam(:)));
end

figure;
hold on
plot(1:20:no_iters, sqrt_max_eigen_slam(1:20:end), '-x')
xlabel("time instant")
ylabel("sqrt. of max eigen value (m)")
legend("SLAM w/o smoothing", 'Location', 'northwest');
%% video
v = VideoWriter('bearing-only-loop-closing.avi', 'Motion JPEG AVI');
open(v);
figure;

for time_instant=1:no_iters
    clf
    title('Bearing Only SLAM')
    ylim([-150,350])
    xlim([-50,450])
    hold on
    plot(xplotx(1:time_instant), xploty(1:time_instant),'.', 'Color', '#0072BD')
    plot_cov_ellipse([xplotx(time_instant); xploty(time_instant)], Cest{time_instant}(1:2, 1:2), 'showMean', true, 'color', 'b', 'labels', ["x (m)", "y (m)"])

    legend("Estimated Location", 'Location', 'northwest')
    frame = getframe(gcf);
    writeVideo(v,frame);
end

close(v);
```

Matlab implementation of hmeas_bearing_only.m

```
function [zpred, Hjacobian] = hmeas_bearing_only(xpred, state_ind)
```

```

%% initializations
p = xpred(1:2);
no_visible_lm = length(state_ind);
zpred = zeros(no_visible_lm, 1);
Hjacobian = zeros(no_visible_lm, length(xpred));
%% predict
for lm = 1:no_visible_lm
    xm = xpred(2*state_ind(lm)-1);
    ym = xpred(2*state_ind(lm));
    zpred(lm) = atan2d(ym-p(2), xm-p(1));
    Hjacobian(lm, 1) = -(ym - p(2)) / ((xm - p(1))^2 + (ym - p(2))^2);
    Hjacobian(lm, 2) = (xm - p(1)) / ((xm - p(1))^2 + (ym - p(2))^2);
    Hjacobian(lm, 2*state_ind(lm)-1) = (ym - p(2)) / ((xm - p(1))^2 + (ym - p(2))^2);
    Hjacobian(lm, 2*state_ind(lm)) = -(xm - p(1)) / ((xm - p(1))^2 + (ym - p(2))^2);

end
    Hjacobian = (pi / 180) * Hjacobian;
end
```

Matlab implementation of `plot_cov_ellipse.m`

```
function plot_cov_ellipse(mu, covMat, varargin)
% PLOT_COV_ELLIPSE plots covariance matrix as a linear a transformation
%         specified by mean M and covariance C Dimentionality of
%         should match M and C
%
% Syntax:   PLOT_COV_ELLIPSE(mu, covMat, 'showMean' true, ...
%                'legendText', "abc", 'labels', ["x","y"])
%
% Inputs:
%   REQUIRED:
%    mu - mean of the distribution
%    covMat - covariance of the distribution
%
%   OPTIONAL PARAMETERS:
%    'showMean' - BOOLEAN to show mean
%    'legendText' - STRING ARRAY to show legend
%    'labels' - STRING ARRAY to show axis labels x followed by y
%                default valye ["x", "y"]
%    'color' - specify color for ellipse only std matlab colors
%                      are valid
defaultShowMean = false;
defaultLegendText = '';
defaultLabels = ["x", "y"];
defaultColor = 'm';
validateLabels = @(x) isequal(size(x), [1, 2]);
validateColor = @(x) isequal(size(x), [1, 1])&& ischar(x)|| isequal(size(x), [1, 3]);
p = inputParser;
addRequired(p, 'mu');
addRequired(p, 'covMat');
addParameter(p, 'showMean', defaultShowMean, @islogical);
addParameter(p, 'legendText', defaultLegendText);
addParameter(p, 'labels', defaultLabels, validateLabels);
addParameter(p, 'color', defaultColor, validateColor);
parse(p, mu, covMat, varargin{:});
if ~(all(numel(p.Results.mu) == size(p.Results.covMat)))
    error('Dimensionality of mu and covMat must match');
end
thetas = linspace(0, 360, 720);
[V, D] = eig(p.Results.covMat);
a = sqrt(diag(D));
points = V * [cosd(thetas) * a(1); sind(thetas) * a(2)] + p.Results.mu;
plot(points(1, :), points(2, :), p.Results.color, 'LineWidth', 1);
hold on
if (p.Results.showMean)
    plot(mu(1), mu(2), '*', 'MarkerSize', 5, 'color' , p.Results.color);
end
if ~(strcmp(p.Results.legendText, ''))
    legend(p.Results.legendText);
end
xlabel(p.Results.labels(1));
ylabel(p.Results.labels(2));
end
```

E N D # #