프로젝트 보고서

〈Line Detection, EKF-based Pose Estimation〉

프로젝트	자율주행 자동차 공학
프로젝트 기간	2021.09.01 ~ 2021.12.31
이름	신건희

1. Spatial Domain image Filtering and Line Detection with Hough Transform

1. Abstract

자율주행 자동차 공학 전공 수업에서 인지 프로젝트로 차선이 포함되어 있는 사진에서 차선을 검출한다. 원본 이미지와 노이즈를 첨가한 이미지에서 차선의 길이와 각도 등 특징을 통해 Filtering, Canny Edge Detection, Hough Transform을 이용하여 차선을 검출한다.

1. 원본 이미지 Line Detection

1. 필터를 사용하여 Edge Detection을 위한 이미지 만들기

- Gaussian Filter와 sharpening Filter를 사용하여 Convolution 해본 결과 Sharpening Filter를 이용한 것이 더 효과적으로 Edge를 검출하는 것을 확인했다.

```
H1=[[-1,-1,-1];[-1,9,-1];[-1,-1,-1]];   % Gaussian Filter
H2=[[0,0,0];[0,2,0];[0,0,0]]-H1;        % Sharpening Filter


img_filtered_gau=imfilter(img,H1,'conv');   % Filtering the original image
using Gaussian Filter
img_filtered_sha=imfilter(img,H2,'conv');   % Filtering the original image
using Sharpening Filter
% Sharpening Filter expressed Edge better than Gaussian Filter
```

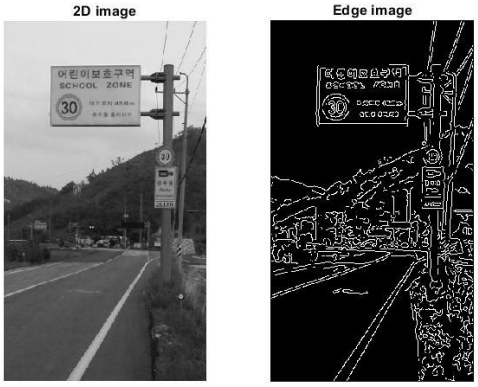
<FIG.1 Edge Detection by Gaussian Filter and Sharpening Filter >

2. Edge detection과 Hough Transformation으로 차선 검출하기

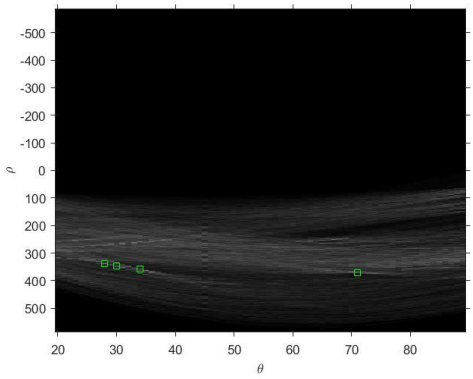
- 먼저 이미지를 2차원 배열로 바꿔준 후 'canny' 방식의 Edge detection을 했다. 그리고 Hough transform과 Houghpeaks을 통해 차선을 검출했다. Hough transform하는 과정에서 theta를 20~89로 한정했다.
- 먼저 theta값을 Default로 먼저 진행했다. 그러나 houghline 함수에서 FillGap 값과 MinLength값을 수정을 해도 사진 내에 가로등 근처에 있는 세로 수직의 선이 계속 차선과 함께 검출되었다.

- 그래서 theta를 20~89로 제한함으로써 가로등 근처의 수직에 가까운 선들을 모두 제거했다.
- 보통의 차선은 다양한 각도를 갖고 있으므로 theta를 제한하는 것은 바람직하지 않지만 본 프로젝트에서의 사진에 있는 차선을 정확히 검출하기 위해서 theta를 제한을 한 채로 진행했다.

```
img_2D=img(:,:,1);
img_Edge=edge(img_2D,'canny');      %canny Edge detection
[H,T,R] = hough(img_Edge,'Theta',20:89);            %Hough Transform,

P=houghpeaks(H,4,'threshold',ceil(0.3*max(H(:))));
% To detect lines in the image, the number of peaks should be more than 4

imshow(H,[],'XData',T,'YData',R,...
            'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color',"green"); % In the Hough graph, peak
values are expressed in a green box.
```



<FIG.2 2D image and Edge image>



<FIG.3 Hough Graph>

3. 이미지에 차선으로 판단되는 선을 표현

- Houghlines 함수를 이용하여 차선으로 판단되는 선을 검출했다. 여러 후보군들의 간격을 나타내는 FIllGap 변수 값을 15로 설정하여 사진 내에 뒤쪽에 있는 차선을 검출하게 했고 라인의 최소 길이를 나타내는 Minlength를 150으로 설정하여 차선의 후보들 중 차선이 아닌 짧은 선들을 제거했다.
- 원본 이미지 위에 차선과 차선 시작, 끝 점을 표시했다.

```
lines = houghlines(img_Edge,T,R,P,'FillGap',15,'MinLength',150);
% Detected edge due to noise was signigicantly removed from the fillgap of 15.
%Long-running noise was removed from the MinLength 150.
```

```
figure, imshow(img), hold on

for k = 1:length(lines)
   road_lane = [lines(k).point1; lines(k).point2];

% Plot beginnings and ends of lines
   plot(road_lane(1,1),road_lane(1,2),'x','LineWidth',2,'Color','yellow');
   plot(road_lane(2,1),road_lane(2,2),'x','LineWidth',2,'Color','red');
end
```

<FIG.4 Line detection in original image>

4, Visualization

```
figure; subplot(1, 3, 1);
imshow(img_noise); title('noise image');
subplot(1, 3, 2);
imshow(img_Edge); title('Edge image');
subplot(1, 3, 3);
imshow(img), hold on
max_len = 0;
for k = 1:length(lines)
   road_lane = [lines(k).point1; lines(k).point2];

   plot(road_lane(:,1),road_lane(:,2),'LineWidth',2,'Color','green');

   % Plot beginnings and ends of lines
   plot(road_lane(1,1),road_lane(1,2),'x','LineWidth',2,'Color','yellow');
   plot(road_lane(2,1),road_lane(2,2),'x','LineWidth',2,'Color','red');
end
title('line detection');
```

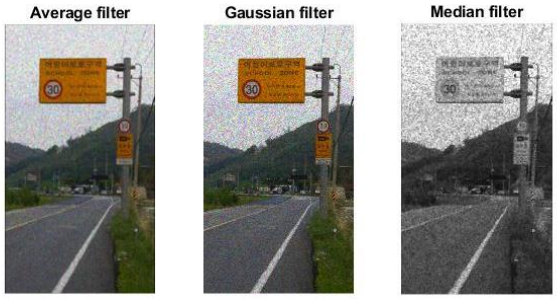
<FIG.5 Process of line detection in original image>

2. 노이즈를 첨부한 이미지 Line Detection

1. 이미지에 노이즈를 첨부하고 필터를 사용하여 노이즈를 제거하기

   ➢ 0.05의 분산의 'speckle' 방식으로 노이즈를 이미지에 첨부했다. 그리고 Average filter, Gaussian filter, Median filter로 노이즈를 제거했다.

```
img = imread('image3.jpg');

% Add noise into the image
img_noise = imnoise(img, 'speckle', 0.05);

H=1/9*[[1,1,1];[1,1,1];[1,1,1]]; %Average Filter

img_denoise_avg=imfilter(img_noise,H,'conv');   % Noise removal using an
average filter.
img_denoise_gau=imgaussfilt(img_noise);         % Noise removal using an
Gaussian filter.
img_noise_2D=img_noise(:,:,1);                 % Transform 3D image to 2D
image.
img_denoise_med=medfilt2(img_noise_2D);         % Noise removal using Median
filter

figure;
subplot(1,3,1);
imshow(img_denoise_avg); title('Average filter');
subplot(1,3,2);
imshow(img_denoise_gau); title('Gaussian filter');
subplot(1,3,3);
imshow(img_denoise_med); title('Median filter');
```



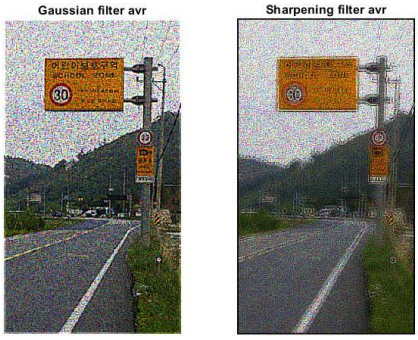
<FIG.6 Denoised images>

2. 필터를 사용하여 Edge-detection을 위한 이미지 만들기
   - Gaussian Filter와 Sharpening Filter를 사용하여 각각의 이미지를 convolution했다.
   - 결과를 확인 후 Gauissian filter로 노이즈를 제거하고 Gaussian filter로 convolution한 이미지를 선택하여 계속 진행했다.

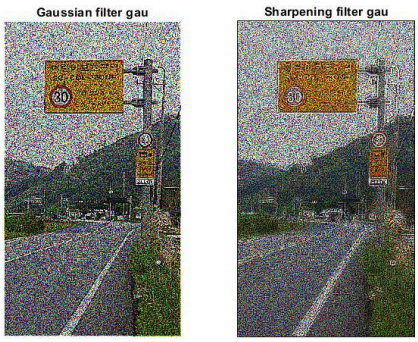
```
%% 1) Apply Filter (Select the type of filter)
H1=[[-1,-1,-1];[-1,9,-1];[-1,-1,-1]];   % Gaussian Filter
H2=[[0,0,0];[0,2,0];[0,0,0]]-H1;        % Sharpening Filter


img_filtered_gau=imfilter(img,H1,'conv');
% Filtering the original image using Gaussian Filter
img_filtered_sha=imfilter(img,H2,'conv');
% Filtering the original image using Sharpening Filter
% Sharpening Filter expressed Edge better than Gaussian Filter

img_filtered_avr_gau=imfilter(img_denoise_avg,H1,'conv');
% Filtering the denoised image(by Averager filter) using Gaussian Filter
img_filtered_avr_sha=imfilter(img_denoise_avg,H2,'conv');
% Filtering the denoised image(by Averager filter) using Sharpening Filter

img_filtered_gau_gau=imfilter(img_denoise_gau,H1,'conv');
% Filtering the denoised image(by Gaussian filter) using Gaussian Filter
img_filtered_gau_sha=imfilter(img_denoise_gau,H2,'conv');
% Filtering the denoised image(by Gaussian filter) using Sharpening Filter

img_filtered_med_gau=imfilter(img_denoise_med,H1,'conv');
% Filtering the denoised image(by Median filter) using Sharpening Filter
img_filtered_med_sha=imfilter(img_denoise_med,H2,'conv');
% Filtering the denoised image(by Median filter) using Sharpening Filter


figure;
subplot(1,2,1);
imshow(img_filtered_gau); title('Gaussian filter');
subplot(1,2,2);
imshow(img_filtered_sha); title('Sharpening filter');

figure;
subplot(1,2,1);
```

```
imshow(img_filtered_avr_gau); title('Gaussian filter avr');
subplot(1,2,2);
imshow(img_filtered_avr_sha); title('Sharpening filter avr');

figure;
subplot(1,2,1);
imshow(img_filtered_gau_gau); title('Gaussian filter gau');
subplot(1,2,2);
imshow(img_filtered_gau_sha); title('Sharpening filter gau');

figure;
subplot(1,2,1);
imshow(img_filtered_med_gau); title('Gaussian filter med');
subplot(1,2,2);
imshow(img_filtered_med_sha); title('Sharpening filter med');

% As a result of filtering, i proceeded a process with a image which is
denoised by Gaussian filter and also edge detected by Gaussian Filter
```

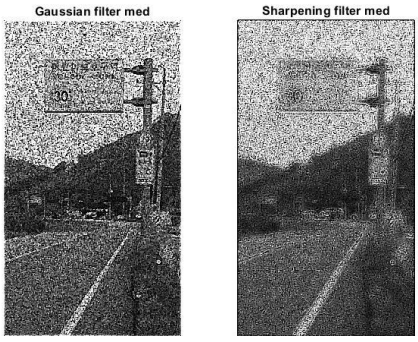
<FIG.7 Original image>



<FIG.8 Denoised by Average filter >



<FIG.9 Denoised by Gaussian filter>



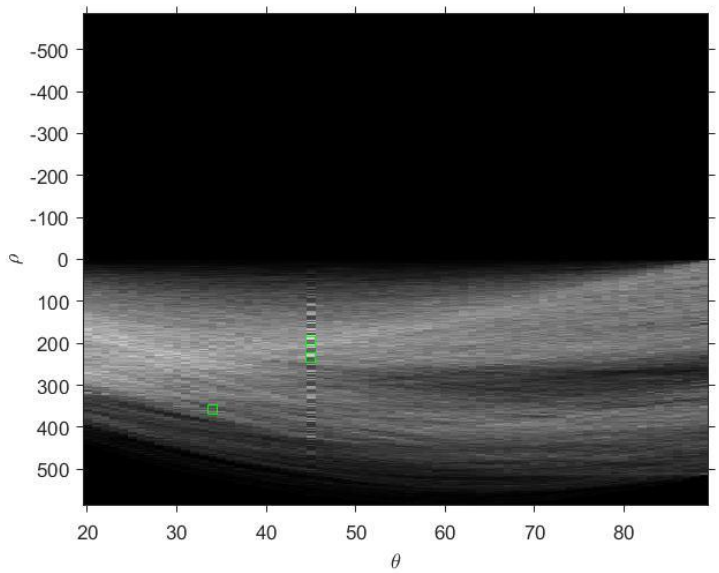
<FIG.10 Denoised by Median filter>

3, Edge detection과 Hough Transformation으로 차선 검출하기

- 먼저 이미지를 2차원 배열로 바꿔준 후 'canny' 방식의 Edge detection을 했다. 그리고 Hough transform과 Houghpeaks을 통해 차선을 검출했다. Hough transform하는 과정에서 theta를 20~89로 한정했다.
- Theta가 20 이상부터 가로등 근처의 세로 선들과 하늘에서 검출되는 선들이 제거되었다.
- 보통 차선은 다양한 각도를 갖고 있으므로 theta를 한정하는 것은 바람직하지 않으나 해당 과제

의 사진에서 정확한 차선 검출을 위해 theta를 제한했다.

```
img_2D=img_filtered_avr_gau(:,:,1); % Change the 3D image to the 2D image
img_Edge=edge(img_2D,'canny');      %canny Edge detection

[H,T,R] = hough(img_Edge,'Theta',20:89);          %Hough Tranform,

% By limitng theta to 20~89, all lines close to vertical near streetlights were
removed
P=houghpeaks(H,3,'threshold',ceil(0.3*max(H(:))));
% To detect lines in the image, the number of peaks should be more than 3

imshow(H,[],'XData',T,'YData',R,...
           'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color',"green");
% In the Hough graph, peak values are expressed in a green box.
```



<FIG.11 Hough Graph>

4. 이미지에 차선으로 판단되는 선을 검출

- Houghlines 함수를 이용하여 차선으로 판단되는 선을 검출했다. 여러 후보군들의 간격을 나타내는 FIllGap 변수 값을 5로 설정하여 노이즈로 인한 작은 선들을 제거할 수 있었고 라인의 최소 길이를 나타내는 Minlength를 150으로 설정하여 길게 이어진 노이즈 선들을 제거했다.
- 원본 이미지 위에 차선과 차선 시작, 끝 점을 표시했다

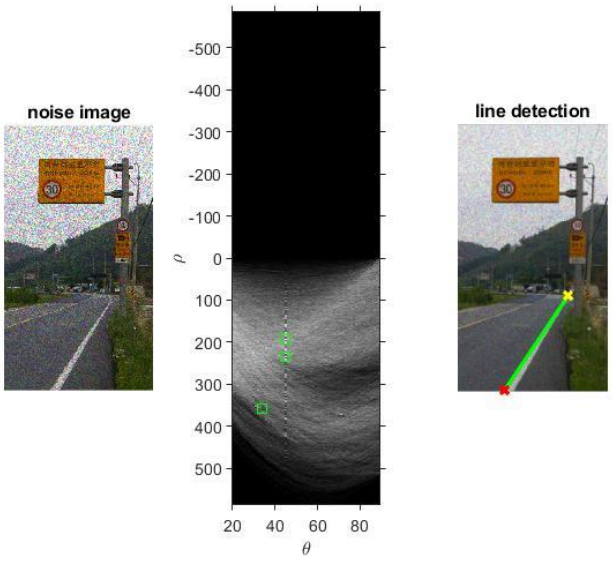
```
lines = houghlines(img_Edge,T,R,P,'FillGap',5,'MinLength',150);
% Detected edge due to noise was signigicantly removed from the fillgap of 5.

%Long-running noise was removed from the MinLength 150.
```

```
figure, imshow(img_denoise_avg), hold on
for k = 1:length(lines)
   road_lane = [lines(k).point1; lines(k).point2];

   plot(road_lane(:,1),road_lane(:,2),'LineWidth',2,'Color','green');

   % Plot beginnings and ends of lines
   plot(road_lane(1,1),road_lane(1,2),'x','LineWidth',2,'Color','yellow');
   plot(road_lane(2,1),road_lane(2,2),'x','LineWidth',2,'Color','red');
end
```

<FIG.12 Line detection in noised image>

5. Visualization

```
figure; subplot(1, 3, 1);
imshow(img_noise); title('noise image');
subplot(1,3,2);
imshow(H,[],'XData',T,'YData',R,...
            'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color',"green");
subplot(1,3,3);
imshow(img_denoise_avg), hold on
max_len = 0;
for k = 1:length(lines)
   road_lane = [lines(k).point1; lines(k).point2];

   plot(road_lane(:,1),road_lane(:,2),'LineWidth',2,'Color','green');

   % Plot beginnings and ends of lines
   plot(road_lane(1,1),road_lane(1,2),'x','LineWidth',2,'Color','yellow');
   plot(road_lane(2,1),road_lane(2,2),'x','LineWidth',2,'Color','red');
end
title('line detection');
```



<FIG.13 Process of line detection in noised image>

2. EKF-based Pose Estimation Using Lidar Sensor and Data

1. Abstract

자율주행 자동차 공학 전공 수업에서 인지 부분 프로젝트로 EKF를 통해 차량의 Pose를 측정한다. 0s~500s동안 반복적으로 차량 Velocity, Orientation 데이터와 라이다 데이터를 통해 Measurement, Motion model을 사용하여 차량의 Pose를 예상하고 수정한다. 차량에는 매우 간단한 형태의 Lidar가 부착되어 각각의 랜드마크를 측정하고 Range and Bearing measurements를 얻는다. Landmark의 Global positions은 사전에 알고 있다고 가정한다. 랜드마크를 측정한 데이터가 어떤 데이터인지 알 수 있는 Data association 또한 사전에 알고 있다고 가정한다.

2. Unpack the Data

- Input signal
  - v= Translational velocity input [1 X 501]
  - om= Rotational velocity input [1 X 501]
- Data associated with Lidar
  - l= x,y position of landmarks, landmark는 8개 존재한다. [8 X 2]
  - b= Bearing to each landmarks center in the frame [501 X 8]
  - r= Range measurements [501 X 8]

- d= distance between vehicle center and laser rangefinder, assumed to be 0

➢ Meeasurement noise covariance matrices and covariance values

- v_var= translation velocity variance
- om_var= rotational velocity variance
- r_var= range measurements variance
- b_var= bearing measurements variance

➢ Initial values

- (x_init, y_init, th_init)=(50,0,1.5708rad)

➢ 변수들의 자료형은 int와 double이다. Matlab에서 식을 구성할 때 변수들의 자료형이 동일해야 하므로 int형인 변수들을 double형으로 변환했다.

```
t = double(pickle_data.t); % timestamps [s]
    x_init = pickle_data.x_init; % initial x position [m]
    y_init = pickle_data.y_init; % initial y position [m]
    th_init = pickle_data.th_init; % initial theta position [rad]

    % input signal
    v = pickle_data.v; % translational velocity input [m/s]
    om = pickle_data.om; % rotational velocity input [rad/s]

    % bearing and range measurements, LIDAR constants
    b = pickle_data.b;
   % bearing to each landmarks center in the frame ,!attached to the laser
    r = pickle_data.r; % range measurements [m]
    l = double(pickle_data.l); % x,y positions of landmarks [m]
    d = double(pickle_data.d);
   % distance between robot center and laser rangefinder ,![m]

    v_var = 0.01; % translation velocity variance
    om_var = 0.01; % rotational velocity variance

    r_var = 0.01; % range measurements variance
    b_var = 10; % bearing measurement variance

    Q_km=diag([v_var, om_var]);
    cov_y=diag([r_var,b_var]);

    x_est= zeros(1,3,size(v,2));   % size(v,2)=501
    P_est= zeros(3,3,size(v,2));

    x_est(1,:,1)=[x_init,y_init,th_init];
    P_est(:,:,1)=diag([1,1,0.1]);
```

3. Main Filter loop

1. Set the initial values

    1. Unpack the data에서 설정한 Pose와 Covariance matrix 초기값을 x_check, P_check 첫 번째 행열에 입력한다
    2. 500초 동안 500번의 EKF를 통해 Pose 추정을 하는데 각각의 추정값은 x_check, P_check에 저장한다.

2. Pose Estimation

    1. Pre-setting
        - 500초 동안 500번의 추정하므로 for문에서 범위는 2~501이다.
        - 500초 동안 500번의 추정하므로 시간 간격은 1s이며 wraptopi 함수는 theta값을 -pi~pi로 제한하는 사용자 정의 함수이다.
    2. Update state prediction at time t with odometry readings
        - Motion model: Motion model은 angular velocity와 odometry를 Input으로 2D pose를 Ouput으로 하는 선형식이다
        - T는 sampling time interval으로 본 알고리즘에서는 1s(Constant)이다.
        - Process noise W_k는 정규 분포 형식으로 분산은 상수 Q이다.

$$\mathbf{x}_k = \mathbf{x}_{k-1} + T \begin{bmatrix} \cos\theta_{k-1} & 0 \\ \sin\theta_{k-1} & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \mathbf{w}_k \right), \quad \mathbf{w}_k = \mathcal{N}(\mathbf{0}, \mathbf{Q})$$

<FIG.14 Motion Model Equation>

    3. Update Propagate Uncertainty Prediction at time t with Motion Model
        - Nonlinear Motion Model을 Taylor Expansion을 통해 Linearization을 한 식에서 Motion Model Jacobian Matric을 구할 수 있다.
        - Jacobian Matric를 통해 Propagate Uncertainty를 Prediction한다.

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}) \approx \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}) + \underbrace{\left.\frac{\partial \mathbf{f}_{k-1}}{\partial \mathbf{x}_{k-1}}\right|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}}}_{\mathbf{F}_{k-1}} (\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}) + \underbrace{\left.\frac{\partial \mathbf{f}_{k-1}}{\partial \mathbf{w}_{k-1}}\right|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}}}_{\mathbf{L}_{k-1}} \mathbf{w}_{k-1}$$

<FIG.15 Linearized Motion Model>

$$\mathbf{F}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_{k-1}}\right|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, 0}, \quad \mathbf{L}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{w}_k}\right|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, 0}.$$

<FIG.16 Jacobian matric of Motion Model>

$$\check{\mathbf{x}}_k = \mathbf{f}\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}\right)$$
$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T$$

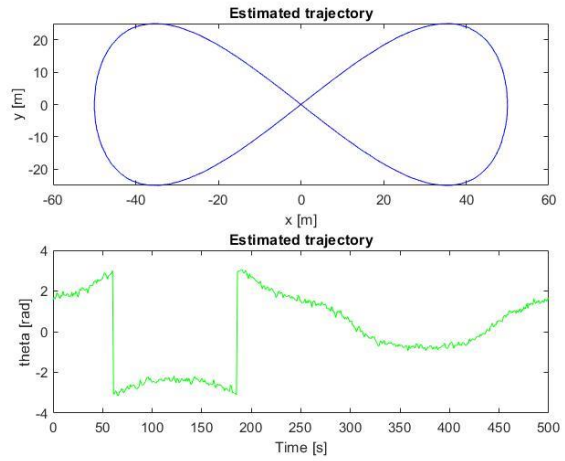
<FIG.17 Pose Prediction Equation>

4. Update State Estimate using Available Landmark Measurements

   - 각 Time Interval에 8개의 Landmark를 측정하고 Measurement 모델을 통해 Pose를 Correction 한다.
   - Measurement_update(l(i,:), r(k,i), b(k,i), P_check, x_check, d, cov_y)함수는 사용자 정의 함수로 선언했다.
   - 각 Time Interval에서 Correction된 Pose값은 x_est 행렬에 차례대로 입력되며 Correction 된 Pose와 Propagate Uncertainty는 바로 다음 Pose Prediction과 Correction에 사용된다.

5. Visualization

   - 3차원 행렬인 X_est를 2차원 그래프의 각 축에 나타내기 위해 reshape함수를 이용하여 1차원으로 변환했다.
   - 그래프를 통해 pose(x,y)와 시간에 따른 Orientation을 확인할 수 있다.



<FIG.18 Vehicle Pose Estimation>

```
%% Main Filter Loop

    % Set the initial values

    P_check=P_est(:,:,1);
    x_check=reshape(x_est(1,:,1),[3,1]);


    for k=2:501
    delta_t=t(k)-t(k-1); % time stamp
    theta=wraptopi(x_check(3));

    % Update state with odometry readings
    F=[cos(theta),0; sin(theta),0; 0,1];
    inp= [v(k-1);om(k-1)];


    % Prediction
    x_check=x_check+F*inp*delta_t;
    x_check(3)=wraptopi(x_check(3));

    % Motion Model Jacobian with respect to last state
    F_km=zeros(3,3);
    F_km=[1,0,-sin(theta)*delta_t*v(k-1);
        0,1,cos(theta)*delta_t*v(k-1);
        0,0,1];

    % Motion Model jacobian with respect to noise
    L_km=zeros(3,2);
    L_km=[cos(theta)*delta_t,0;
        sin(theta)*delta_t,0;
        0,1];

    % Propagate uncertainty
    P_check= (F_km*P_check*F_km.'+L_km*Q_km*L_km.');

    % Update state estimate using available landmark measurements
    for i=1:8

    [x_check,P_check]=measurement_update(l(i,:),r(k,i),b(k,i),P_check,x_check,d
    ,cov_y);
    end

    % Set final state predcitions for timestep
    x_est(1,1,k)=x_check(1);
    x_est(1,2,k)=x_check(2);
    x_est(1,3,k)=x_check(3);
    P_est(:,:,k)=P_check;

    end


    x_est_mod= reshape(x_est(:,1,:),1,size(x_est,3));
    y_est_mod= reshape(x_est(:,2,:),1,size(x_est,3));
    theta_mode=reshape(x_est(:,3,:),1,size(x_est,3));
```

```
figure
subplot(2,1,1);
plot(x_est_mod,y_est_mod,'b');
title('Estimated trajectory');
xlabel('x [m]');
ylabel('y [m]');

subplot(2,1,2);
plot(t(:),theta_mode,'g');
title('Estimated trajectory');
xlabel('Time [s]');
ylabel('theta [rad]');
```

4. Correction Step

1. measurement_update(l(i,:), r(k,i), b(k,i), P_check, x_check, d, cov_y)
   - Predicted Pose 값을 수정하는 사용자 정의 함수이다.
   - 입력 값으로는 사전에 알고 있는 landmark의 Global Position, Current Pose에서 Landmark 측정 값, Predicted Pose, Sensor와 Vehicle의 위치 차이, Mesurement 값의 Covariance이다.
2. Measurement Model
   - Predicted Pose에서 Landmark 측정 값을 이용한 Landmark 추정 값과 사전에 알고 있는 Landmark의 Global Position 차이를 이용하여 Measurement Model을 구한다.

$$\mathbf{y}_k^l = \begin{bmatrix} \sqrt{(x_l - x_k - d\cos\theta_k)^2 + (y_l - y_k - d\sin\theta_k)^2} \\ atan2\left(y_l - y_k - d\sin\theta_k, x_l - x_k - d\cos\theta_k\right) - \theta_k \end{bmatrix} + \mathbf{n}_k^l, \quad \mathbf{n}_k^l = \mathcal{N}(\mathbf{0}, \mathbf{R})$$

<FIG.19 Measurement Model>

3. Correction using Kalman Gain
   - Prediction과 마찬가지로 Nonlinear Measurement Model을 Taylor Expansion을 통해 선형화하고 Jacobian Matric of Measurement Model을 구한다.
   - Jacobian Matric을 통해 Kalman Gain을 구하고 Pose와 Propagate Uncertainty를 Correction한다.

$$\mathbf{y}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{v}_k) \approx \mathbf{h}_k(\check{\mathbf{x}}_k, \mathbf{0}) + \underbrace{\left.\frac{\partial \mathbf{h}_k}{\partial \mathbf{x}_k}\right|_{\check{\mathbf{x}}_k, \mathbf{0}}}_{\mathbf{H}_k} (\mathbf{x}_k - \check{\mathbf{x}}_k) + \underbrace{\left.\frac{\partial \mathbf{h}_k}{\partial \mathbf{v}_k}\right|_{\check{\mathbf{x}}_k, \mathbf{0}}}_{\mathbf{M}_k} \mathbf{v}_k$$

<FIG.20 Linearized Measurement Model>

$$\mathbf{K}_k = \check{\mathbf{P}}_k \mathbf{H}_k^T \left(\mathbf{H}_k \check{\mathbf{P}}_k \mathbf{H}_k^T + \mathbf{M}_k \mathbf{R}_k \mathbf{M}_k^T\right)^{-1}$$

<FIG.21 Kalman Gain>

$$\check{\mathbf{y}}_k^l = \mathbf{h}\left(\check{\mathbf{x}}_k, \mathbf{0}\right)$$
$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \mathbf{K}_k\left(\mathbf{y}_k^l - \check{\mathbf{y}}_k^l\right) \quad \hat{\mathbf{P}}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\check{\mathbf{P}}_k$$

<FIG.22 Pose Correction Equation>

```
%% Prediction 의 결과와 Measurement 결과를 통해 Correction 해주는 사용자 정의 함수

function [x_check,P_check]=measurement_update(lk,rk,bk,P_check,x_check,d,cov_y)
    x_k=x_check(1);
    y_k=x_check(2);
    theta_k=wraptopi(x_check(3));

    x_l=lk(1);
    y_l=lk(2);

    d_x=x_l-x_k-d*cos(theta_k);
    d_y=y_l-y_k-d*sin(theta_k);

    r = sqrt(d_x.^2+d_y.^2);
    phi = atan2(d_y,d_x)-theta_k;

    %Compute measurement Jacobian
    H_k=zeros(2,3);
    H_k(1,1) = -d_x/r;
    H_k(1,2) = -d_y/r;
    H_k(1,3) = d*(d_x*sin(theta_k) - d_y*cos(theta_k))/r;
    H_k(2,1) = d_y/r.^2;
    H_k(2,2) = -d_x/r.^2;
    H_k(2,3) = -1-d*(d_y*sin(theta_k) + d_x*cos(theta_k))/r.^2;
    H_K=double(H_k);

    M_k=eye(2,2);
    y_out=[r;wraptopi(phi)];
    y_mes=[rk;wraptopi(bk)];

    %Compute Kalman Gain
    K_k=(P_check*H_k.')/(H_k*P_check*H_k.'+M_k*cov_y*M_k.');

    %Correct predicted state
    x_check=x_check+K_k*(y_mes-y_out);
    x_check(3)= wraptopi(x_check(3));

    %Correct covariance
    P_check=(eye(3,3)-K_k*H_k)*P_check;

end %return x_check,P_check
```