EE183DA
Team Buffalo
Lab 2: Kalman Filter State Estimation Algorithm

Iou-Sheng (Danny) Chang
UID: 804-743-003

February 4th 2019

Contents

1 Introduction and Lab Overview

In this lab, we use Kalman Filter to estimate the state of the two-wheeled robot with sensor measurements. By using the developed mathematical model of the complete robot system, we implement and evaluate a Kalman Filter based state estimation algorithm.

Since the process dynamics of our system are non-linear but smooth, we may use its linearized approximation for the process model. The Block diagram of the two-wheeled robot system is shown in 1.

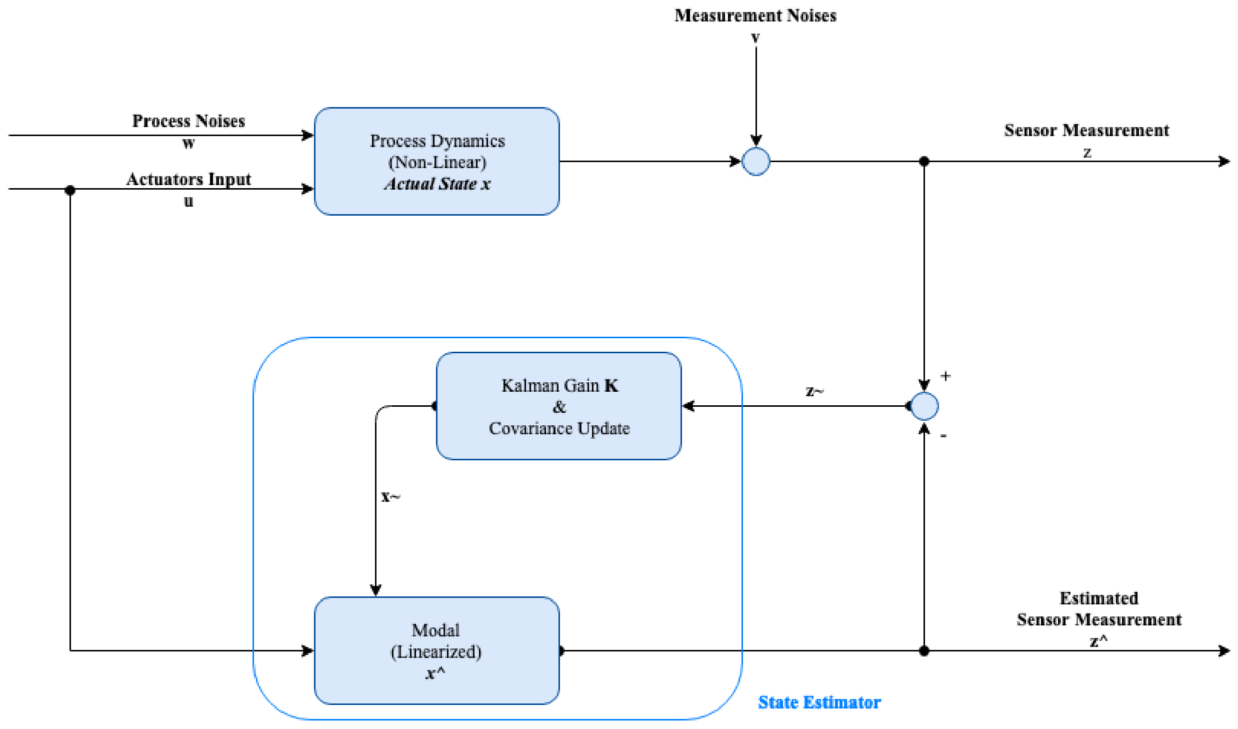

Figure 1: System Block Diagram

2 Symbols and Conventions

2.1 Symbols

List of all symbols used for deriving the theoretical, geometric model.

L	length of the box
W	width of the box
C_W	distance between left and right wheels
C_C	distance between rotational center and car reference point
C_f	distance between front lidar sensor and car reference point
C_r	distance between right lidar sensor and car reference point
PWM_L	input PWM of the left wheel
PWM_R	input PWM of the right wheel
v	linear velocity of the car
v_L	linear velocity of the left wheel
v_R	linear velocity of the right wheel
ω	angular velocity of the car
ω_L	angular velocity of the left wheel
ω_R	angular velocity of the right wheel
p_x	position: absolute x coordinate of the car
p_y	position: absolute y coordinate of the car
p_θ	absolute orientation of the car
θ_{MPU}	MPU9250 magnetometer angle measurement
l_f	front laser range sensor reading
l_r	right laser range sensor reading
δt	time traveled by the car
δt_L	time traveled by the left wheel
δt_R	time traveled by the right wheel
N_W	process noise
V	measurement noise
N_v	linear velocity noise
N_ω	angular velocity noise

2.2 Assumptions

The following assumptions are made according the the car's geometry and for the derivation of theoretical, geometric model.

1. (p_x, p_y) is taken as the car's reference point, which is in the top right corner of the car.
2. r_θ is measured from the car's rotational center, which is the mid-point between the two wheels.

3. When the car is going straight (either forward or backward), $v = v_L = v_R$, having same sign and magnitude.
4. When the car is turning (either clockwise or counter-clockwise), it is turning about its rotational center, In this case, $|v| = |v_L| = |v_R|$, however v_L and v_R have opposite sign.
5. $p_\theta = \theta_{MPU}$ assuming the MPU9250 magnetometer is calibrated.
6. $N_W = 0$,there is no process noise.

2.3 Systems

Input

$$u = \begin{bmatrix} PWM_L \\ PWM_R \end{bmatrix}$$

State

$$x = \begin{bmatrix} p_x \\ p_y \\ p_\theta \end{bmatrix}$$

Sensor Measurements

$$z = \begin{bmatrix} l_f \\ l_r \\ \theta_{MPU} \end{bmatrix}$$

3 Robot State Update Dynamics and Linearized Model

Recall the theoretical, geometric model of the robot state update dynamics, that we derived in lab 1, which has the form:

$$x'(t) = x_{t+1} = f(x_t, u_t, t) = Ax_t + Bu_t$$

In order to have complete control of the car, we split the movements of the car into two cases, traveling along a straight line (forward/backward) and turning about its rotational axis (clockwise/counter-clockwise).

Note that the following equations work for $p_{\theta_t}, \theta_{MPU} = [0, 2\pi]$, (all values of θ).

3.1 Case 1: Traveling along a straight line (forward/backward)

$$\begin{aligned} p_{x,t+1} &= p_{x,t} + (v_t)t \cdot \cos p_{\theta_t} \\ p_{y,t+1} &= p_{y,t} + (v_t)t \cdot \sin p_{\theta_t} \\ p_{\theta_{t+1}} &= p_{\theta_t} \end{aligned}$$

Expressing in matrix form yields:

$$\begin{aligned} x'(t) = x_{t+1} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{x,t} \\ p_{y,t} \\ p_{\theta_t} \end{bmatrix} + \begin{bmatrix} (v_t)t \cdot \cos p_{\theta_t} \\ (v_t)t \cdot \sin p_{\theta_t} \\ 0 \end{bmatrix} \\ &= I_3 x_t + \begin{bmatrix} (v_t)t \cdot \cos p_{\theta_t} \\ (v_t)t \cdot \sin p_{\theta_t} \\ 0 \end{bmatrix} \end{aligned}$$

Note that for forward mode $v_t > 0$, while for backward mode $v_t < 0$.

3.2 Case 2: Turning about its rotational axis (clockwise/counter-clockwise)

Note that $p_{x,t+1} \neq p_{x,t}$ and $p_{y,t+1} \neq p_{y,t}$ due to the fact that rotational axis is different from the car reference point.

$$\begin{aligned} p_{x,t+1} &= p_{x,t} - C_C[\cos p_{\theta_t} - \cos p_{\theta_{t+1}}] \\ p_{y,t+1} &= p_{y,t} + C_C[\sin p_{\theta_t} - \sin p_{\theta_{t+1}}] \\ p_{\theta_{t+1}} &= p_{\theta_t} + (\omega_t)t \end{aligned}$$

Expressing in matrix form yields:

$$\begin{aligned} x'(t) = x_{t+1} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{x,t} \\ p_{y,t} \\ \theta_t \end{bmatrix} + \begin{bmatrix} -C_C[\cos p_{\theta_t} - \cos p_{\theta_{t+1}}] \\ +C_C[\sin p_{\theta_t} - \sin p_{\theta_{t+1}}] \\ (\omega_t)t \end{bmatrix} \\ &= I_3 x_t + \begin{bmatrix} -C_C[\cos p_{\theta_t} - \cos p_{\theta_{t+1}}] \\ +C_C[\sin p_{\theta_t} - \sin p_{\theta_{t+1}}] \\ (\omega_t)t \end{bmatrix} \end{aligned}$$

Note that for counter-clockwise (left turn) mode $\omega_t < 0$, while for clockwise (right turn) mode $\omega_t > 0$.

3.3 Matlab Implementation

Listing 1: MATLAB code for Robot State Update Dynamics

```
function [y] = estimate(x,mode)
    % variables and constants
    A = eye(3);
    C_c = 56;
    theta = x(3);
    % Four modes of moving
    switch mode
        % Forward mode
        case 1
            PWM_R = 63;
            PWM_L = 107;
            t = 0.065;
            input = [PWM_R^2;
                     PWM_R;
                     1;
                     PWM_L^2;
                     PWM_L;
                     1];
            c = [-0.1863   19.934   -382.02   0        0        0;
                 0         0        0         0.016    -0.649   20.575];
            v = mean(c*input);
            u = [v;
                 v;
                 theta];
            B = [t*cosd(theta)    0                0;
                 0                -t*sind(theta)   0;
                 0                0                0];
            y = A*x + B*u;
        % Backward mode
        case 2
            PWM_R = 104;
            PWM_L = 65;
            t = 0.065;
            input = [PWM_R^2;
                     PWM_R;
                     1;
                     PWM_L^2;
                     PWM_L;
                     1];
            c = [0.2874  -64.469  3462.5   0        0        0;
                 0       0        0        0.2789   -31.524  735.09];
            v = mean(c*input);
            u = [-v;
                 -v;
                 theta];
            B = [-t*cosd(theta)   0                0;
                 0                t*sind(theta)    0;
                 0                0                0];
            y = A*x + B*u;
        % Left turn mode
        case 3
            PWM_R = 64;
            PWM_L = 65;
            t = 0.045;
            input = [PWM_R^2;
                     PWM_R;
                     1;
                     PWM_L^2;
                     PWM_L;
                     1];
            coeff = [-0.1062, 9.9628, -51.161, -0.0245, -2.6331,
               428.17];
            w = -1* mean(coeff*input);
            C_c= C_c*0.3;
            B = [-C_c*(cosd(theta) - cosd(theta + w*t));
                 C_c*(sind(theta) - sind(theta + w*t));
                 w*t];
            y = A*x + B;
        % Right turn mode
        case 4
            PWM_R = 114;
            PWM_L = 105;
            t = 0.03;
            input = [PWM_R^2;
                     PWM_R;
                     1;
                     PWM_L^2;
                     PWM_L;
                     1];
            coeff = [-0.0955, 23.74, -1287.7, -0.6161, 134.99,
               -7210.5];
            w = 1*mean(coeff*input);
            B = [C_c*(sind(theta) - sind(theta + w*t));
                 C_c*(cosd(theta) - cosd(theta + w*t));
                 w*t];
            %w = 1*mean(coeff*input);
            %B = [-C_c*(cosd(theta) - cosd(theta + w*t));
            %      C_c*(sind(theta) - sind(theta + w*t));
            %      w*t];
            y = A*x + B;
        otherwise
            y = x;
    end

    if y(3) < 0
        y(3) = y(3) + 360;
    end
    if y(3) > 360
        y(3) = y(3) - 360;
    end
end
```

4 Sensor Measurement and Linearized Model

In order to develop the idealized sensor measurement model, a geometric approach derived by the team is used.

4.1 Ideal Geometric Lidar Sensor Measurement

Process for determining the ideal geometric lidar sensor readings is listed in detail as follow.

1. Given the robot's current state (p_x, p_y, p_θ).
2. Draw a line perpendicular to the respective edge of the car that each of the lidar sensors sit on (front and right edge).
3. The line drawn will intersect with two out of the four edges of the box, assuming that these lines continue infinitely in both directions outside the box. Note that the equations of the edges of the box are:

$$\begin{aligned} &x = 0 \ (left\ edge\ of\ the\ box) \\ &x = L \ (right\ edge\ of\ the\ box) \\ &y = 0 \ (bottom\ edge\ of\ the\ box) \\ &y = W \ (top\ edge\ of\ the\ box) \end{aligned}$$

4. Figure 2 shows that out of the two intersections, one will occur right at the edge of the actual box (inside the boundary of the box), while the other will occur outside the boundary of the box.
5. Knowing the two intersection points, and the current state of the car, the distance between the car and each of the points is then calculated.
6. After the distances are calculated, we take the minimum of the two distances as the distance between the lidar sensor and the edge/wall of the box.
7. This gives us the ideal geometric lidar sensor reading.

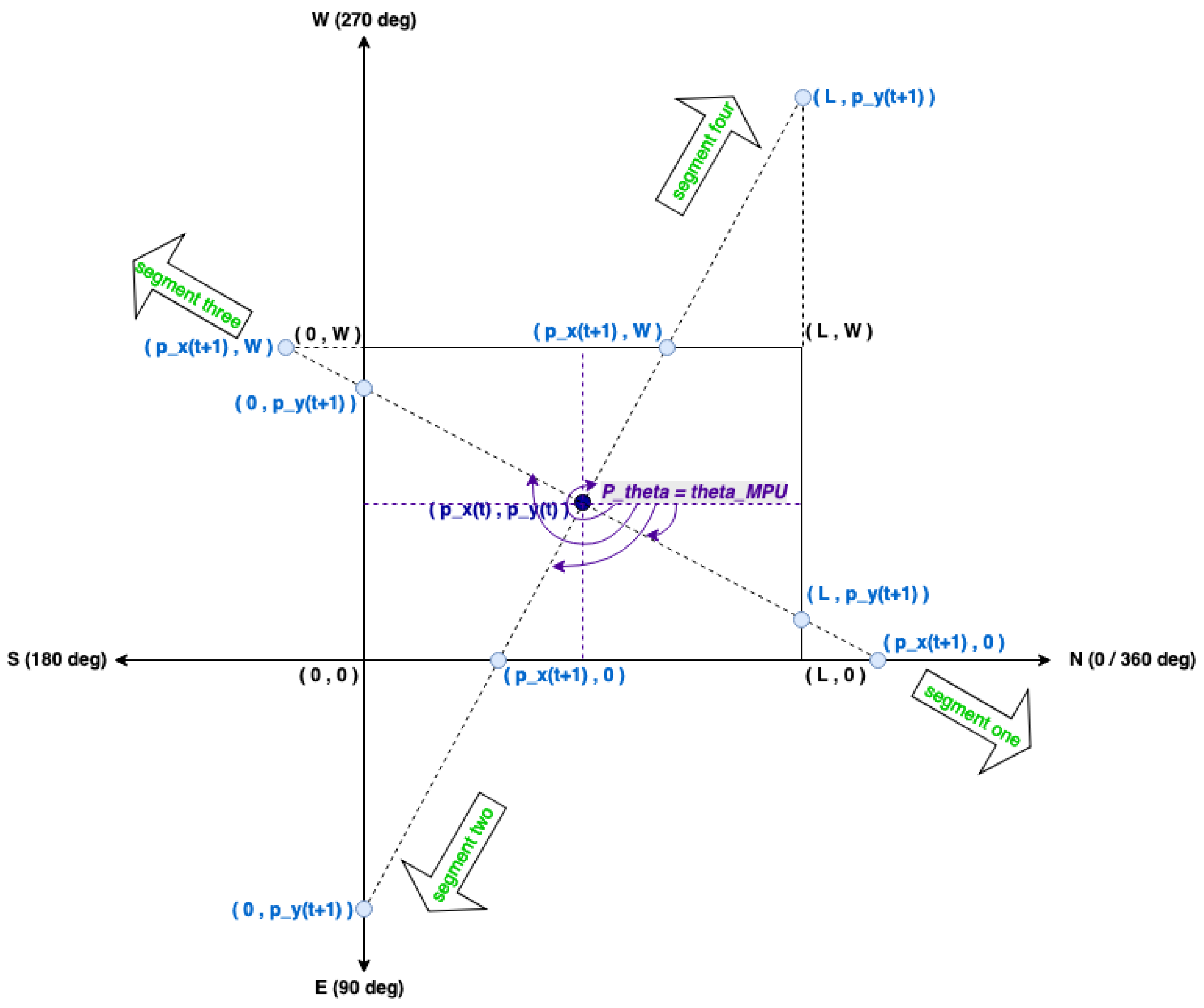

Figure 2: Sensor Geometric Model

4.2 Equations

In order to take into account of all cases ($p_{\theta_t}, \theta_{MPU} = [0, 2\pi]$), we separate into 4 cases. We first define the intersection points for different cases as:

$$Y_f = Front\ lidar\ sensor\ intersection's\ Y\ parameter$$
$$X_f = Front\ lidar\ sensor\ intersection's\ X\ parameter$$
$$Y_r = Right\ lidar\ sensor\ intersection's\ Y\ parameter$$
$$X_r = Right\ lidar\ sensor\ intersection's\ X\ parameter$$

$\forall p_{\theta_t} \in (0°, 90°)$

$$\begin{aligned} Y_f &= 0 \\ X_f &= L \\ Y_r &= 0 \\ X_r &= 0 \end{aligned}$$

$\forall p_{\theta_t} \in (90°, 180°)$

$$\begin{aligned} Y_f &= 0 \\ X_f &= 0 \\ Y_r &= W \\ X_r &= 0 \end{aligned}$$

$\forall p_{\theta_t} \in (180°, 270°)$

$$\begin{aligned} Y_f &= W \\ X_f &= 0 \\ Y_r &= W \\ X_r &= L \end{aligned}$$

$\forall p_{\theta_t} \in (270°, 360°)$

$$\begin{aligned} Y_f &= W \\ X_f &= L \\ Y_r &= 0 \\ X_r &= L \end{aligned}$$

The Front lidar sensor's two intersections can then be calculated using the following equations.

$$\begin{aligned} p'_x &= p_x + \frac{p_y - Y_f}{tan(p_\theta)} \\ p'_y &= p_y + (p_x - X_f)tan(p_\theta) \end{aligned}$$

The Right lidar sensor's two intersections can then be calculated using the following equations.

$$\begin{aligned} p'_x &= p_x + (Y_r - p_y)tan(p_\theta) \\ p'_y &= p_y + \frac{X_r - p_x}{tan(p_\theta)} \end{aligned}$$

Distances of the two intersections in each segment as shown in Figure 2, can be obtained by the following equations.

$$r_{x_f} = r_{x_r} = \sqrt{(p'_x - p_x)^2 + (Y - p_y)^2}$$
$$r_{y_f} = r_{y_r} = \sqrt{(X - p_x)^2 + (p'_y - p_y)^2}$$

4.3 Linearization

Assume that from previous session, we found the minimum distance of the two intersections, therefore knowing the correct point of intersection with the box and the actual lidar reading.

We define some more new symbols to help simplify the linearization cases.

(X_I, Y_I)	Actual intersection point
L_I	Actual lidar reading (distance to the box/wall)

where

$$L_I = \sqrt{(p_{x_0} - X_I)^2 + (p_{y_0} - Y_I)^2}$$

Linearizing the equations from previous session about $(p_{x_0}, p_{y_0}, p_{\theta_0})$ gives us $z_{3\times1} = H_{3\times3}x_{3\times1} + D_{3\times1}$ for Extended Kalman Filter.

Expressing in matrix form yields:

$$\begin{bmatrix} l_f \\ l_r \\ \theta_{MPU} \end{bmatrix} = z = \begin{bmatrix} \frac{p_{x_0}-X_I}{L_I} & \frac{p_{y_0}-Y_I}{L_I} & 0 \\ \frac{p_{x_0}-X_I}{L_I} & \frac{p_{y_0}-Y_I}{L_I} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_\theta \end{bmatrix} + \begin{bmatrix} -p_{x_0}\frac{p_{x_0}-X_I}{L_I} \\ -p_{x_0}\frac{p_{y_0}-Y_I}{L_I} \\ L_I \end{bmatrix}$$
$$= Hx + D$$
$$H = \begin{bmatrix} \frac{p_{x_0}-X_I}{L_I} & \frac{p_{y_0}-Y_I}{L_I} & 0 \\ \frac{p_{x_0}-X_I}{L_I} & \frac{p_{y_0}-Y_I}{L_I} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$D = \begin{bmatrix} -p_{x_0}\frac{p_{x_0}-X_I}{L_I} \\ -p_{x_0}\frac{p_{y_0}-Y_I}{L_I} \\ L_I \end{bmatrix}$$

By plugging in different intersection points into the linearlized equation, we can derive the $H_{3\times3}$ and $D_{3\times1}$ matrices for all different cases as listed in previous sessions.

4.4 Special Cases

As the intersection points are found using the slope ($tan(p_\theta)$) of the line that passes through the front lidar sensor, there are four special cases that we need to be aware of.

Due to the fact that

$$tan(0°) = tan(180°) = tan(360°) = 0$$

$$tan(90°) = tan(270°) = undefined$$

the above equations will not hold for these special cases, hence we derive new $H_{3\times3}$ and $D_{3\times1}$ matrices for these special cases.

$\forall p_{\theta_t} = 0°\ and\ 360°$

$$H = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} L \\ 0 \\ 0 \end{bmatrix}$$

$\forall p_{\theta_t} = 90°$

$$H = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$\forall p_{\theta_t} = 180°$

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \\ W \\ 0 \end{bmatrix}$$

$\forall p_{\theta_t} = 270°$

$$H = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} W \\ L \\ 0 \end{bmatrix}$$

4.5 Matlab Implementation

Listing 2: MATLAB code for Sensor Measurement Linearized Model

```matlab
function [H, D] = linearize(x)
    % Symbols and Constants
    L = 762.5;  % Length of the box [mm]
    W = 495;    % Width of the box [mm]
    Lf_c = 50;  % front lidar sensor to the car sensor reference
       point [mm]
    Lr_c = 53;  % right lidar sensor to the car sensor reference
       point [mm]

    % MPU9250
    p_theta = x(3);
    % Lidar
    p_x = x(1);
    p_y = x(2);

    if p_theta < 0
        p_theta = p_theta + 360;
    end
    if p_theta > 360
        p_theta = p_theta - 360;
    end

    % Cases
    switch p_theta
    case 0
        H = [-1 0   0;
             0  1   0;
             0  0   1];
        D = [L;
             0;
             0];
    case 360
        H = [-1 0   0;
             0  1   0;
             0  0   1];
        D = [L;
             0;
             0];
    case 180
        H = [1  0   0;
             0  -1  0;
             0  0   1];
        D = [0;
             W;
```

```
                 0];
    case 90
        H = [0   1    0;
             1   0    0;
             0   0    1];
        D = [0;
             0;
             0];
    case 270
        H = [0   -1   0;
             -1  0    0;
             0   0    1];
        D = [0;
             0;
             0];
    otherwise
        if p_theta < 0
            p_theta = p_theta + 360;
        end
        if p_theta < 90
            xf = L;
            yf = 0;
            xr = 0;
            yr = 0;
        elseif p_theta <180 && p_theta>90
            xf = 0;
            yf = 0;
            xr = 0;
            yr = W;
        elseif p_theta <270 && p_theta>180
            xf = 0;
            yf = W;
            xr = L;
            yr = W;
        elseif p_theta <360 && p_theta>270
            xf = L;
            yf = W;
            xr = L;
            yr = 0;
        end

        % Lidar (Test in process)
        %p_y = x(2) + Lf_c*cosd(p_theta);
        %f_p_x = x(1) + Lf_c*sind(p_theta);
        %r_p_x = x(1) - Lf_c*sind(p_theta);
        %fx = f_p_x + (p_y - yf) / tan(p_theta);
        %fy = p_y + (f_p_x - xf) * tan(p_theta);
        %rx = r_p_x + (yr - p_y) * tan(p_theta);
        %ry = p_y + (xr - r_p_x) / tan(p_theta);

        fx = p_x + (p_y - yf) / tan(p_theta);
        fy = p_y + (p_x - xf) * tan(p_theta);
        rx = p_x + (yr - p_y) * tan(p_theta);
        ry = p_y + (xr - p_x) / tan(p_theta);

        points = [xf fy;
                  fx yf;
                  rx yr;
                  xr ry];

        for i = 1:2
            % (Test in process)
%              if i == 1
%                  p_x = f_p_x;
%              else if i == 2
%                  p_x = r_p_x;
%              end
            tempA = pdist([p_x, p_y; points(2*i-1,:)]);
            tempB = pdist([p_x, p_y; points(2*i,:)]);
            if tempA < tempB
                p = [points(2*i-1,:) tempA];
            else
                p = [points(2*i,:) tempB];
            end
            H(i,:) = [(p_x - p(1))/p(3),(p_y - p(2))/p(3),0];
            D(i,:) = [(-p_x + p(1))/p(3)*p_x - (p_y - p(2))/p(3)*p_y
                + p(3)];
        end
        H(3,:) = [0 0 1];
        D(3) = 0;
    end
end
```

5 Kalman Filter Procedure

Since the process dynamics of our system are non-linear but smooth, we may use its linearized approximation for the process model, hence applying to Extended Kalman Filter, which is the non-linear version of Kalman Filter that linearizes about the current mean and covariance.

5.1 Predict

1. $\hat{x}^-_{t+1} = A\hat{x}^+_t + Bu_t$
2. $P^-_{t+1} = AP^+_t A^T + Q$

5.2 Kalman Gain

1. $K_{t+1} = P^- C^T (CP^- C^T + R)^{-1}$

5.3 Measurement Update

1. $\hat{x}^+ = \hat{x}^- + K(z - H\hat{x}^-)$
2. $P^+ = (I - KH)P^-$

where

$$\begin{aligned} e &= x(t) - \hat{x}^+(t) \\ P &= E[ee^T] = Error\ Covariance \\ R &= E[vv^T] \\ Q &= E[ww^T] = 0 \end{aligned}$$

5.4 Matlab Implementation

Listing 3: MATLAB code for Kalman Filter

```matlab
%% EE 183DA
% Team Buffalo
% Lab 2 - Kalman Filter state estimator
clc; clear all; close all;

%% Load and subtract data
exp1 = load ('measure.txt');
exp2 = load ('measure_NEW.txt');
SENSOR = exp2(1:3,:);
LIDAR_F = exp2(1,:);
```

```
LIDAR_R = exp2(2,:);
MPU = exp2(3,:);
MODE = exp2(4,:);

%% Initial State
% exp1
%p_x = 150;
%p_y = 100;
%p_theta = 333;
% exp2
p_x = 50;
p_y = 200;
p_theta = 180;

x = [p_x; p_y; p_theta];

%% Sensor offset
offset = [28.236;
          63.228;
          0];
PtoL = [1.0065  0       0;
        0       1.0547  0;
        0       0       1];

%% Kalman Filter
% Variables and constant
I = eye(3);
A = I;
R = 0.1* eye(3);    % Measurement Noise
% Measurement Noise (Test in process)
%v = [];
%for i = 1:1000
%    v = [v, wgn(3,1,1)];
%end
%R = 0.01*v*v';

for i = 1:length(exp2)
    %estimation update
    x(:,i+1) = estimate(x(:,i),MODE(i));

    % z = Hx + V
    %mid = findmiddle(x(:,i))';
    [H,D] = linearize(x(:,i));
    %mid = findmiddle(x(:,i+1))';
    z(:,i) = PtoL * (H*x(:,i+1) + D) + offset;

    % (Test in process)
    %e = x(:,i) - x(:,i+1);
    %p = 0.01*e*e';
    p = 0.0001 * eye(3);

    % measurement unpade
    K = p*H.'*inv(H*p*H.'+R);
    x(:,i+1) = x(:,i+1) + K*(SENSOR(i)- z(:,i));
    p = (I - K*H) * p;
end

%% Plot the car state
plot_car_state(x);
```

6 Results

6.1 Car State Estimator

The resulted state estimator is plotted in a MATLAB figure, as shown in Figure 3.

In the MATLAB figure, the x axis limit and y axis limit is set to be the dimension of the box that the car is running in. The green line shows the complete path the car is following, and the red $*$ indicates the car reference point. To easier visualize how the car is performing in the box, we further plotted a rectangle indicating the car's body, while the dark blue line indicates the head of the car.

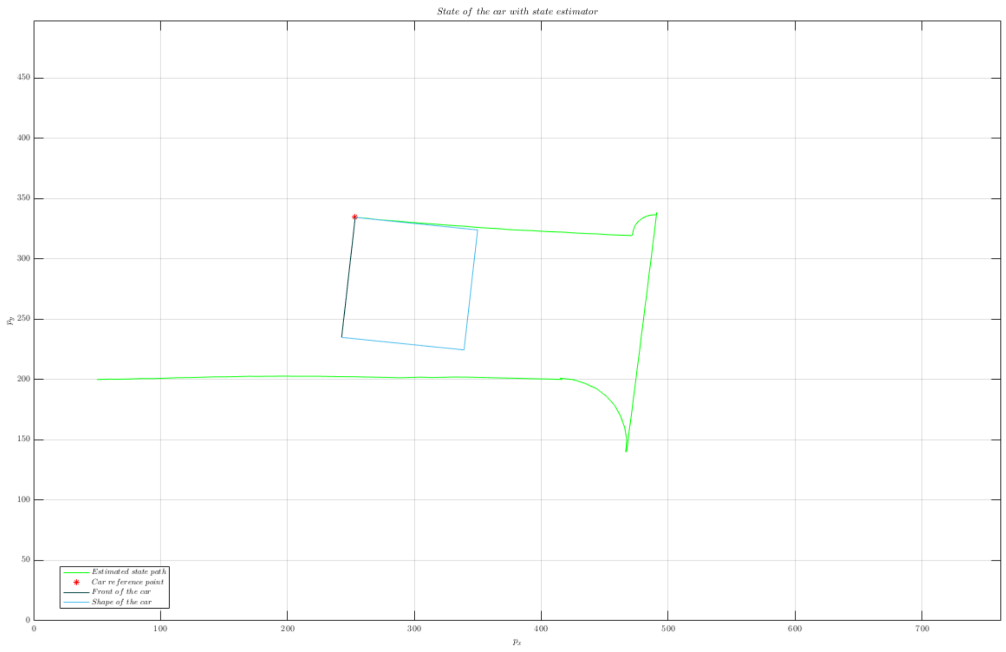

Figure 3: MATLAB Car State estimator

6.2 True State

In order to determine the true state of the car observed by an external observer, we used a birds-eye camera and created a box with grids drawn as shown in Figure 4. Each grid is precisely drawn to be 50×50 $[mm]$, therefore from the birds-eye view we can tell if the state estimated result is close to the true state.

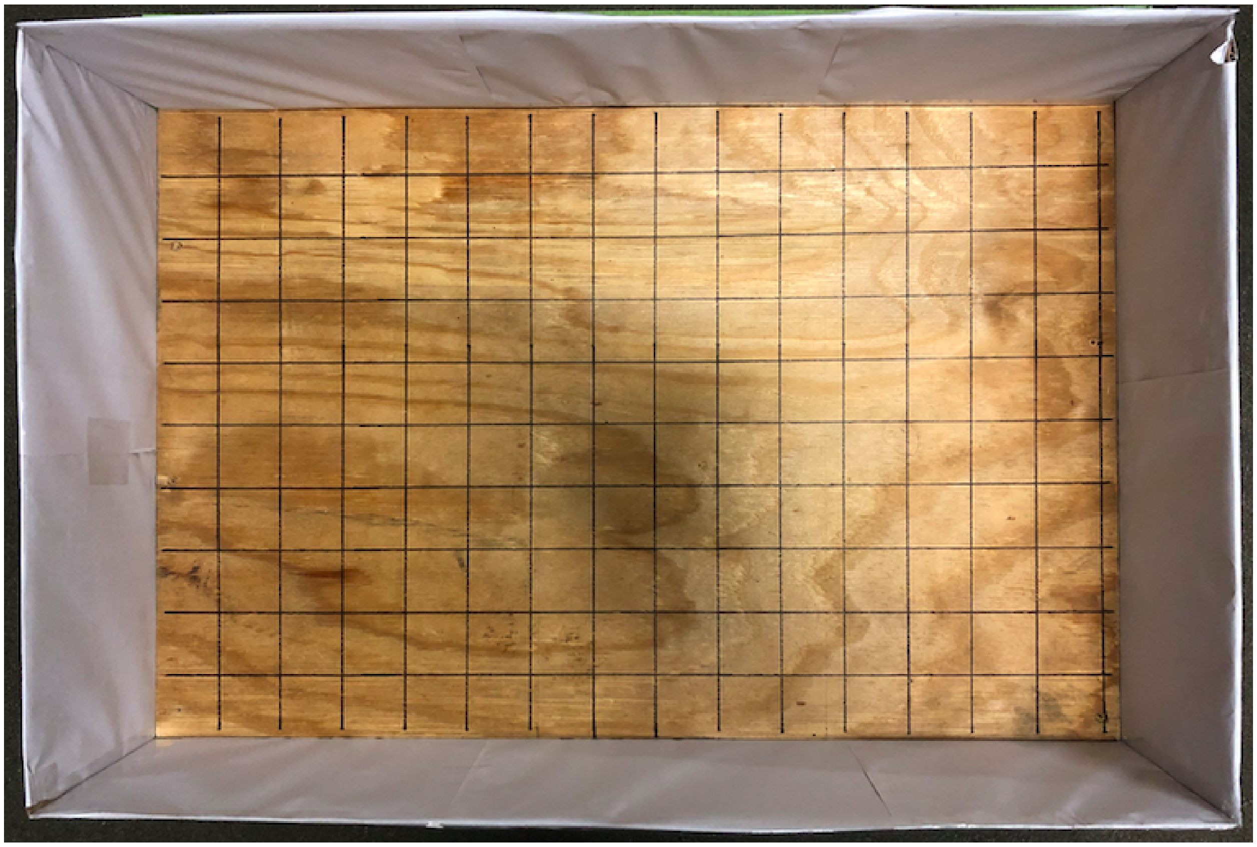

Figure 4: Box with Grids for True State Determination

6.3 Comparison of State Estimates to True States

From the Matlab plot and the birds-eye view video (please refer to appendix and demonstration for video link and detailed information), we can tell that the path the state estimator predicted and the true path have the same general shape, meaning that the mathematical model we derived is accurate.

However, while we increase the noise and uncertainty in the Kalman Filter Algorithm, as well as the case where the initial state is unknown, the path doesn't quite follow the true state anymore, even though still having the same starting and ending position (state) in the box.

The factor leading to the error in the Kalman Filter Algorithm is the Error Covariance. When the error covariance is measured, and determined to be samll (on the order of 0.1 or less), the Kalman filter performs as it should, correcting the state model as the robot experiences noise and deviates from the path expected by the model. However, as error covariance grows, the Kalman Filter exhibits undefined behavior, and instead of correcting erronous state estimates by the model, it actually causes the model to be less accurate, as the path it depicts does not in any way resemble the actual state.

In cases where the error covariance is small, the closed loop state estimator, with the Kalman

Filter implemented, was more accurate than the open loop state estimator. In cases where the error covariance is large, the closed loop state estimator is less accurate than the open loop state estimator, because, as discussed above, the closed loop state estimator makes the model less accurate.

6.4 Discussion and Future Work

While we discovered that when increasing the noise and uncertainty will lead to a under performed state estimate result, the team discussed that a possible reason leads to this error may be due to the cable connected to the ESP WiFi chip while collecting experimental data.

After few trials of testing, we figured when the cable is connected to the ESP WiFi chip, the car won't follow the desired route, as the tension in the cable is doing work on the car even though we treat it carefully while collecting data. We then verified our assumption with wirelessly controlling the car and it performs just fine as it should be.

Future work and improvement will include writing a code that allows wireless data collection and allow the data collected to be sent and save to MATLAB. This way we get rid of the possible source of error. The team should also further look into what is the reason that causes the state estimator to underperform while increasing the noise and uncertainty in our Kalman Filter Algorithm.

7 Appendix and Demonstration

All class materials and documents can be found on GitHub.
All codes, videos, excel sheets and related documents for Lab 2 can be found on GitHub.

For specific experiment videos and demonstrations please visit the links attached to the topics:

- Video: MATLAB Live Car State Estimates
- Video: True State - Birds-eye view
- MATLAB Code
- Images and Figures
- Experimental Data

8 References

1. Mehta, Ankur. ”Design of Robotic Systems I.” EE183DA. University of California Los Angeles, Los Angeles, California. Lectures.
2. Asada, and Harry. ”Lecture Notes.” MIT Open CourseWare. Massachusetts Institute of Technology, n.d. Web. 31 Jan. 2019.