Project 1: Lane Detection

ENPM673: Perception for Autonomous Robots

Project Report

Guide: Prof. Cornelia Fermuller

Yash Shah
UID: 115710498

Contents

Lane Detection

In this project, MATLAB is used as an Image Processing Tool to detect Lanes on the road. The following techniques are used for lane detection.

- Color Masking
- Canny Edge Detection
- Region of Interest Selection
- Hough Transform Line detection

```
%==========================================================================

% MATLAB code for Project 1 (Perception Class)
% Written by Yash Shah (115710498)
% email ID: ysshah95@umd.edu

%==========================================================================
```

Pre-processing the Image

The first step is to import the video file and initialize the variables to be use din the code. Some variables are also imported from the .mat file to be used in the code.

Initializing the Code

```
clc
close all
clear all

%-------------------------Importing the Video File-------------------------
VideoFile = VideoReader('project_video.mp4');

%-------------------Loading Region of Interest Variables-------------------
load('roi_variables', 'c', 'r');

%-----------------Defining Variables for saving Video File-----------------

Output_Video=VideoWriter('Result_Yash');
Output_Video.FrameRate= 25;
open(Output_Video);
```

Initializing the loop to take frames one by one

First the frame is read and them filtered using a Gaussian Filter.

```
while hasFrame(VideoFile)

    %------------------Reading each frame from Video File------------------
    frame = readFrame(VideoFile);
    figure('Name','Original Image'), imshow(frame);

    frame = imgaussfilt3(frame);
    figure('Name','Filtered Image'), imshow(frame);
```

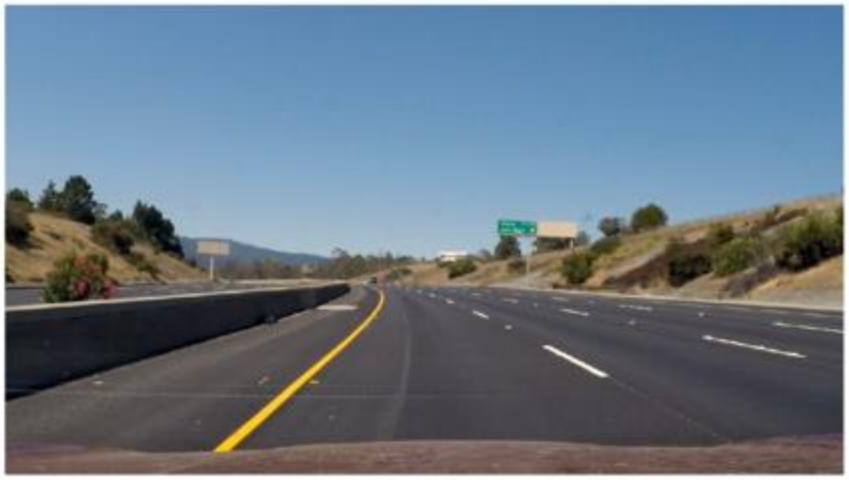

Fig 1: Original Image

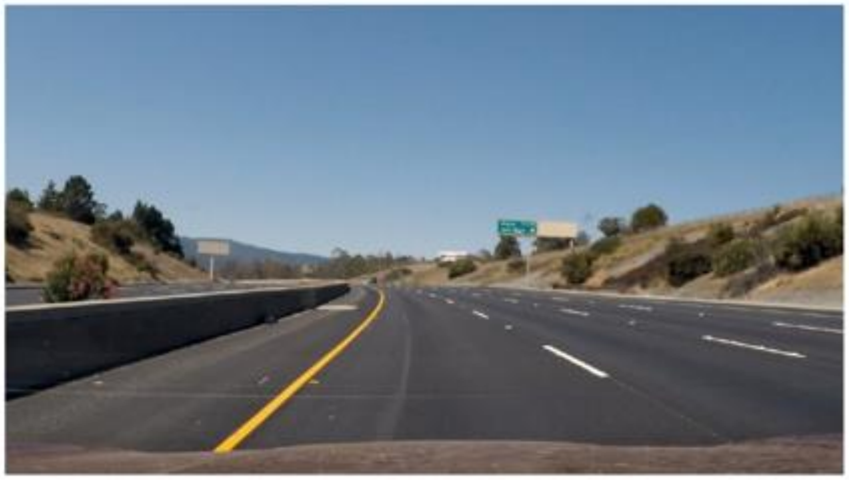

Fig 2: Filtered Image

Masking the image for White and Yellow Color

The frame is masked with yellow and white color to detect the lane lines perfectly.

```
%--------------Define Thresholds for masking Yellow Color--------------

%----------------------Define thresholds for 'Hue'---------------------
channel1MinY = 130;
channel1MaxY = 255;
%------------------Define thresholds for 'Saturation'------------------
channel2MinY = 130;
channel2MaxY = 255;
%---------------------Define thresholds for 'Value'--------------------
channel3MinY = 0;
channel3MaxY = 130;

%-----------Create mask based on chosen histogram thresholds-----------
Yellow=((frame(:,:,1)>=channel1MinY)|(frame(:,:,1)<=channel1MaxY))& ...
    (frame(:,:,2)>=channel2MinY)&(frame(:,:,2)<=channel2MaxY)&...
    (frame(:,:,3)>=channel3MinY)&(frame(:,:,3)<=channel3MaxY);

figure('Name','Yellow Mask'), imshow(Yellow);

%--------------Define Thresholds for masking White Color---------------

%----------------------Define thresholds for 'Hue'---------------------
channel1MinW = 200;
channel1MaxW = 255;
%------------------Define thresholds for 'Saturation'------------------
channel2MinW = 200;
channel2MaxW = 255;
%---------------------Define thresholds for 'Value'--------------------
channel3MinW = 200;
channel3MaxW = 255;

%-----------Create mask based on chosen histogram thresholds-----------
White=((frame(:,:,1)>=channel1MinW)|(frame(:,:,1)<=channel1MaxW))&...
    (frame(:,:,2)>=channel2MinW)&(frame(:,:,2)<=channel2MaxW)& ...
    (frame(:,:,3)>=channel3MinW)&(frame(:,:,3)<=channel3MaxW);

figure('Name','White Mask'), imshow(White);
```

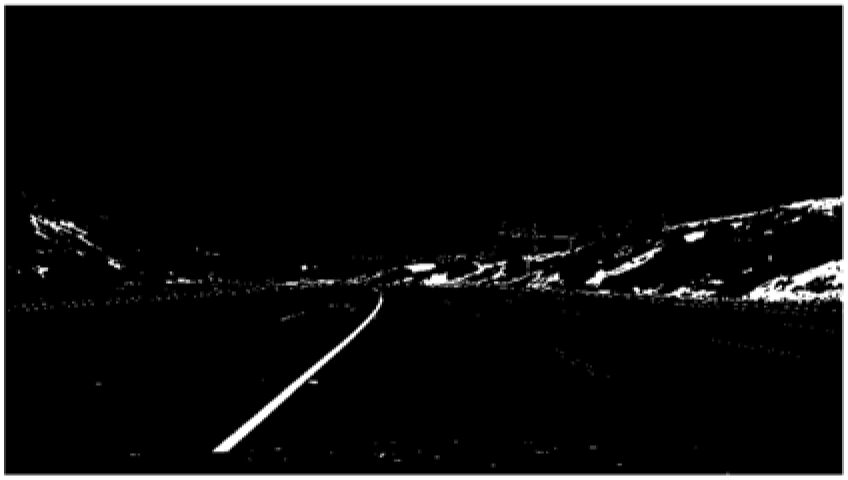

Fig 3: Yellow Mask

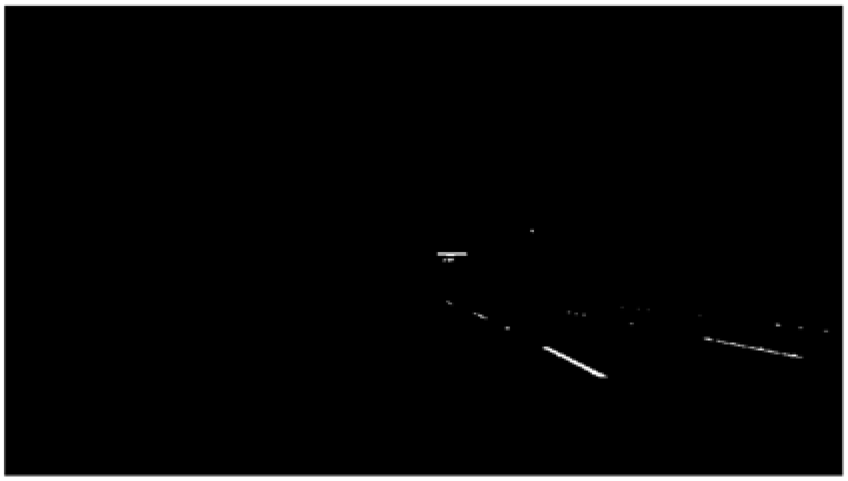

Fig 4: White Mask

Edge Detection

In this section, edges are obtained from the masked image and closed edges with smaller areas are neglected.

```
    frameW = edge(White, 'canny', 0.2);
    frameY = edge(Yellow, 'canny', 0.2);
```

Neglecting closed edges in smaller areas

```
frameY = bwareaopen(frameY,15);
frameW = bwareaopen(frameW,15);

figure('Name','Detecting Edges of Yellow mask'), imshow(frameY);
figure('Name','Detecting Edges of White mask'), imshow(frameW);
```

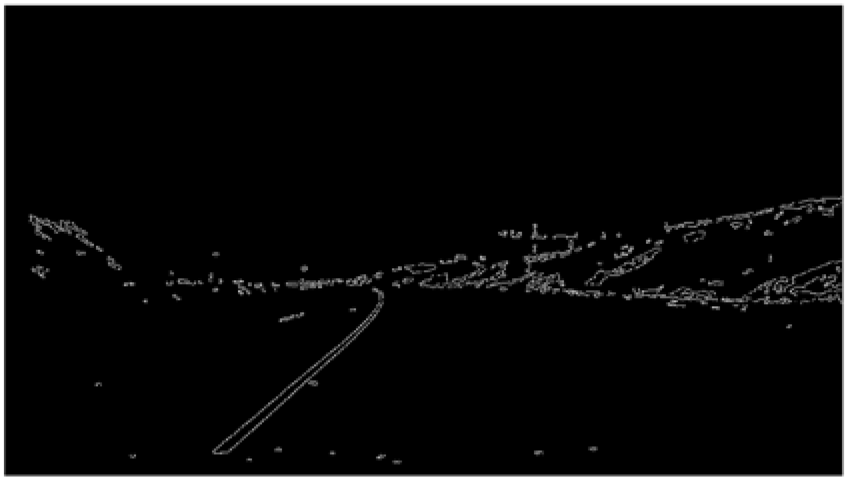

Fig 5: Detecting Edges of Yellow Mask

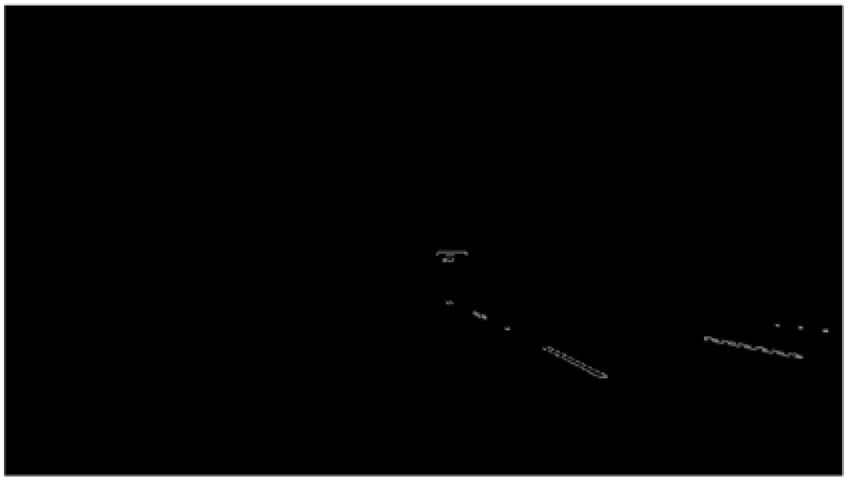

Fig 6: Detecting Edges of White Mask

Extraction of Region of Interest

As guided in the pipeline for the implementation of the project 1 the region of interest is extracted using the 'roipoly' function and selecting the points from the frame.

Deciding ROI Points and Extracting ROI

```
%--------------Deciding ROI points by plotting it on image-------------

% figure(1)
% imshow(frame);
% [r c] = ginput(10);

%---------Extracting Region of Interest from Yellow Edge Frame---------

roiY = roipoly(frameY, r, c);
[R , C] = size(roiY);
for i = 1:R
    for j = 1:C
        if roiY(i,j) == 1
            frame_roiY(i,j) = frameY(i,j);
        else
            frame_roiY(i,j) = 0;
        end
    end
end

figure('Name','Filtering ROI from Yellow mask'), imshow(frame_roiY);

%---------Extracting Region of Interest from White Edge Frame----------

roiW = roipoly(frameW, r, c);
[R , C] = size(roiW);
for i = 1:R
    for j = 1:C
        if roiW(i,j) == 1
            frame_roiW(i,j) = frameW(i,j);
        else
            frame_roiW(i,j) = 0;
        end
    end
end

figure('Name','Filtering ROI from White mask'), imshow(frame_roiW);
```

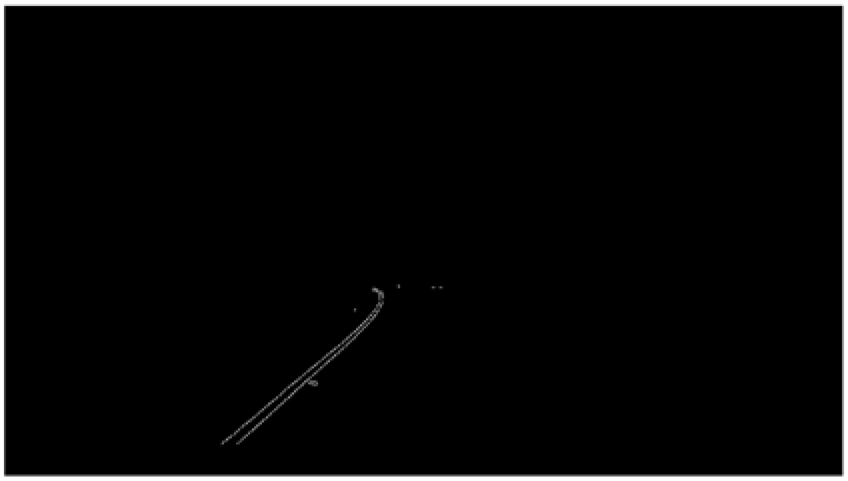

Fig 7: Filtering ROI from Yellow Mask

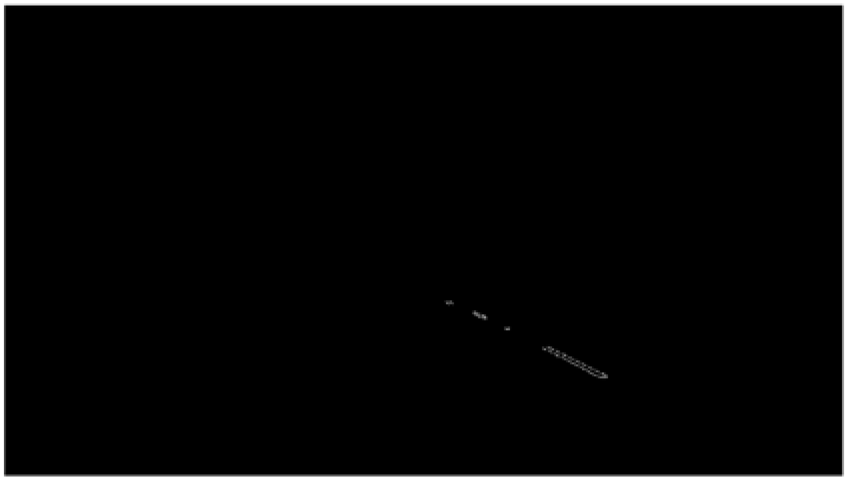

Fig 8: Filtering ROI from White Mask

Hough Transform

In this section I have used the hough function to get the hough transfrom of the binary edge detected image, which gives us the hough values and then I have plotted the hough plot as shown in the figure below.

Applying Hough Tansform to get straight lines from Image

```
%----------Applying Hough Transform to White and Yellow Frames---------


    [H_Y,theta_Y,rho_Y] = hough(frame_roiY);
    [H_W,theta_W,rho_W] = hough(frame_roiW);

    %--------Extracting Hough Peaks from Hough Transform of frames---------

    P_Y = houghpeaks(H_Y,2,'threshold',2);
    P_W = houghpeaks(H_W,2,'threshold',2);

    %----------Plotting Hough Transform and detecting Hough Peaks----------

    figure('Name','Hough Peaks for White Line')

imshow(imadjust(rescale(H_W)),[],'XData',theta_W,'YData',rho_W,'InitialMagnification',
'fit');
    xlabel('\theta (degrees)')
    ylabel('\rho')
    axis on
    axis normal
    hold on
    colormap(gca,hot)

    x = theta_W(P_W(:,2));
    y = rho_W(P_W(:,1));
    plot(x,y,'s','color','blue');
    hold off


    figure('Name','Hough Peaks for Yellow Line')

imshow(imadjust(rescale(H_Y)),[],'XData',theta_Y,'YData',rho_Y,'InitialMagnification',
'fit');
    xlabel('\theta (degrees)')
    ylabel('\rho')
    axis on
    axis normal
    hold on
    colormap(gca,hot)

    x = theta_W(P_Y(:,2));
    y = rho_W(P_Y(:,1));
    plot(x,y,'s','color','blue');
    hold off

    %--------------Extracting Lines from Detected Hough Peaks--------------

    lines_Y = houghlines(frame_roiY,theta_Y,rho_Y,P_Y,'FillGap',3000,'MinLength',20);
```

```
figure('Name','Hough Lines found in image'), imshow(frame), hold on
max_len = 0;
for k = 1:length(lines_Y)
   xy = [lines_Y(k).point1; lines_Y(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

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


lines_W = houghlines(frame_roiW,theta_W,rho_W,P_W,'FillGap',3000,'MinLength',20);

max_len = 0;
for k = 1:2
   xy = [lines_W(k).point1; lines_W(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

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

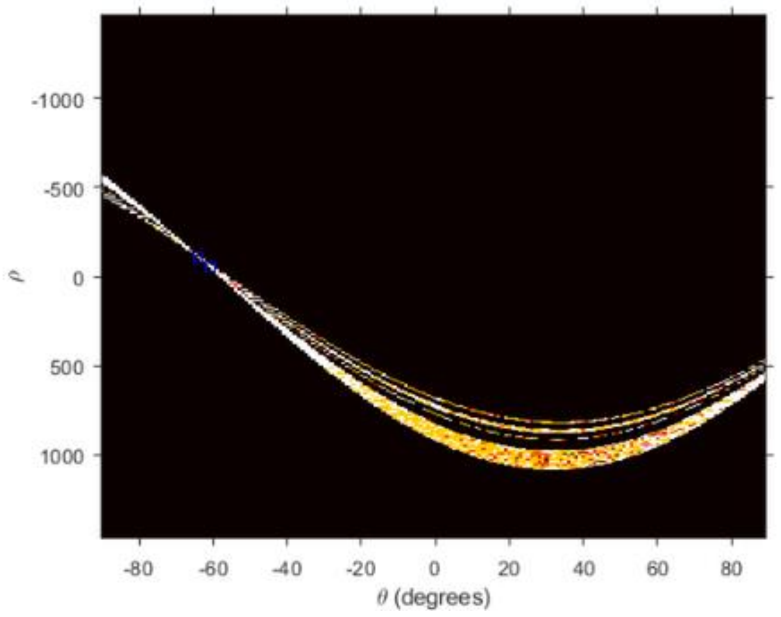

Fig 9: Hough peaks found for yellow lines

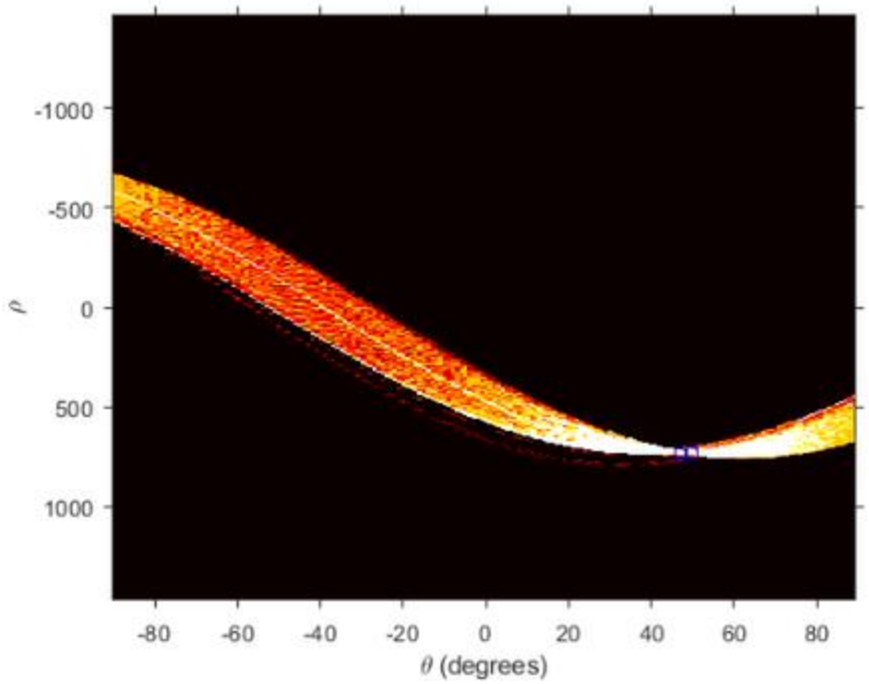

Fig 10: Hough peaks found for white lines

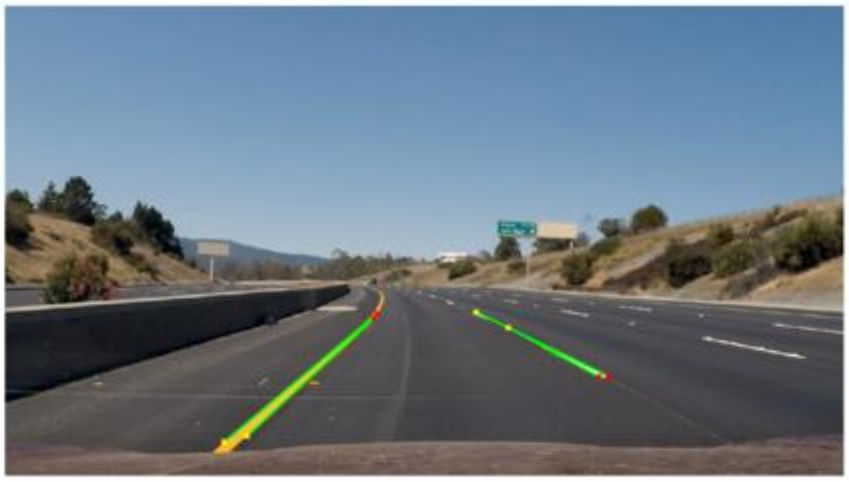

Fig 11: Hough lines found in image

Extrapolation of Lines

In this section the lines that are found in hough lines are extrapolated on the main filtered image.

Plotting best fitting line after Extrapolation

```
%-----------------Extract start and end points of lines----------------

    leftp1 = [lines_Y(1).point1; lines_Y(1).point2];
    leftp2 = [lines_Y(2).point1; lines_Y(2).point2];

    rightp1 = [lines_W(1).point1; lines_W(1).point2];
    rightp2 = [lines_W(2).point1; lines_W(2).point2];

    if leftp1(1,1) < leftp2(1,1)
        left_plot(1,:) = leftp1(1,:);
    else
        left_plot(1,:) = leftp2(1,:);
    end

    if leftp1(2,2) < leftp2(2,2)
        left_plot(2,:) = leftp1(2,:);
    else
        left_plot(2,:) = leftp2(2,:);
    end

    if rightp1(1,2) < rightp2(1,2)
        right_plot(1,:) = rightp1(1,:);
    else
        right_plot(1,:) = rightp2(1,:);
    end

    if rightp1(2,1) > rightp2(2,2)
        right_plot(2,:) = rightp1(2,:);
    else
        right_plot(2,:) = rightp2(2,:);
    end

%     right_plot = rightp1;
%     left_plot = leftp1;

    %----------------Calculate slope of left and right lines---------------

    slopeL = (left_plot(2,2)-left_plot(1,2))/(left_plot(2,1)-left_plot(1,1));
    slopeR = (right_plot(2,2)-right_plot(1,2))/(right_plot(2,1)-right_plot(1,1));

    %------Make equations of left and right lines to extrapolate them------

    xLeftY = 1; % x is on the left edge
    yLeftY = slopeL * (xLeftY - left_plot(1,1)) + left_plot(1,2);
    xRightY = 550; % x is on the reight edge.
    yRightY = slopeL * (xRightY - left_plot(2,1)) + left_plot(2,2);

    xLeftW = 750; % x is on the left edge
    yLeftW = slopeR * (xLeftW - right_plot(1,1)) + right_plot(1,2);
    xRightW = 1300; % x is on the reight edge.
    yRightW = slopeR * (xRightW - right_plot(2,1)) + right_plot(2,2);

    %------Making a transparent Trapezoid between 4 poits of 2 lines-------

    points = [xLeftY yLeftY; xRightY yRightY ;xLeftW yLeftW; xRightW yRightW ];
    number = [1 2 3 4];
```

Turn Prediction

In this section, I have predicted where to turn by looking at the vanishing point found from the extrapolated lines.

```
    %------------------Turn Prediction---------------

    Yellow_dir = cross([left_plot(1,1), left_plot(1,2), 1], [left_plot(2,1),
left_plot(2,2), 1]);
    Yellow_dir = Yellow_dir ./ sqrt(Yellow_dir(1)^2 + Yellow_dir(2)^2);
    theta_y = atan2(Yellow_dir(2), Yellow_dir(1));
    rho_y = Yellow_dir(3);
    yellow_line = [cos(theta_y), sin(theta_y), rho_y];

    %-------------Finding vanishing point using cross poduct---------------
    white_dir = cross([right_plot(1,1),right_plot(1,2),1],
[right_plot(2,1),right_plot(2,2),1]);
    white_dir = white_dir ./ (sqrt(white_dir(1)^2 + white_dir(2)^2));
    theta_w = atan2(white_dir(2),white_dir(1));
    rho_w = white_dir(3);
    white_line = [cos(theta_w), sin(theta_w), rho_w];

    line1 = [0, 1, -left_plot(2,1)];
    point_on_w_lane = cross(line1,white_line);
    point_on_w_lane = point_on_w_lane ./ point_on_w_lane(3);
    line2 = [0, 1, -left_plot(2,2)];
    point_on_w_lane_2 = cross(line2,white_line);
    point_on_w_lane_2 = point_on_w_lane_2 ./ point_on_w_lane_2(3);

    vanishing_point = cross(yellow_line, white_line);
    vanishing_point = vanishing_point ./ vanishing_point(3);
    vanishing_ratio = vanishing_point(1) / size(frame, 2);

    if vanishing_ratio > 0.47 && vanishing_ratio < 0.49
        direction = 'Turn Left';
    elseif vanishing_ratio >= 0.49 && vanishing_ratio <= 0.51
        direction = 'Go Straight';
    else
        direction = 'Turn Right';
    end
```

Plotting Everything on the image

```
%--Plot the extrapolated lines, Trapezoid and direction on each frame--

    figure('Name','Final Output')
    imshow(frame);
    hold on
    plot([xLeftY, xRightY], [yLeftY, yRightY], 'LineWidth',8,'Color','red');
    plot([xLeftW, xRightW], [yLeftW, yRightW], 'LineWidth',8,'Color','red');
    text(650, 65, direction,'horizontalAlignment', 'center',
'Color','red','FontSize',20)
    patch('Faces', number, 'Vertices', points,
'FaceColor','green','Edgecolor','green','FaceAlpha',0.4)
    hold off

    %------------------Save each frame to the Output File------------------
    writeVideo(Output_Video,getframe);
end

%---------------------Closing Save Video File Variable---------------------
close(Output_Video)
```

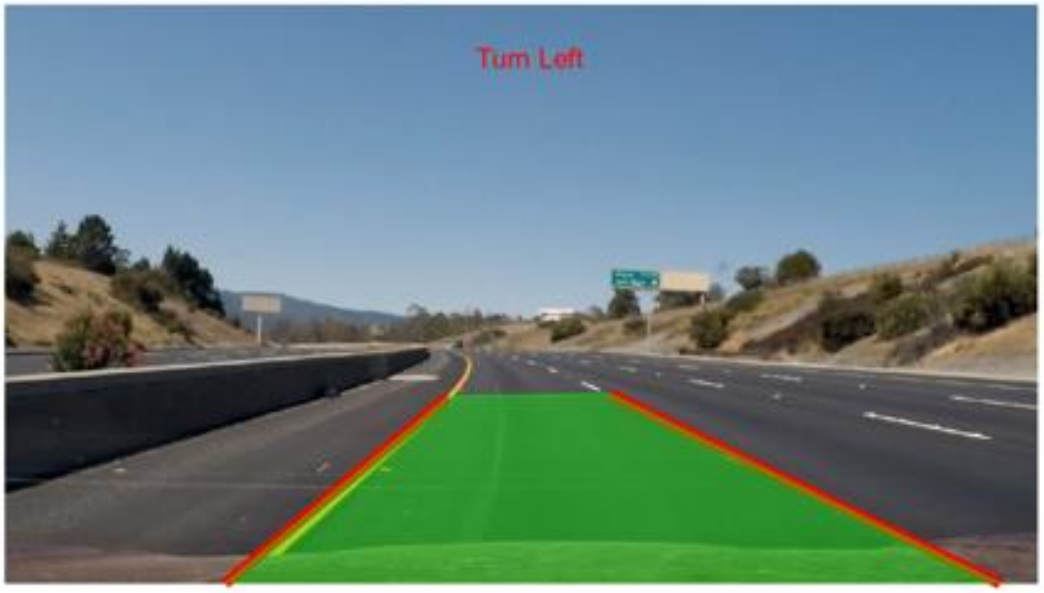

Fig12 : Final Image

Conclusion

The project was successful in that the video images clearly show the lane lines are detected properly and lines are very smoothly handled.

It only detects the straight lane lines. It is an advanced topic to handle curved lanes (or the curvature of lanes). We'll need to use perspective transformation and also poly fitting lane lines rather than fitting to straight lines.