#### LANE-LINE DETECTION SYSTEM IN PYTHON

#### USING OPEN CV

(Mini-Project)

A project report submitted to the Srinivas University as partial fulfilment for the award of the degree of

**Bachelor of Technology in Cloud Technology and Information Security**

Submitted By

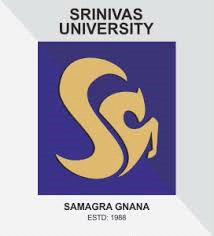
**Ajay Surya S**

**USN: 1SU18DS003**

Under the Guidance of

**Mrs. Renisha P S**

Assistant Professor



**Department of Cloud Technology & Data Science**

**College of Engineering and Technology**

**SRINIVAS UNIVERSITY**

**Mukka, Mangalore – 574146**

**January 2022**

# BONAFIDE CERTIFICATE

#### This is to certify that this project report entitled “LANE-LINE DETECTION SYSTEM IN PYTHON USING OPEN CV” is submitted to Srinivas University College of Engineering and Technology, Mukka, is a bonafide record of work done by **AJAY SURYA S** under my supervision from 1ST of January 2022 to 28th of January 2022

Mrs. Renisha P S

Assistant Professor

Prof. Daniel Selveraj

Head of Department

Department of Cloud Technology and Data Science

Srinivas University, Mukka

Date:

Place: Mukka

# TABLE OF CONTENT

[LIST OF TABLES 4](#_Toc94335751)

[LIST OF FIGURES 5](#_Toc94335752)

[ABBREVIATIONS AND NOMENCLATURE 6](#_Toc94335753)

[ABSTRACT 7](#_Toc94335754)

[1. INRODUCTION 8](#_Toc94335755)

[1.1 THE DOMAIN 8](#_Toc94335756)

[1.2 THE PROBLEM 9](#_Toc94335757)

[1.3 THE TECHNOLOGY 10](#_Toc94335758)

[2. SYSTEM ANALYSIS 11](#_Toc94335759)

[2.1. LITERATURE REVIEW 11](#_Toc94335760)

[2.2 EXISTING SYSTEMS 16](#_Toc94335761)

[2.3. PROPOSED SYSTEM 17](#_Toc94335762)

[2.4. HARDWARE AND SOFTWARE SPECIFICATIONS 18](#_Toc94335763)

[3. SYSTEM DESIGN 20](#_Toc94335764)

[3.1 MODULES DESCRIPTION 20](#_Toc94335765)

[3.2 ARCHITECTURE DIAGRAM 25](#_Toc94335766)

[3.3. USE CASES 27](#_Toc94335767)

[3.3 CLASS DIAGRAM 29](#_Toc94335768)

[3.7 ER DIAGRAM 31](#_Toc94335769)

[3.6 SEQUENCE DIAGRAM 33](#_Toc94335770)

[4. IMPLEMENTATION 34](#_Toc94335771)

[5. TESTING 51](#_Toc94335784)

[6. CONCLUSION AND FUTURE ENHANCEMENTS 55](#_Toc94335785)

[7. REFERENCES 56](#_Toc94335786)

**ABSTRACT**

Being able to detect lane lines could be a crucial task for any self-driving autonomous vehicle. [In](https://projectchampionz.com.ng/tag/in/?amp) this project, to identify lane lines [on](https://projectchampionz.com.ng/tag/on/?amp) the road OpenCV is used. OpenCV method uses the input images to find any lane lines command among and also for rendering out an illustration of the lane. The OpenCV tools like colour selection, the region of interest selection, grey scaling, Gaussian smoothing, Canny Edge [Detection](https://projectchampionz.com.ng/tag/detection/?amp), and Hough Transform line detection are being employed. A colour detection algorithm identifies pixels in a picture that matches a given colour or colour range. Region of interest selection allows you to select a rectangle in an image, crop the rectangular region and finally display the cropped image. Grey scaling is the method of changing an image from different colour spaces e.g. RGB, CMYK, HSV, etc. to shades of grey. In gaussian Blur operation, the image is convolved with a mathematician filter rather than the box filter. The Gaussian filter could be a low-pass filter that removes the high-frequency elements. Canny Edge Detection is used to detect the edges in a picture. It accepts a grayscale image as input and it uses a multi-stage algorithm. The Hough Transform line is a method that is used in image processing to detect any shape if that shape can be represented in mathematical form. The goal is to piece along a pipeline to detect the line segments within the image, then average/extrapolate them and draw them onto the image for the show.

# ****INTRODUCTION****

With the rapid development of society, automobiles have become one of the transportation tools for people to travel. In the narrow road, there are more and more vehicles of all kinds As more and more vehicles are driving on the road, the number of victims of car accidents is increasing every year . How to drive safely under the condition of numerous vehicles and narrow roads has become the focus of attention. Advanced driver assistance systems which include lane departure warning , Lane Keeping Assist, and Adaptive Cruise Control (ACC) can help people analyse the current driving environment and provide appropriate feedback for safe driving or alert the driver in dangerous circumstances. This kind of auxiliary driving system is expected to become more and more perfect . However, the bottleneck of the development of this system is that the ro

ad traffic environment is difficult to predict . After investigation, in the complex traffic environment where vehicles are numerous and speed is too fast, the probability of accidents is much greater than usual. In such a complex traffic situation, road colour extraction and texture detection as well as road boundary and lane marking are the main perceptual clues of human driving

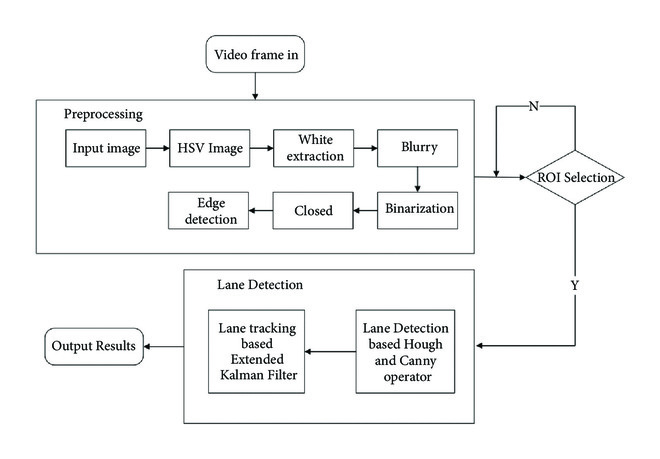
Lane detection is a hot topic in the field of machine learning and computer vision and has been applied in intelligent vehicle systems [8] . The lane detection system comes from lane markers in a complex environment and is used to estimate the vehicle’s [position](https://projectchampionz.com.ng/tag/position/) and trajectory relative to the lane reliably [9] . At the same time, lane detection plays an important role in the lane departure warning system. The lane detection task is mainly divided into two steps: edge detection and line detection.

Line detection is as important as edge detection in lane detection. With regard to line detection, we usually have two methods which include feather-based method and model-based methods. Hence, in this study, we propose a method for  lane-line detection system in python using Opencv.

#### **Overview of the Proposed System**

This paper presents an advanced lane detection technology to improve the efficiency and accuracy of real-time lane detection [16]. The lane detection module is usually divided into two steps: (1) image preprocessing and (2) the establishment and matching of line lane detection model.

Figure [1](https://www.hindawi.com/journals/am/2018/8320207/fig1/) shows the overall diagram of our proposed system where lane detection blocks are the main contributions of this paper. The first step is to read the frames in the video stream. The second step is to enter the image preprocessing module. What is different from others is that in the preprocessing stage we not only process the image itself but also do colour feature extraction and edge feature extraction [17]. In order to reduce the influence of noise in the process of motion and tracking, after extracting the colour features of the image, we need to use Gaussian filter to smooth the image. Then, the image is obtained by binary threshold processing and morphological closure. These are the preprocessing methods mentioned in this paper.



**Figure 1**

**Block diagram of proposed methods.**

**Technology used**

Autonomous Driving Car is one of the most disruptive innovations in AI. Fuelled by Deep Learning algorithms, they are continuously driving our society forward and creating new opportunities in the mobility sector. An autonomous car can go anywhere a traditional car can go and does everything that an experienced human driver does. But it’s very essential to train it properly. One of the many steps involved during the training of an autonomous driving car is lane detection, which is the preliminary step.

**Capturing and decoding video file:**We will capture the video using VideoCapture object and after the capturing has been initialized every video frame is decoded (i.e. converting into a sequence of images).

**Grayscale conversion of image:**The video frames are in RGB format, RGB is converted to grayscale because processing a single channel image is faster than processing a three-channel colored image.

**Reduce noise:**Noise can create false edges, therefore before going further, it’s imperative to perform image smoothening. Gaussian filter is used to perform this process.

**Canny Edge Detector:**It computes gradient in all directions of our blurred image and traces the edges with large changes in intensity. For more explanation please go through this article: [Canny Edge Detector](https://www.geeksforgeeks.org/real-time-edge-detection-using-opencv-python/)

**Region of Interest:**This step is to take into account only the region covered by the road lane. A mask is created here, which is of the same dimension as our road image. Furthermore, bitwise AND operation is performed between each pixel of our canny image and this mask. It ultimately masks the canny image and shows the region of interest traced by the polygonal contour of the mask.

**Hough Line Transform:**The Hough Line Transform is a transform used to detect straight lines. The Probabilistic Hough Line Transform is used here, which gives output as the extremes of the detected lines

Diagram

Description automatically generated

**Existing Systems**

Jae-Hyun Cho et al. (2014)[1] applied the Hough transform with optimized the accumulator cells in the four ROI in parallel and detects lanes with high efficiency. Although Hough Transform can detect only straight lines, the poor lane recognition rate on the curve road has been resolved fairly.

Chan Yee Low et al. (2014) [2] presented a robust road lane marker detection algorithm to detect the left and right lane markers. The algorithm consists of optimization of Canny edge detection and Hough Transform. Canny edge detection performs features recognition then followed by Hough Transform lane generation. Hough Transform is applied to find relevant lines that can be used as the left and right lane boundaries. Reducing the image to smaller region of interest can reduce high computational cost.

Dajun Ding et al. (2013) [3] proposed an algorithm based on road ROI determination for detecting road region using information of vanishing points and line segments. Unnecessary information included in input images was analyzed in a region of interest (ROI) for reducing amount of computation. Hough Transform is used for detecting line segments. Road ROI is determined automatically in every frame. This method works effectively in various road conditions.

HongliFani and Weihua Wang (2013) [4] proposed a new algorithm for color road image edge detection. The original color data in RGB color model were converted to Lab color model and the difference information between the gray image from L channel and the red-green image was obtained with different image method, and the threshold was obtained using optimal threshold value algorithm, then edge detection was carried out. The results show that algorithm has high resistance to noise and retain better edges for color road image edge detection than the traditional algorithms.

N. Phaneendra et al. (2013) [5] adopted lane detection method which consisted of image preprocessing, binary processing and dynamical threshold choosing, and Hough transform model fitting. Instead of Hough transform, Kalman filter was used for improving lane detection performance. Based on distance between lane and center of bottom in captured image coordinate, decision making of lane departure was proposed. Efficiency and feasibility of the solution was indicated by the experimental results.

**1.2 Statement of Problem**

This [thesis](https://projectchampionz.com.ng/tag/thesis/) will investigate different techniques for road and lane detection and how they can be implemented. Our project will be an element of a more extensive project, whose purpose is to create a prototype of an autonomous vehicle. In order to achieve this we need to provide the vehicle with an awareness of the surroundings. To create an artificial computer vision, one tries to imitate functions that the human vision provides. By analyzing digital frequencies in detail from images, the computer can create a visual understanding and act accordingly.

**1.3 Purpose of the Study**

**T**he purpose of this project is to investigate different lane keeping models and implement the most suitable with the input of a RGB camera. The lane keeping algorithm should be able to find curved lanes, be able to communicate with other program’s as well as give an estimated direction angle to make the car position itself between two lines. The goal is to create a narrow AI that’s capable of recreate the function of an human’s vision. The obtained [stream](https://projectchampionz.com.ng/tag/stream/) of frames from the camera is analyzed and processed, with the intention of extracting certain information of the surroundings. The information should be used as an input to the lane detection algorithm, to create an estimated direction angle.

**1.4 Aims and Objective**

The general objective of this study is to develop a lane-line detection system in python using Opencv. The specific objectives include;

1. **Investigate a suitable algorithm for lane keeping based on our initial goals.**
2. **Determine which camera that is suitable for capturing data while moving.**
3. **Determine how should the main neural network and the lane detection algorithm, written in different languages efficiently communicate in real time.**

**1.5 Delimitation of the Study**

The main goal of the project is to investigate existing solutions regarding lane detection and lane keeping and implement an own solution that should be able to detect road lines on slightly curved roads or straight roads, and be able to calculate an estimated direction angle. The lane keeping algorithm will only manage straight and curved lines with good conditions such as marked lanes, beneficial light and limited noise.

When we drive, we use our eyes to decide where to go. The lines on the road that show us where the lanes are act as our constant reference for where to steer the vehicle. Naturally, one of the first things we would like to do in developing a self-driving car is to automatically detect lane lines using an algorithm.

In this project you will detect lane lines in images using Python and OpenCV. OpenCV means "Open-Source Computer Vision", which is a package that has many useful tools for analyzing images.

**The tools we have are color selection, region of interest selection, grayscaling, Gaussian smoothing, Canny Edge Detection and Hough Tranform line detection. Our goal is piece together a pipeline to detect the line segments in the image, then average/extrapolate them and draw them onto the image for display. Once we have a working pipeline, we will try it out on the video stream.**

# Color Selection

First let us select some colors. For Instance: Lane Lines are usually **White** in color and we know the RGB value of White is (255,255,255). Here we will define a color threshold in the variables **red\_threshold**, **green\_threshold**, and **blue\_threshold** and populate **rgb\_threshold** with these values. This vector contains the minimum values for red, green, and blue (R,G,B) that I will allow in my selection.

 import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import numpy as np

*# Read in the image*

image = mpimg.imread('test\_images/solidWhiteRight.jpg')

*# Grab the x and y size and make a copy of the image*

ysize = image.shape[0]

xsize = image.shape[1]

color\_select = np.copy(image)

*# Define color selection criteria*

*###### MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION*

red\_threshold = 200

green\_threshold = 200

blue\_threshold = 200

*######*

rgb\_threshold = [red\_threshold, green\_threshold, blue\_threshold]

*# Do a boolean or with the "|" character to identify*

*# pixels below the thresholds*

thresholds = (image[:,:,0] < rgb\_threshold[0]) \

| (image[:,:,1] < rgb\_threshold[1]) \

| (image[:,:,2] < rgb\_threshold[2])

color\_select[thresholds] = [0,0,0]

*# Display the image*

plt.imshow(image)

plt.title("Input Image")

plt.show()

plt.imshow(color\_select)

plt.title("Color Selected Image")

plt.show()

*# Uncomment the following code if you are running the code locally and wish to save the image*

*# mpimg.imsave("test-after.jpg", color\_select)*

+++++++

++



A picture containing night sky

Description automatically generated

**In the above output we can clearly see the lane lines**

# . Region Masking

I'll assume that the front facing camera that took the image is mounted in a fixed position on the car, such that the lane lines will always appear in the same general region of the image. Next, I'll take advantage of this by adding a criterion to only consider pixels for color selection in the region where we expect to find the lane lines.

Check out the code below. The variables **left\_bottom**, **right\_bottom**, and **apex** represent the vertices of a **triangular region** that I would like to retain for my color selection, while masking everything else out. Here I'm using a triangular mask to illustrate the simplest case, but we can use a quadrilateral, and in principle, we could use any polygon.

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import numpy as np

*# Read in the image*

image = mpimg.imread('test\_images/solidWhiteRight.jpg')

*# Grab the x and y size and make a copy of the image*

ysize = image.shape[0]

xsize = image.shape[1]

color\_select = np.copy(image)

line\_image = np.copy(image)

*# Define color selection criteria*

*# MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION*

red\_threshold = 200

green\_threshold = 200

blue\_threshold = 200

rgb\_threshold = [red\_threshold, green\_threshold, blue\_threshold]

*# Define the vertices of a triangular mask.*

*# Keep in mind the origin (x=0, y=0) is in the upper left*

*# MODIFY THESE VALUES TO ISOLATE THE REGION*

*# WHERE THE LANE LINES ARE IN THE IMAGE*

left\_bottom = [100, 539]

right\_bottom = [950, 539]

apex = [480, 290]

*# Perform a linear fit (y=Ax+B) to each of the three sides of the triangle*

*# np.polyfit returns the coefficients [A, B] of the fit*

fit\_left = np.polyfit((left\_bottom[0], apex[0]), (left\_bottom[1], apex[1]), 1)

fit\_right = np.polyfit((right\_bottom[0], apex[0]), (right\_bottom[1], apex[1]), 1)

fit\_bottom = np.polyfit((left\_bottom[0], right\_bottom[0]), (left\_bottom[1], right\_bottom[1]), 1)

*# Mask pixels below the threshold*

color\_thresholds = (image[:,:,0] < rgb\_threshold[0]) | \

(image[:,:,1] < rgb\_threshold[1]) | \

(image[:,:,2] < rgb\_threshold[2])

*# Find the region inside the lines*

XX, YY = np.meshgrid(np.arange(0, xsize), np.arange(0, ysize))

region\_thresholds = (YY > (XX\*fit\_left[0] + fit\_left[1])) & \

(YY > (XX\*fit\_right[0] + fit\_right[1])) & \

(YY < (XX\*fit\_bottom[0] + fit\_bottom[1]))

*# Mask color and region selection*

color\_select[color\_thresholds | ~region\_thresholds] = [0, 0, 0]

*# Color pixels red where both color and region selections met*

line\_image[~color\_thresholds & region\_thresholds] = [9, 255, 0]

*# Display the image and show region and color selections*

plt.imshow(image)

x = [left\_bottom[0], right\_bottom[0], apex[0], left\_bottom[0]]

y = [left\_bottom[1], right\_bottom[1], apex[1], left\_bottom[1]]

plt.plot(x, y, 'r--', lw=4)

plt.title("Region Of Interest")

plt.show()

plt.imshow(color\_select)

plt.title("Color Selection in the Triangular Region")

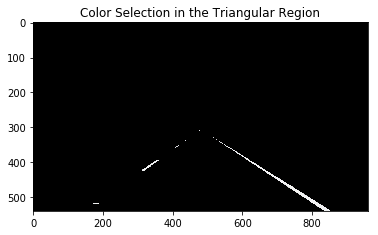
plt.show()

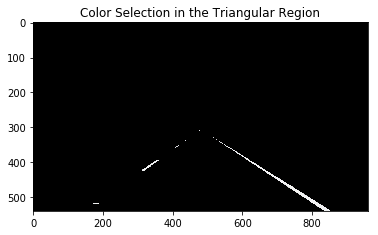
plt.imshow(line\_image)

plt.title("Region Masked Image [Lane Lines in Green]")

plt.show()









# We've successfully detected the Lane Lines [¶](https://www.kaggle.com/code/soumya044/lane-line-detection#Yeah!-We've-successfully-detected-the-Lane-Lines-(Really-?))

Testing

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import numpy as np

*# Read in the image*

image = mpimg.imread('test\_images/solidYellowLeft.jpg')

*# Grab the x and y size and make a copy of the image*

ysize = image.shape[0]

xsize = image.shape[1]

color\_select = np.copy(image)

line\_image = np.copy(image)

*# Define color selection criteria*

*# MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION*

red\_threshold = 200

green\_threshold = 200

blue\_threshold = 200

rgb\_threshold = [red\_threshold, green\_threshold, blue\_threshold]

*# Define the vertices of a triangular mask.*

*# Keep in mind the origin (x=0, y=0) is in the upper left*

*# MODIFY THESE VALUES TO ISOLATE THE REGION*

*# WHERE THE LANE LINES ARE IN THE IMAGE*

left\_bottom = [100, 539]

right\_bottom = [950, 539]

apex = [480, 290]

*# Perform a linear fit (y=Ax+B) to each of the three sides of the triangle*

*# np.polyfit returns the coefficients [A, B] of the fit*

fit\_left = np.polyfit((left\_bottom[0], apex[0]), (left\_bottom[1], apex[1]), 1)

fit\_right = np.polyfit((right\_bottom[0], apex[0]), (right\_bottom[1], apex[1]), 1)

fit\_bottom = np.polyfit((left\_bottom[0], right\_bottom[0]), (left\_bottom[1], right\_bottom[1]), 1)

*# Mask pixels below the threshold*

color\_thresholds = (image[:,:,0] < rgb\_threshold[0]) | \

(image[:,:,1] < rgb\_threshold[1]) | \

(image[:,:,2] < rgb\_threshold[2])

*# Find the region inside the lines*

XX, YY = np.meshgrid(np.arange(0, xsize), np.arange(0, ysize))

region\_thresholds = (YY > (XX\*fit\_left[0] + fit\_left[1])) & \

(YY > (XX\*fit\_right[0] + fit\_right[1])) & \

(YY < (XX\*fit\_bottom[0] + fit\_bottom[1]))

*# Mask color and region selection*

color\_select[color\_thresholds | ~region\_thresholds] = [0, 0, 0]

*# Color pixels red where both color and region selections met*

line\_image[~color\_thresholds & region\_thresholds] = [9, 255, 0]

*# Display the image and show region and color selections*

plt.imshow(image)

x = [left\_bottom[0], right\_bottom[0], apex[0], left\_bottom[0]]

y = [left\_bottom[1], right\_bottom[1], apex[1], left\_bottom[1]]

plt.plot(x, y, 'r--', lw=4)

plt.title("Region Of Interest")

plt.show()

plt.imshow(color\_select)

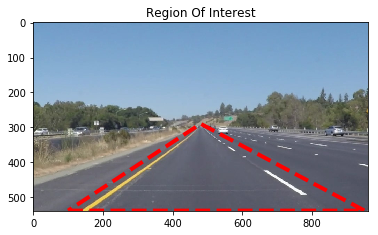
plt.title("Color Selection")

plt.show()

plt.imshow(line\_image)

plt.title("Output Image")

plt.show()



A plane flying in the sky

Description automatically generated with low confidence



In this above image yellow line is undetected

# Canny Edge Detection

Now we are applying **Canny** to the gray-scaled image and our output will be another image called edges. **low\_threshold** and **high\_threshold** are your **thresholds for edge detection**.

The algorithm will first detect strong **edge** (strong gradient) pixels above the **high\_threshold**, and reject pixels below the **low\_threshold**. Next, pixels with values between the **low\_threshold** and **high\_threshold** will be included as long as they are connected to strong edges. **The output edges is a binary image with white pixels tracing out the detected edges and black everywhere else**. See the [OpenCV Canny Docs](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html) for more details.

**What would make sense as a reasonable range for these parameters?** In our case, converting to grayscale has left us with an 8-bit image, so each pixel can take 2^8 = 256 possible values. Hence, the pixel values range from 0 to 255.

This range implies that **derivatives** (essentially, the value differences from pixel to pixel) will be on the scale of tens or hundreds. So, a reasonable range for your threshold parameters would also be in the tens to hundreds.

As far as a ratio of **low\_threshold** to **high\_threshold**, [**John Canny**](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html#steps)**himself recommended a low to high ratio of 1:2 or 1:3.**

We'll also include **Gaussian smoothing**, before running **Canny**, which is **essentially a way of suppressing noise and spurious gradients by averaging (check out the OpenCV docs for GaussianBlur). cv2.Canny() actually applies Gaussian smoothing internally, but we include it here because you can get a different result by applying further smoothing (and it's not a changeable parameter within cv2.Canny()!).\*\***

You can choose the **kernel\_size** for Gaussian smoothing to be any odd number. **A larger kernel\_size implies averaging, or smoothing, over a larger area.**

*# Do all the relevant imports*

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import numpy as np

import cv2

*# Read in the image and convert to grayscale*

*# Note: in the previous example we were reading a .jpg*

*# Here we read a .png and convert to 0,255 bytescale*

image = mpimg.imread('test\_images/solidYellowLeft.jpg')

gray = cv2.cvtColor(image,cv2.COLOR\_RGB2GRAY)

*# Define a kernel size for Gaussian smoothing / blurring*

kernel\_size = 5 *# Must be an odd number (3, 5, 7...)*

blur\_gray = cv2.GaussianBlur(gray,(kernel\_size, kernel\_size),0)

*# Define our parameters for Canny and run it*

low\_threshold = 180

high\_threshold = 240

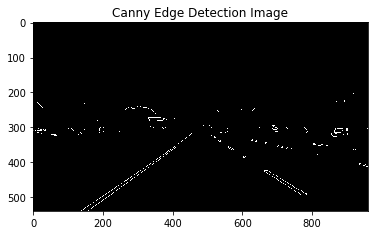
edges = cv2.Canny(blur\_gray, low\_threshold, high\_threshold)

*# Display the image*

plt.imshow(edges, cmap='Greys\_r')

plt.title("Canny Edge Detection Image")

plt.show()



# Hough Transform and detecting Lane Lines

## **Hough Transform**

Hough Transform [7] is a technique used for extracting features that can be used in image anlaysis and digital image processing.Traditional Hough Transform is basically used for identifying lines in the images. There was a difficulty in detecting straight lines, circles etc. in automated analysis of digital images. The edge detector has been used in pre-processing stage for obtaining points on image that lie on desired curve but due to some problem in image, some of the pixels were missing on desired curve. So for solving this problem Hough Transform is used. Hough Transform is an efficient [8] tool for the detection of straight lines in images, even in the presence of noise and occlusion. By counting unique equation for every possible line through point of image, it is able to find dominant lines in an image. By selecting pixels form image object set, the edge pixels can be grouped into an object class. For the detection of lines in an image, it is first converted into a binary image using some threshold. Then the dataset is added with suitable instances. Hough space is the main part of Hough Transform. In a Hough Space each point (d, T) is matched to a line at angle T and distance d from origin. The point along a line is given by the value of a function in Hough space. For each point, consider all line which goes through that point at discrete set of angles based on priority basis. An array called accumulator is used to detect lines in Hough transform. The dimension of the accumulator is equal to number of unknown Hough transform parameters. Initially, lines are generated that can pass through each point. In case of an intersection of a line with other lines of other points, the vote for those (d,T) parameters is incremented. Finally the pair of (d,T) parameters with the highest vote is selected as predominant line present on the image plane based on the points that compose this line The Hough Transform is basically used for the detection of straight lanes. But it can be improved to detect the curved lanes effectively and efficiently. This improvement is not given much focus till now.

*# Read in and grayscale the image*

image = mpimg.imread('test\_images/solidYellowLeft.jpg')

gray = cv2.cvtColor(image,cv2.COLOR\_RGB2GRAY)

*# Define a kernel size and apply Gaussian smoothing*

kernel\_size = 5

blur\_gray = cv2.GaussianBlur(gray,(kernel\_size, kernel\_size),0)

*# Define our parameters for Canny and apply*

low\_threshold = 180

high\_threshold = 240

edges = cv2.Canny(blur\_gray, low\_threshold, high\_threshold)

*# Next we'll create a masked edges image using cv2.fillPoly()*

mask = np.zeros\_like(edges)

ignore\_mask\_color = 255

*# This time we are defining a four sided polygon to mask*

imshape = image.shape

vertices = np.array([[(0,imshape[0]),(450, 290), (490, 290), (imshape[1],imshape[0])]], dtype=np.int32)

cv2.fillPoly(mask, vertices, ignore\_mask\_color)

masked\_edges = cv2.bitwise\_and(edges, mask)

*# Define the Hough transform parameters*

*# Make a blank the same size as our image to draw on*

rho = 1 *# distance resolution in pixels of the Hough grid*

theta = np.pi/180 *# angular resolution in radians of the Hough grid*

threshold = 2 *# minimum number of votes (intersections in Hough grid cell)*

min\_line\_length = 4 *#minimum number of pixels making up a line*

max\_line\_gap = 5 *# maximum gap in pixels between connectable line segments*

line\_image = np.copy(image)\*0 *# creating a blank to draw lines on*

*# Run Hough on edge detected image*

*# Output "lines" is an array containing endpoints of detected line segments*

lines = cv2.HoughLinesP(masked\_edges, rho, theta, threshold, np.array([]),

min\_line\_length, max\_line\_gap)

*# Iterate over the output "lines" and draw lines on a blank image*

for line **in** lines:

for x1,y1,x2,y2 **in** line:

cv2.line(line\_image,(x1,y1),(x2,y2),(255,0,0),10)

*# Create a "color" binary image to combine with line image*

color\_edges = np.dstack((edges, edges, edges))

*# Draw the lines on the edge image*

lines\_edges = cv2.addWeighted(color\_edges, 0.8, line\_image, 1, 0)

lines\_edges = cv2.polylines(lines\_edges,vertices, True, (0,0,255), 10)

plt.imshow(image)

plt.title("Input Image")

plt.show()

plt.imshow(lines\_edges)

plt.title("Colored Lane line [In RED] and Region of Interest [In Blue]")

plt.show()

,

,

**Conclusion And Future Enhancements**

In this paper, we proposed a new lane detection preprocessing and ROI selection methods to design a lane detection system. The main idea is to add white extraction before the conventional basic preprocessing. Edge extraction has also been added during the preprocessing stage to improve lane detection accuracy. We also placed the ROI selection after the proposed preprocessing. Compared with selecting the ROI in the original image, it reduced the nonlane parameters and improved the accuracy of lane detection. Currently, we only use the Hough transform to detect straight lane and EKF to track lane and do not develop advanced lane detection methods. In the future, we will exploit a more advanced lane detection approach to improve the performance.

**REFERENCES**

1. Cho, Jae-Hyun, Young-Min Jang, and Sang-Bock Cho. "Lane recognition algorithm using the Hough transform with applied accumulator cells in multichannel ROI." In Consumer Electronics (ISCE 2014), The 18th IEEE International Symposium on, pp. 1- 3. IEEE, 2014.

2. Low, Chan Yee, HairiZamzuri, and SaifulAmriMazlan. "Simple Robust Road Lane Detection Algorithm”. IEEE, 2014

. 3. Ding, Dajun, Chanho Lee, and Kwang-yeob Lee. "An adaptive road ROI determination algorithm for lane detection." In TENCON 2013-2013 IEEE Region 10 Conference (31194), pp. 1-4. IEEE, 2013.

4. Fani, Hongli, and Weihua Wang. "Edge Detection of Color Road Image Based on Lab Model." In Computational and Information Sciences (ICCIS), 2013 Fifth International Conference on, pp. 298- 301. IEEE, 2013.

5. Nalla, Phaneendra, GCL AbhiramanGoud, and V. Padmaja. "ACCIDENT AVOIDING SYSTEM USING LANE DETECTION." IJRECE 1, no. 1 (2013): 01-04

. 6. Jian, Wang, Sun Sisi, Gong Jingchao, and Cao Yu. "Research of lane detection and recognition technology based on morphology feature." In Control and Decision Conference (CCDC), 2013 25th Chinese, pp. 3827-3831. IEEE, 2013.

7. Mariut, F., C. Fosalau, and D. Petrisor. "Lane mark detection using Hough Transform" In Electrical and Power Engineering (EPE), 2012 International Conference and Exposition on, pp. 871-875. IEEE, 2012.

8. Ghazali, Kamarul, Rui Xiao, and Jie Ma. "Road lane detection using H-maxima and improved hough transform." In Computational Intelligence, Modelling and Simulation (CIMSiM), 2012 Fourth International Conference on, pp. 205-208. IEEE, 2012.

9. Chen, Yong, and Mingyi He. "Sharp curve lane boundaries projective model and detection." In Industrial Informatics (INDIN), 2012 10th IEEE International Conference on, pp. 1188-1193. IEEE, 2012.

10. Xu, Zhiyuan, Xiaoming Liu, and Na Ji. "Fog removal from color images using contrast limited adaptive histogram equalization." In Image and Signal Processing, 2009. CISP'09. 2nd International Congress on, pp. 1-5. IEEE, 2009

. 11. Rani, Versha. "A brief study of various noise model and filtering techniques."Journal of global research in computer science 4, no. 4 (2013): 166-171.

12. Lee, Seonyoung, Haengseon Son, and Kyungwon Min. "Implementation of lane detection system using optimized hough transform circuit." In Circuits and Systems (APCCAS), 2010 IEEE Asia Pacific Conference on, pp. 406-409. IEEE, 2010.

13. Saha, Anik, Dipanjan Das Roy, TauhidulAlam, and Kaushik Deb. "Automated Road Lane Detection for Intelligent Vehicles." Global Journal of Computer Science and Technology 12, no. 6 (2012)

. 14. Prakash, J., M. B. Meenavathi, and K. Rajesh. "Linear feature extraction using combined approach of Hough transform, Eigen values and Raster scan algorithms." In Intelligent Sensing and Information Processing, 2006. ICISIP 2006. Fourth International Conference on, pp. 65-70. IEEE, 2006.