ROAD DETECTION FROM A PICTURE USING COMPUTER VISION

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

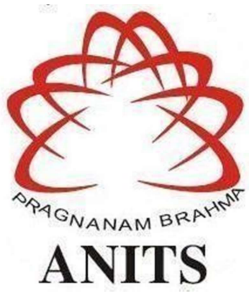
IN

INFORMATION TECHNOLOGY

Submitted by

B.KAVYA	317126511063
N.SAI LAVANYA	317126511096
P.SRAVANTHI	317126511097
P.ISHITHA	317126511099

Under the guidance of
Mrs. M.SWATHI
(Assistant Professor)



DEPARTMENT OF INFORMATION TECHNOLOGY

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC AUTONOMOUS)

(***Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade***)

Sangivalasa, Bheemili mandal, Visakhapatnam - 531162(A.P)

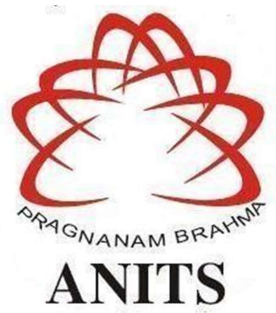
(2017-2021)

DEPARTMENT OF INFORMATION TECHNOLOGY

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES (UGC AUTONOMOUS)

(***Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade***)

Sangivalasa, Bheemili mandal, Visakhapatnam dist. (A.P)



BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**ROAD DETECTION FROM A PICTURE USING COMPUTER VISION"** submitted by B.Kavya(317126511063),N.Sai Lavanya(317126511096),P.Sravanthi (317126511097),P.Ishitha(317126511099) in fulfillment of Final Year Project in IV/IV B.Tech, in Information Technology at ANITS, Visakhapatnam during the year 2020 -2021. It is certified that all corrections indicated for internal assessment have been incorporated to account.

Mrs.M.Swathi
ASSISTANT PROFESSOR
Department of IT
ANITS

Dr.Poosapati Padmaja
HEAD OF THE DEPARTMENT
Department of IT
ANITS

DECLARATION

This is to certify that the project work entitled "**ROAD DETECTION FROM A PICTURE USING COMPUTER VISION**" is a bonafide work carried out by **B.Kavya**, **N.Sai Lavanya, P.Sravanthi, P.Ishitha** as a part of **B.TECH** final year 2nd semester of **Information Technology** of Andhra University, Visakhapatnam during the year 2017-2021.

We, **B.Kavya**, **N.Sai Lavanya, P.Sravanthi, P.Ishitha** students of final semester B.Tech**,** Information Technology from ANITS, Visakhapatnam, hereby declare that the project entitled "**ROAD DETECTION FROM A PICTURE USING COMPUTER VISION**" is carried out by us and submitted in fulfilment of the requirements for the award of **Bachelor of Technology in Information Technology**, under Anil Neerukonda Institute of Technology and Sciences during the academic year 2017-2021 and has not been submitted to any other university for the award of any kind of degree.

B.KAVYA	317126511063
N.SAI LAVANYA	317126511096
P.SRAVANTHI	317126511097
P.ISHITHA	317126511099

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mrs.M.Swathi,** Assistant Professor, Department of Information Technology, ANITS, for his/her guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr.Poosapati Padmaja**, Head of the Department, Information Technology, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa,**for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of IT, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of IT, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last, but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

B.KAVYA	317126511063
N.SAI LAVANYA	317126511096
P.SRAVANTHI	317126511097
P.ISHITHA	317126511099

TABLE OF CONTENTS

ABSTRACT

Given a picture captured from a camera hooked up to a vehicle moving on a road during which captured road could or might not be levelled, or have clearly described edges, or some previous acknowledged patterns thereon, then road detection from one image will be applied to search out the road in a picture so it might be used as a district in automation of driving system within the vehicles for moving the vehicle in correct road. during this method of finding the road within the image captured by the vehicle, we are able to use some algorithms for vanishing point detection, exploitation Hough Transformation Space, finding the region of interest, edge detection exploitation Canny edge detection for road detection. We have a tendency to use thousands of pictures of various roads to coach our model so the model might notice the road as a result within the new image processed through the vehicle. Keywords Patterns thereon, automation of driving system, vanishing point detection, exploitation, Hough Transformation Space, Region of Interest, Canny Edge Detection.

Keywords

Patterns thereon, automation of driving system, vanishing point detection, exploitation, Hough Transformation Space, Region of Interest, Canny Edge Detection.

LIST OF FIGURES

1. INTRODUCTION

1.1 DIGITAL IMAGE PROCESSING

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing is a method to perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it.

If we talk about the basic definition of image processing then "Image processing is the analysis and manipulation of a digitized image, especially in order to improve its quality".

Digital-Image: An image may be defined as a two-dimensional function f(x, y), where x and y are spatial(plane) coordinates, and the amplitude of fat any pair of coordinates (x, y) is called the intensity or grey level of the image at that point.

In another word An image is nothing more than a two-dimensional matrix (3-D in case of coloured images) which is defined by the mathematical function f(x, y) at any point is giving the pixel value at that point of an image, the pixel value describes how bright that pixel is, and what colour it should be.

Image processing is basically signal processing in which input is an image and output is image or characteristics according to requirement associated with that image.

1.1.1 Steps in Image Processing:

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools;
- Analysing and manipulating the image;
- Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information extraction.

1.1.2 What is an Image?

An image is defined as a two-dimensional function, F (x, y), where x and y are spatial coordinates, and the amplitude of F at any pair of coordinates (x,y) is called the intensity of that image at that point. When x,y, and amplitude values of F are finite, we call it a digital image. In other words, an image can be defined by a two-dimensional array specifically arranged in rows and columns. Digital Image is composed of a finite number of elements, each of which elements have a particular value at a particular location. These elements are referred to as picture elements, image elements, and pixels. A Pixel is most widely used to denote the elements of a Digital Image.

1.1.3 Image in Matrix Representation

As we know, images are represented in rows and columns we have the following syntax in which images are represented:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ \vdots & \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

The right side of this equation is digital image by definition. Every element of this matrix is called image element, picture element, or pixel.

1.1.4 Types of an Image

Binary Image– The binary image as its name suggests, contain only two pixel elements i.e. 0 & 1, where 0 refers to black and 1 refers to white. This image is also known as Monochrome.

Black and White Image– The image which consist of only black and white colour is called Black and White Image.

8 Bit Colour Format– It is the most famous image format. It has 256 different shades of colours in it and commonly known as Grayscale Image. In this format, 0 stands for Black, and 255 stands for white, and 127 stands for grey.

16 Bit Colour Format– It is a colour image format. It has 65,536 different colours in it. It is also known as High Colour Format. In this format the distribution of color is not as same as Grayscale image.

A 16-bit format is actually divided into three further formats which are Red, Green and Blue. That famous RGB format.

1.2 SELF DRIVING CARS

A self-driving car (sometimes called an autonomous car or driverless car) is a vehicle that uses a combination of sensors, cameras, radar and artificial intelligence (AI) to travel between destinations without a human operator. To qualify as fully autonomous, a vehicle must be able to navigate without human intervention to a predetermined destination over roads that have not been adapted for its use.

Companies developing and/or testing autonomous cars include Audi, BMW, Ford, Google, General Motors, Tesla, Volkswagen and Volvo. Google's test involved a fleet of self-driving cars -- including Toyota Prii and an Audi TT -- navigating over 140,000 miles of California streets and highways.

Levels of autonomy in self-driving cars

The U.S. National Highway Traffic Safety Administration (NHTSA) lays out six levels of automation, beginning with zero, where humans do the driving, through driver assistance technologies up to fully autonomous cars. Here are the five levels that follow zero automation:

Level 1: Advanced driver assistance system (ADAS) aid the human driver with either steering, braking or accelerating, though not simultaneously. ADAS includes rear view cameras and features like a vibrating seat warning to alert drivers when they drift out of the traveling lane.

Level 2: An ADAS that can steer and either brake or accelerate simultaneously while the driver remains fully aware behind the wheel and continues to act as the driver.

Level 3: An automated driving system (ADS) can perform all driving tasks under certain circumstances, such as parking the car. In these circumstances, the human driver must be ready to re-take control and is still required to be the main driver of the vehicle.

Level 4: An ADS is able to perform all driving tasks and monitor the driving environment in certain circumstances. In those circumstances, the ADS is reliable enough that the human driver needn't pay attention.

Level 5: The vehicle's ADS acts as a virtual chauffeur and does all the driving in all circumstances. The human occupants are passengers and are never expected to drive the vehicle.

1.3 WORKING OF SELF DRIVING CARS

Lane lines are being drawn as the car drives. Also, you can see the radius of curvature is being calculated to help the car steer. It is cheap to equip cars with a front facing camera. Much cheaper than RADAR or LIDAR. Once we get a camera image from the front facing camera of self-driving car, we make several modifications to it. The steps I followed are detailed below:

1.3.1 Distortion correction

Image distortion occurs when a camera looks at 3D objects in the real world and transforms them into a 2D image. This transformation isn't always perfect and distortion can result in a change in apparent size, shape or position of an object. So we need to correct this distortion to give the camera an accurate view of the image. This is done by computing a camera calibration matrix by taking several chessboard pictures of a camera.

See example below of a distortion corrected image. Please note that the correction is very small in normal lenses and the difference isn't visible much to the naked eye.

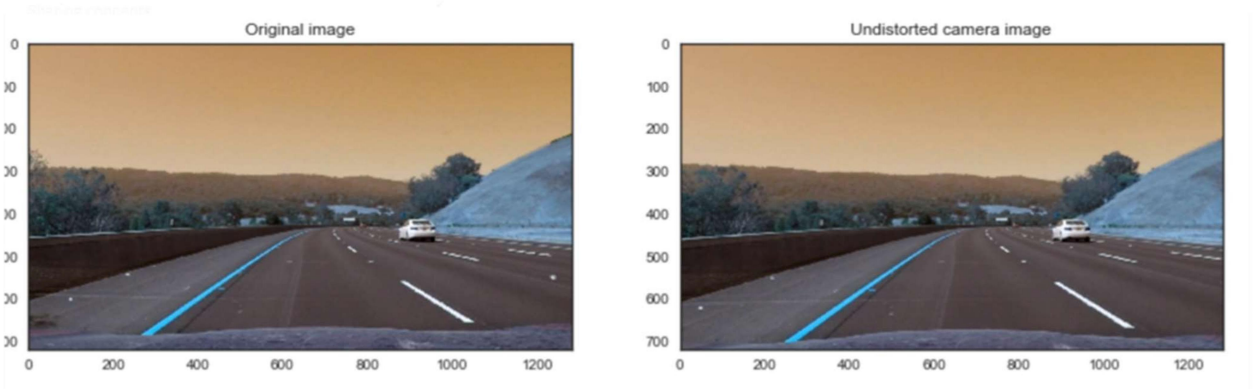


Figure 1.a**:** Describing the original image and undistorted image

1.3.2 Create a binary image

Now that we have the undistorted image, we can start our analysis. We need to explore different schemes so that we can clearly detect the object of interest on the road, in this case lane lines while ignoring the rest. I did this in two ways:

- **Using Sobel operator to compute x-gradient**

The gradient of an image can be used to identify sharp changes in colour in a black and white image. It is a very useful technique to detect edges in an image. For

the image of a road, we usually have a lane line in either yellow or white on a black road and so x-gradient can be very useful.

- **Explore other colour channels**

  HSV (Hue, Saturation and Value) colour space can be very useful in isolating the yellow and line white lines because it isolates colour (hue), amount of colour (saturation) and brightness (value). We can use the S colour channel in the image.

- **Birds Eye View Image**

  After the thresholding operation, we perform a perspective transform to change the image to bird's eye view. This is done because from this top view we can identify the curvature of the lane and decide how to steer the car. To perform the perspective transform, I identified 4 source points that form a trapezoid on the image and 4 destination points such that lane lines are parallel to each other after the transformation. The destination points were chosen by trial and error but once chosen works well for all images and the video since the camera is mounted in a fixed position. *OpenCV* can be used to perform this. See how clearly the curvature of the lane lines is visible in this view.

- **Fit curve lines to the bird eye view image**

  In order to better estimate where the lane is, we use a histogram of the bottom half of image to identify potential left and right lane markings. Modification of this function to narrow down the area in which left and right lanes can exist so that highway lane separators or any other noise doesn't get identified as a lane. Once the initial left and right lane bottom points are identified.

- Plot the result identified by the system clearly. This plotting can be done filling the space area with transparent colour using OpenCV.

Thus, Self-Driving car works and road detection can be useful in detection of road from an image captured from car.

1.4 MOTIVATION FOR THE WORK

In the past five years, autonomous driving has gone from “maybe possible” to “definitely possible” to “inevitable” to “how did anyone ever think this wasn’t inevitable?” to "now commercially available." In December 2018, Waymo, the company that emerged from Google’s self-driving-car project, officially started its commercial self-driving-car service in the suburbs of Phoenix. The details of the program—it's available only to a few hundred vetted riders, and human safety operators will remain behind the wheel—may be underwhelming but don't erase its significance. People are now paying for robot rides. And it's just a start. Waymo will expand the service's capability and availability over time. Meanwhile, its onetime monopoly has evaporated. Smaller start-ups like May Mobility and Drive.ai are running small-scale but revenue-generating shuttle services. Every significant automaker is pursuing the tech, eager to rebrand and rebuild itself as a “mobility provider” before the idea of car ownership goes kaput. Ride-hailing companies like Lyft and Uber are hustling to dismiss the profit-gobbling human drivers who now shuttle their users about. Tech giants like Apple, IBM, and Intel are looking to carve off their slice of the pie. Countless hungry start-ups have materialized to fill niches in a burgeoning ecosystem, focusing on laser sensors, compressing mapping data, setting up service centres, and more. This 21st-century gold rush is motivated by the intertwined forces of opportunity and survival instinct. By one account, driverless tech will add $7 trillion to the global economy and save hundreds of thousands of lives in the next few decades. Simultaneously, it could devastate the auto industry and its associated gas stations, drive-thrust, taxi drivers, and truckers. Some people will prosper. Most will benefit. Many will be left behind. It’s worth remembering that when automobiles first started rumbling down manure-clogged streets, people called them horseless carriages. The moniker made sense: Here were vehicles that did what carriages did, minus the hooves. By the time “car” caught on as a term, the invention had become something entirely new. Over a century, it reshaped how humanity moves and thus how (and where and with whom) humanity lives. This cycle has restarted, and the term “driverless car” will soon seem as anachronistic as “horseless carriage.” We don’t know how cars that don’t need human chauffeurs will mold society, but we can be sure a similar gear shift is on the way. Just over a decade ago, the idea of being chauffeured around by a string of zeros and ones was ludicrous to pretty much everybody who wasn’t at an abandoned Air Force base

outside Los Angeles, watching a dozen driverless cars glide through real traffic. That event was the Urban Challenge, the third and final competition for autonomous vehicles put on by Darpa, the Pentagon's skunkworks arm. At the time, America's military-industrial complex had already thrown vast sums and years of research trying to make unmanned trucks. It had laid a foundation for this technology, but stalled when it came to making a vehicle that could drive at practical speeds, through all the hazards of the real world. So, Darpa figured, maybe someone else—someone outside the DOD's standard roster of contractors, someone not tied to a list of detailed requirements but striving for a slightly crazy goal—could put it all together. It invited the whole world to build a vehicle that could drive across California's Mojave Desert, and whoever's robot did it the fastest would get a million-dollar prize. The most successful vehicle went just seven miles. Most crashed, flipped, or rolled over within sight of the starting gate. But the race created a community of people—geeks, dreamers, and lots of students not yet jaded by commercial enterprise—who believed the robot drivers people had been craving for nearly forever were possible, and who were suddenly driven to make them real.

They came back for a follow-up race in 2005 and proved that making a car drive itself was indeed possible: Five vehicles finished the course. By the 2007 Urban Challenge, the vehicles were not just avoiding obstacles and sticking to trails but following traffic laws, merging, parking, even making safe, legal U-turns.

When Google launched its self-driving car project in 2009, it started by hiring a team of Darpa Challenge veterans. Within 18 months, they had built a system that could handle some of California's toughest roads (including the famously winding block of San Francisco's Lombard Street) with minimal human involvement. A few years later, Elon Musk announced Tesla would build a self-driving system into its cars. And the proliferation of ride-hailing services like Uber and Lyft weakened the link between being in a car and owning that car, helping set the stage for a day when actually driving that car falls away too. In 2015, Uber poached dozens of scientists from Carnegie Mellon University—a robotics and artificial intelligence powerhouse—to get its effort going.

1.5 PROBLEM STATEMENT

Given an image captured from a camera attached to a vehicle moving on a road in which captured road may or may not be well levelled, or have clearly delineated edges, or some prior

known patterns on it, then road detection from a single image can be applied to find the road in an image so that it could be used as a part in automation of driving system in the vehicles for moving the vehicle in correct road. In this process of finding the road in the image captured by the vehicle, we can use some algorithms for vanishing point detection using Hough transform space, finding the region of interest, edge detection using canny edge detection algorithm and then road detection. We use thousands of images of different roads to train our model so that the model could detect the road which is present in the new image processed through the vehicle.

2. LITERATURE SURVEY

2.1 EDGE DETECTION

Edges characterize boundaries and are therefore a problem of fundamental importance in image processing. Edges in images are areas with strong intensity contrasts – a jump in intensity from one pixel to the next. Edge detecting an image significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image.

Canny edge detection algorithm is also known as the optimal edge detector. Cranny's intentions were to enhance the many edge detectors in the image.

- The first criterion should have low error rate and filter out unwanted information while the useful information preserve.
- The second criterion is to keep the lower variation as possible between the original image and the processed image.
- Third criterion removes multiple responses to an edge.

Based on these criteria, the canny edge detector first smoothens the image to eliminate noise. It then finds the image gradient to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and suppresses any pixel that is not at the maximum using non-maximum suppression. The gradient array is now further reduced by hysteresis to remove streaking and thinning the edges.

2.2 FILTER OUT NOISE

2.2.1 Convolution

First step to Canny edge detection require some method of filter out any noise and still preserve the useful image. Convolution is a simple mathematic method to many common image-processing operators.

I1	I2	I3	I4	I5	I6	I7	I8	I9
I10	I11	I12	I13	I14	I15	I16	I17	I18
I19	I20	I21	I22	I23	I24	I25	I26	I27
I28	I29	I30	I31	I32	I33	I34	I35	I36
I37	I38	I39	I40	I41	I42	I43	I44	I45
I46	I47	I48	I49	I50	I51	I52	I53	I54
I55	I56	I57	I58	I59	I60	I61	I62	I63
I64	I65	I66	I67	I68	I69	I70	I71	I72
I73	I74	I75	I76	I77	I78	I79	I80	I81

K1	K2	K3
K4	K5	K6
K7	K8	K9

Figure 2.a: An example small image (left), kernel (right)

2.2.2 Convolution operation:

Convolution is performed by sliding the kernel or mask over a grey-level image. The kernel starts from the top left corner and moves through entire image within image boundaries. Each kernel position corresponds to a single output pixel. Each pixel is multiplied with the kernel cell value and added together. The output image will have M-m+1 rows and N-n+1 column, M image rows and N image columns, m kernel rows and n kernel columns.

The output image will be smaller when compared to the original image. This is due to the bottom and right edge pixels, which can't be completely mapped by the kernel therefore m –1 right hand pixel and n-1 bottom pixels can't be used.

Step 1: Gaussian filtering to remove noise

The first step of canny edge detection is to filter out any noise in the original image before trying to locate and detect any edges. The Gaussian filter is used to blur and remove unwanted detail and noise. By calculating a suitable 5 X 5 mask, the Gaussian smoothing can be performed using standard convolution method. A convolution mask is much smaller than the actual image. As a result, the mask is slid over the image, calculating every square of pixels at a time.

Gaussian filter uses 2D distribution to perform convolution. The larger the width of the Gaussian mask, the lower is the detector's sensitivity to noise. The weight of the matrix is concentrated at the center, therefore any noise appearing in the outside columns and rows will be eliminated, as the weight decreases outward from the center value. The localization error in the detected edges also increases slightly as the Gaussian width is increased. The increasing of standard deviation reduces or blurs the intensity of the noise.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 2.b: Gaussian filter in mathematical form

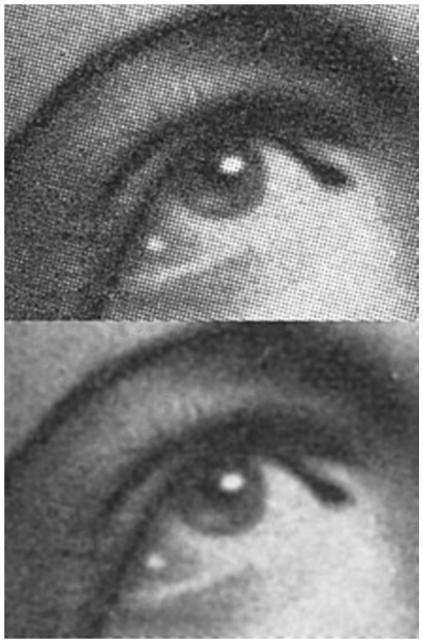


Figure 2.c**:** Original Image on top and Gaussian filtered image at bottom

Step 2: Sobel Operator

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image.

Then, the approximate absolute gradient magnitude (edge strength) at each point can be found by the formula below which is simpler to calculate compared to the above exact gradient magnitude.

Approximate gradient magnitude given below:

$$|G| = |G_x| + |G_y|$$

The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows).

Sobel G_x and G_y masks shown below each one estimates gradient x direction and y direction respectively:

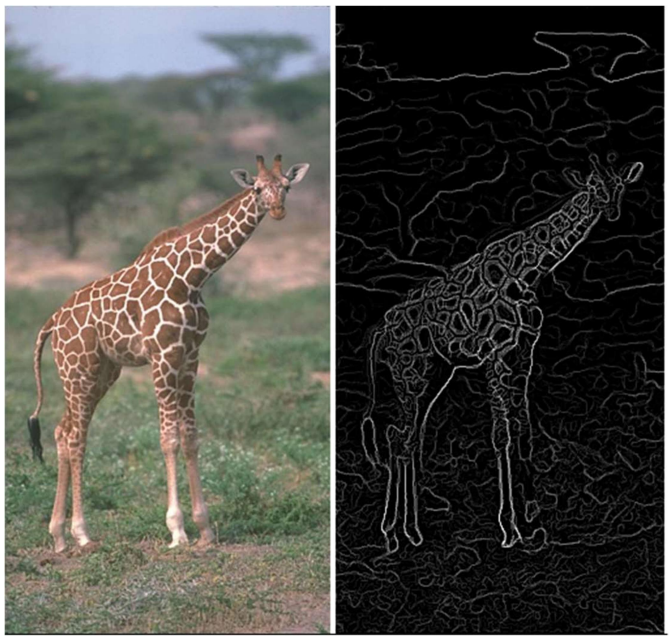


Figure 2.d: Original image and image after sobel operation

Step 3: Finding Gradient angle

Finding the edge direction is trivial once the gradient in the x and y directions are known. However, you will generate an error whenever sum of G_x is equal to zero i.e. G_x value in denominator meaning calculating arctan of infinity. So, the edge direction will equal to 90 or 0 degrees depend on Gx value and 0 degrees depend on G_y value.

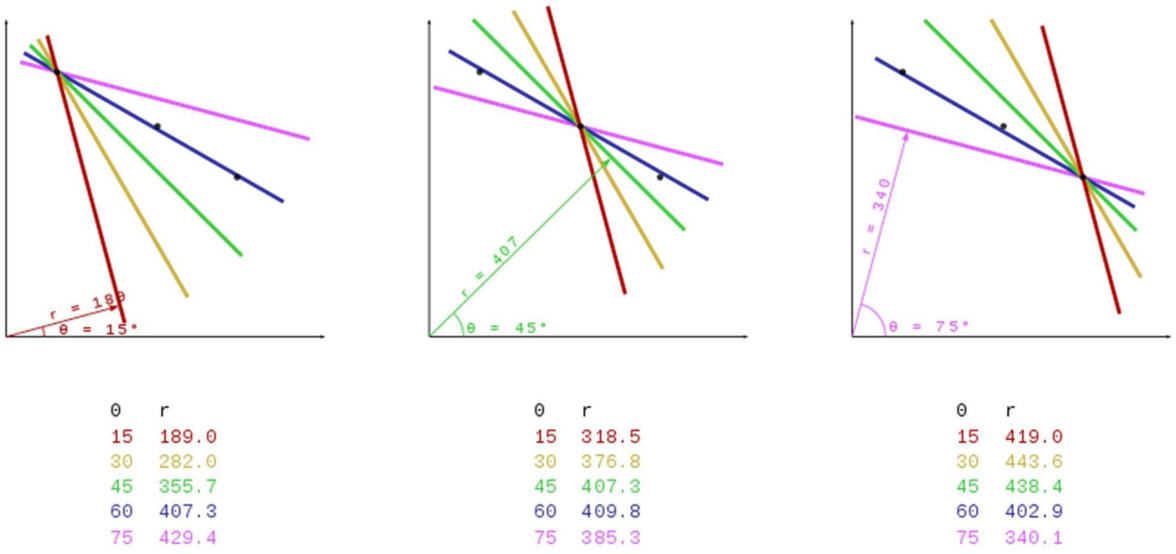
Step 4: Tracing the edge in the image using (angle)

Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image. So, if use the 5x5 matrix to calculate the angle of the edge, the smaller the matrix the fewer angles would have in the image.

x	x	x	x	x
x	x	x	x	x
x	x	a	x	x
x	x	x	x	x
x	x	x	x	x

By looking at the center pixel "a", there are four possible directions when describing the surrounding pixels - 0 degrees (in the horizontal direction), 45 degrees (along the positive diagonal), 90 degrees (in the vertical direction), or 135 degrees (along the negative diagonal), 180 degrees region is just an mirror region of 0 degrees region. Therefore, any edge direction calculated will be round up to the closest angle.

So, any edge direction falling within the A and E (0 to 22.5 & 157.5 to 180 degrees) is set to 0 degrees. Any edge direction falling in the D (22.5 to 67.5 degrees) is set to 45 degrees. Any edge direction falling in the C (67.5 to 112.5 degrees) is set to 90 degrees. And finally, any edge direction falling within the B (112.5 to 157.5 degrees) is set to 135 degrees.



0	r
15	189.0
30	282.0
45	355.7
60	407.3
75	429.4

0	r
15	318.5
30	376.8
45	407.3
60	409.8
75	385.3

0	r
15	419.0
30	443.6
45	438.4
60	402.9
75	340.1

Figure 2.e: Tracing an Edge of an Image

The bigger the matrix the greater number of angles one could get, which implies the edge angle will be more precise i.e. will follow the edge better, but on the down side it will be a bigger computation task as now the kernel/mask size is bigger.

Step 5: Non maximum Suppression

After the edge directions are known, non-maximum

suppression is applied. Non- maximum suppression is used to trace along the gradient in the edge direction and compare the value perpendicular to the gradient. Two perpendicular pixel values are compared with the value in the edge direction. If their value is lower than the pixel on the edge then they are suppressed i.e. their pixel value is changed to 0, else the higher pixel value is set as the edge and the other two suppressed with a pixel value of 0.

We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

Step 6: Hysteresis

Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. If a single threshold, T1 is applied to an image, and an edge has an average strength equal to T1, then due to noise, there will be instances where the edge dips below the threshold. Equally it will also extend above the threshold making an edge look like a dashed line. To avoid this, hysteresis uses 2 thresholds, a high and a low. Any pixel in the image that has a value greater than T1 is presumed to be an edge pixel, and is marked as such immediately. Then, any pixels that are connected to this edge pixel and that have a value greater than T2 are also selected as edge pixels. If you think of following an edge, you need a gradient of T2 to start but you don't stop till you hit a gradient below T1.

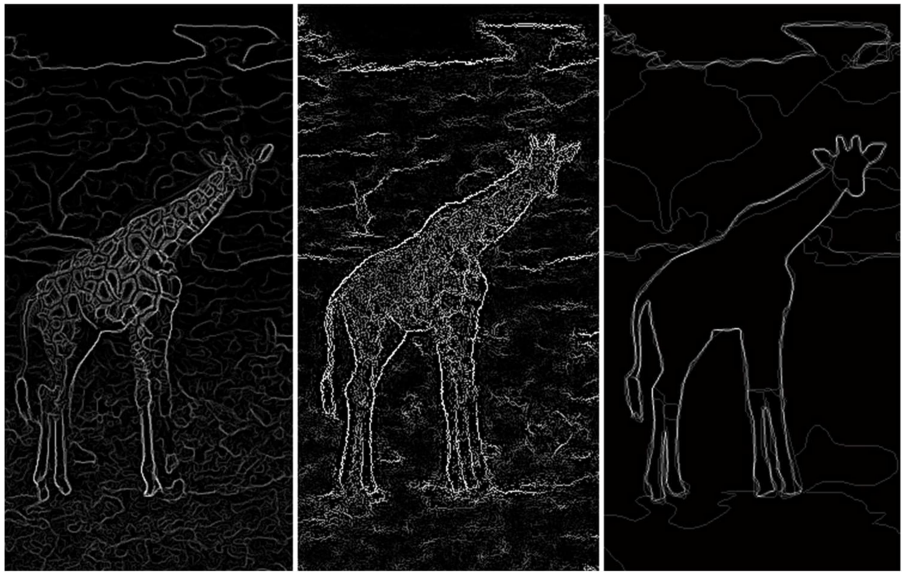


Figure 2.f: Showing Canny Edge Detection Process

2.3 Sample Input and Output of Canny Edge Detection Algorithm:

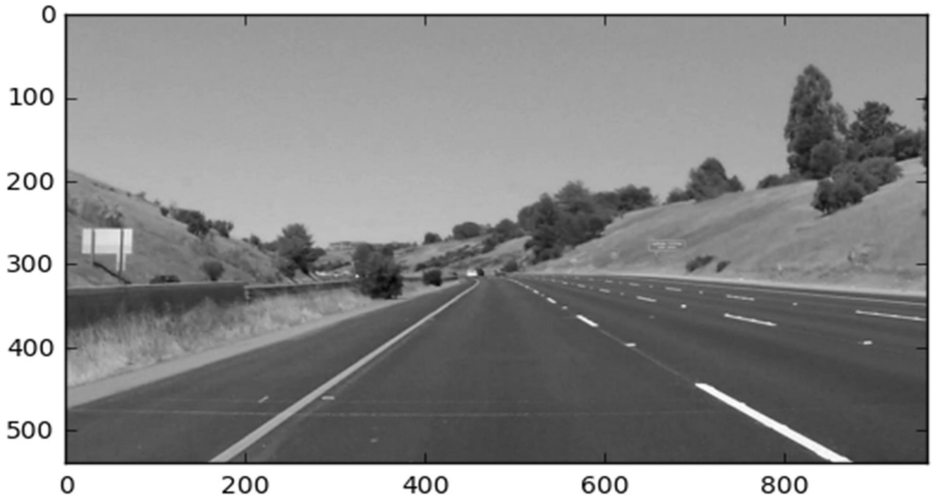


Figure 2.g: Describing the Input Image

Figure 2.h: Describing the Output Image

2.4 HOUGH TRANSFORM SPACE

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

The classical Hough transform was concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses. The Hough transform as it is universally used today was invented by Richard Duda and Peter Hart in 1972, who called it a "generalized Hough transform" after the related 1962 patent of Paul Hough. The transform was popularized in the computer vision community by Dana H Ballard through a 1981 journal article titled "Generalizing the Hough transform to detect arbitrary shapes"

2.4.1 Theory of Hough Transform Space

In automated analysis of digital images, a sub problem often arises of detecting simple shapes, such as straight lines, circles or ellipses. In many cases

an edge detector can be used as a pre-processing stage to obtain image points or image pixels that are on the desired curve in the image space. Due to imperfections in either the image data or the edge detector, however, there may be missing points or pixels on the desired curves as well as spatial deviations between the ideal line/circle/ellipse and the noisy edge points as they are obtained from the edge detector. For. The purpose of the Hough transform is to address this problem by making it possible to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects (Shapiro and Stockman, 304). The simplest case of Hough transform is detecting straight lines. In general, the straight-line $y = mx + b$ can be represented as a point (b, m) in the parameter space. However, vertical lines pose a problem. They would give rise to unbounded values of the slope parameter m. Thus, for computational reasons, Duda and Hart proposed the use of the Hesse normal form. These reasons, it is often non-trivial to group the extracted edge features to an appropriate set of lines, circles or ellipses.

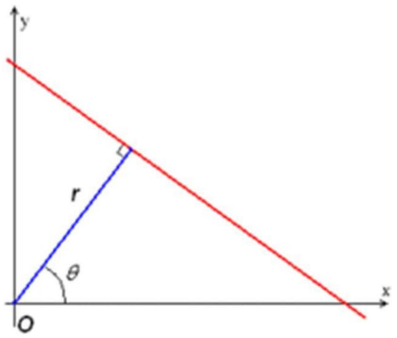


Figure 2.i: Hesse Normal Form Graph

It is therefore possible to associate with each line of the image a pair (r, θ). The (r, theta) plane is sometimes referred to as *Hough space* for the set of straight lines in two dimensions. This representation makes the Hough transform conceptually very close to the two-dimensional Radon transform. (They can be seen as different ways of looking at the same transform)

Given a *single point* in the plane, then the set of *all* straight lines going through that point corresponds to a sinusoidal curve in the (r, θ) plane, which is

unique to that point. A set of two or more points that form a straight line will produce sinusoids which cross at the (*r, θ*) for that line. Thus, the problem of detecting collinear points can be converted to the problem of finding concurrent curves.

2.5 IMPLEMENTATION OF HOUGH TRANSFORM SPACE

The linear Hough transform algorithm uses a two-dimensional array, called an accumulator, to detect the existence of a line described by r=x cos theta+ y sin theta. The dimension of the accumulator equals the number of unknown parameters, i.e., two, considering quantized values of r and θ in the pair (r, θ). For each pixel at *(x, y)* and its neighbourhood, the Hough transform algorithm determines if there is enough evidence of a straight line at that pixel. If so, it will calculate the parameters (r, θ) of that line, and then look for the accumulator's bin that the parameters fall into, and increment the value of that bin. By finding the bins with the highest values, typically by looking for local maxima in the accumulator space, the most likely lines can be extracted, and their (approximate) geometric definitions read off. (Shapiro and Stockman, 304) The simplest way of finding these *peaks* is by applying some form of threshold, but other techniques may yield better results in different circumstances – determining which lines are found as well as how many. Since the lines returned do not contain any length information, it is often necessary, in the next step, to find which parts of the image match up with which lines. Moreover, due to imperfection errors in the edge detection step, there will usually be errors in the accumulator space, which may make it non-trivial to find the appropriate peaks, and thus the appropriate lines.

The final result of the linear Hough transform is a two-dimensional array (matrix) similar to the accumulator—one dimension of this matrix is the quantized angle θ and the other dimension is the quantized distance r. Each element of the matrix has a value equal to the sum of the points or pixels that are positioned on the line represented by quantized parameters (r, θ). So, the element with the highest value indicates the straight line that is most represented in the input image.

2.6 VARIATIONS AND EXTENSIONS

2.6.1 Using the gradient direction to reduce the number of votes

An improvement suggested by O'Gorman and Clowes can be used to detect lines if one takes into account that the local gradient of the image intensity will necessarily be orthogonal to the edge. Since edge detection generally involves computing the intensity gradient magnitude, the gradient direction is often found as a side effect. If a given point of coordinates (*x, y*) happens to indeed be on a line, then the local direction of the gradient gives the θ parameter corresponding to said line, and the *r* parameter is then immediately obtained. (Shapiro and Stockman, 305) The gradient direction can be estimated to within 20°, which shortens the sinusoid trace from the full 180° to roughly 45°. This reduces the computation time and has the interesting effect of reducing the number of useless votes, thus enhancing the visibility of the spikes corresponding to real lines in the image.

2.6.2 Kernel Based Hough Transform (KHT)

Fernandes and Oliveira suggested an improved voting scheme for the Hough transform that allows a software implementation to achieve real-time performance even on relatively large images (e.g., 1280×960). The Kernel-based Hough transform uses the same (r, theta) parameterization proposed by Duda and Hart but operates on clusters of approximately collinear pixels. For each cluster, votes are cast using an oriented Elliptical-Gaussian kernel that models the uncertainty associated with the best-fitting line with respect to the corresponding cluster. The approach not only significantly improves the performance of the voting scheme, but also produces a much cleaner accumulator and makes the transform more robust to the detection of spurious lines.

2.6.3 3-D Kernel-based Hough transform for plane detection (3DKHT)

Limberger and Oliveira suggested a deterministic technique for plane detection in unorganized point clouds whose cost is nlogn in the number of samples, achieving real-time performance for relatively large datasets (up to 10^5 points on a 3.4 GHz CPU). It is based on a fast Hough-transform voting strategy for planar regions, inspired by the Kernel-based Hough transform (KHT). This 3D Kernel-based Hough transform

(3DKHT) uses a fast and robust algorithm to segment clusters of approximately co-planar samples, and casts votes for individual clusters (instead of for individual samples) on a (theta, sigma, row) spherical accumulator using a trivariate Gaussian kernel. The approach is several orders of magnitude faster than existing (non-deterministic) techniques for plane detection in point clouds, such as RHT and RANSAC, and scales better with the size of the datasets. It can be used with any application that requires fast detection of planar features on large datasets.

2.7 HOUGH TRANSFORM OF CURVES, AND ITS GENERALIZATION FOR ANALYTICAL AND NON-ANALYTICAL SHAPES

Although the version of the transform described above applies only to finding straight lines, a similar transform can be used for finding any shape which can be represented by a set of parameters. A circle, for instance, can be transformed into a set of three parameters, representing its centre and radius, so that the Hough space becomes three dimensional. Arbitrary ellipses and curves can also be found this way, as can any shape easily expressed as a set of parameters.

The generalization of the Hough transform for detecting analytical shapes in spaces having any dimensionality was proposed by Fernandes and Oliveira.[11] In contrast to other Hough transform-based approaches for analytical shapes, Fernandes' technique does not depend on the shape one wants to detect nor on the input data type. The detection can be driven to a type of analytical shape by changing the assumed model of geometry where data have been encoded (e.g., Euclidean space, projective space, conformal geometry, and so on), while the proposed formulation remains unchanged. Also, it guarantees that the intended shapes are represented with the smallest possible number of parameters, and it allows the concurrent detection of different kinds of shapes that best fit an input set of entries with different dimensionalities and different geometric definitions (e.g., the concurrent detection of planes and spheres that best fit a set of points, straight lines and circles).

For more complicated shapes in the plane (i.e., shapes that cannot be represented analytically in some 2D space), the Generalized Hough transform is used, which allows a feature to vote for a particular position, orientation and/or scaling of the shape using a predefined look-up table.

2.8 CIRCLE DETECTION PROCESS

Altering the algorithm to detect circular shapes instead of lines is relatively straightforward.

- First, we create the accumulator space, which is made up of a cell for each pixel. Initially each cell is set to 0.
- For each edge point (i, j) in the image, increment all cells which according to the equation of a circle ($(i-a)^2+(j-b)^2=r^2$) could be the centre of a circle. These cells are represented by the letter *a* in the equation.
- For each possible value of *a* found in the previous step, find all possible values of *b* which satisfy the equation.
- Search for local maxima in the accumulator space. These cells represent circles that were detected by the algorithm.

If we do not know the radius of the circle we are trying to locate beforehand, we can use a three-dimensional accumulator space to search for circles with an arbitrary radius. Naturally, this is more computationally expensive.

Y circle's area is still present within it.

2.9 Detection of 3D objects (Planes and cylinders)

Hough transform can also be used for the detection of 3D objects in range data or 3D point clouds. The extension of classical Hough transform for plane detection is quite straightforward. A plane is represented by its explicit equation $z=a_xx+a_yy+d$ for which we can use a 3D Hough space corresponding to a_x, a_y and d. This extension suffers from the same problems as its 2D counterpart i.e., near horizontal planes can be reliably detected, while the performance deteriorates as planar direction becomes vertical (big values of a_x and a_y amplify the noise in the data). This formulation of the plane has been used for the detection of planes in the point clouds acquired from airborne laser scanning [13] and works very well because in that domain all planes are nearly horizontal.

For generalized plane detection using Hough transform, the plane can be parametrized by its normal vector n (using spherical coordinates) and its distance from the origin row resulting in a three dimensional Hough space. This results in each point in the input

data voting for a sinusoidal surface in the Hough space. The intersection of these sinusoidal surfaces indicates presence of a plane. A more general approach for more than 3 dimensions requires search heuristics to remain feasible.

Hough transform has also been used to find cylindrical objects in point clouds using a two-step approach. The first step finds the orientation of the cylinder and the second step finds the position and radius.

2.10 USING WEIGHTED FEATURES

One common variation detail. That is, finding the bins with the highest count in one stage can be used to constrain the range of values searched in the next.

2.11 CAREFULLY CHOSEN PARAMETER SPACE

A high-dimensional parameter space for the Hough transform is not only slow, but if implemented without forethought can easily overrun the available memory. Even if the programming environment allows the allocation of an array larger than the available memory space through virtual memory, the number of page swaps required for this will be very demanding because the accumulator array is used in a randomly accessed fashion, rarely stopping in contiguous memory as it skips from index to index.

Consider the task of finding ellipses in an 800x600 image. Assuming that the radii of the ellipses are oriented along principal axes, the parameter space is four-dimensional. (x, y) defines the centre of the ellipse, and a and b denote the two radii. Allowing the centre to be anywhere in the image, adds the constraint $0<x<800$ and $0<y<600$. If the radii are given the same values as constraints, what is left is a sparsely filled accumulator array of more than 230 billion values.

A program thus conceived is unlikely to be allowed to allocate sufficient memory. This doesn't mean that the problem can't be solved, but only that new ways to constrain the size of the accumulator array are to be found, which makes it feasible. For instance:

1. If it is reasonable to assume that the ellipses are each contained entirely within the image, the range of the radii can be reduced. The largest the radii can be is if the centre of the ellipse is in the centre of the image, allowing the edges of the ellipse to stretch

to the edges. In this extreme case, the radii can only each be half the magnitude of the image size oriented in the same direction. Reducing the range of a and b in this fashion reduces the accumulator array to 57 billion values.

2. Trade accuracy for space in the estimation of the centre: If the centre is predicted to be off by 3 on both the x and y axis this reduces the size of the accumulator array to about 6 billion values.
3. Trade accuracy for space in the estimation of the radii: If the radii are estimated to each be off by 5 further reduction of the size of the accumulator array occurs, by about 256 million values.
4. Crop the image to areas of interest. This is image dependent, and therefore unpredictable, but imagine a case where all of the edges of interest in an image are in the upper left quadrant of that image. The accumulator array can be reduced even further in this case by constraining all 4 parameters by a factor of 2, for a total reduction factor of 16.

By applying just, the first three of these constraints to the example stated about, the size of the accumulator array is reduced by almost a factor of 1000, bringing it down to a size that is much more likely to fit within a modern computer's memory.

2.12 EFFICIENT ELLIPSE DETECTION ALGORITHM

Yonghong Xie and Qiang Ji give an efficient way of implementing the Hough transform for ellipse detection by overcoming the memory issues As discussed in the algorithm (on page 2 of the paper), this approach uses only a one-dimensional accumulator (for the minor axis) in order to detect ellipses in the image. The complexity is $O(N^3)$ in the number of non-zero points in the image.

2.13 EXISTING SYSTEM

In the current existing system is permitted only to use in ideal road conditions such as runway. This could not be used in general roads because the edge detection used till now was Simulink Edge Detection which is implemented in MATLAB. The secondary thing is in current system Hough transform Space is only used for angle rotation and has very limited road dataset to detect the objects in single dimension of an image.

2.14 LIMITATIONS OF EXISTING SYSTEM

The Hough transform is only efficient if a high number of votes fall in the right bin, so that the bin can be easily detected amid the background noise. This means that the bin must not be too small, or else some votes will fall in the neighbouring bins, thus reducing the visibility of the main bin.

Also, when the number of parameters is large (that is, when we are using the Hough transform with typically more than three parameters), the average number of votes cast in a single bin is very low, and those bins corresponding to a real figure in the image do not necessarily appear to have a much higher number of votes than their neighbours. The complexity increases at a rate of $O(A^{m-2})$ with each additional parameter, where A is the size of the image space and *m* is the number of parameters. (Shapiro and Stockman, 310) Thus, the Hough transform must be used with great care to detect anything other than lines or circles.

Finally, much of the efficiency of the Hough transform is dependent on the quality of the input data: the edges must be detected well for the Hough transform to be efficient. Use of the Hough transform on noisy images is a very delicate matter and generally, a denoising stage must be used before. In the case where the image is corrupted by speckle, as is the case in radar images, the Radon transform is sometimes preferred to detect lines, because it attenuates the noise through summation.

2.15 PROPOSED SYSTEM

In our proposed system we use Canny Edge Detection replacing the Simulink Edge Detection which is recent and efficient implementation in Python instead of MATLAB. Since, Python is the Scripting and Statistical Modelling Language it supports faster execution for mathematical functions which could be used by Canny Edge Detection technique. Secondly, we use Hough Transform Space for 3-Dimensional Object detection which could faster and accurate compared to single dimension object detection.

3. METHODOLOGY

3.1 IMAGE PROCESSING METHODOLOGY

Digital image processing consists of the manipulation of images using digital computers. Its use has been increasing exponentially in the last decades. Its applications range from medicine to entertainment, passing by geological processing and remote sensing. Multimedia systems, one of the pillars of the modern information society, rely heavily on digital image processing.

The discipline of digital image processing is a vast one, encompassing digital signal processing techniques as well as techniques that are specific to images. An image can be regarded as a function f (x, y) of two continuous variables x and y. To be processed digitally, it has to be sampled and transformed into a matrix of numbers. Since a computer represents the numbers using finite precision, these numbers have to be quantized to be represented digitally. Digital image processing consists of the manipulation of those finite precision numbers. The processing of digital images can be divided into several classes: image enhancement, image restoration, image analysis, and image compression. In image enhancement, an image is manipulated, mostly by heuristic techniques, so that a human viewer can extract useful information from it. Image restoration techniques aim at processing corrupted images from which there is a statistical or mathematical description of the degradation so that it can be reverted. Image analysis techniques permit that an image be processed so that information can be automatically extracted from it. Examples of image analysis are image segmentation, edge extraction, and texture and motion analysis. An important characteristic of images is the huge amount of information required to represent them. Even a grey-scale image of moderate resolution, say 512 × 512, needs $512 \times 512 \times 8 \approx 2 \times 106$ bits for its representation. Therefore, to be practical to store and transmit digital images, one needs to perform some sort of image compression, whereby the redundancy of the images is exploited for reducing the number of bits needed in their representation.

Digital image processing is to process images by computer. Digital image processing can be defined as subjecting a numerical representation of an object to a series of operations in order to obtain a desired result. Digital image processing consists of the conversion of a physical image into a corresponding digital image and the extraction of significant information from the digital image by applying various algorithms. Digital image processing mainly

includes image collection, image processing, and image analysis. At its most basic level, a digital image processing system is comprised of three components, i.e., a computer system on which to process images, an image digitizer, and an image display device. Physical images are divided into small areas called pixels. The division plan used often is the rectangular sampling grid method shown in Fig. 13.6, in which an image is segmented into many horizontal lines composed of adjacent pixels, and the value of each pixel position reflects the brightness of corresponding point on the physical Physical images cannot be directly analysed by a computer because the computer can only process digits rather than images, so an image must be converted into a digital form before processed by a computer. The conversion process is called digitization image.

At each pixel position, the brightness is sampled and quantized to obtain an integer value indicating the brightness of the corresponding position in the image. After the conversion of all pixels of an image is completed, the image can be represented by a matrix of integers. Each pixel has two attributions: position and grey level. The position is determined by the two coordinates of sampling point in the scanning line, namely row and column. The integer indicating the brightness of the pixel position is called grey level. Images displayed by digital matrix are called digital images, and all digital image processing is based on the digital matrix. The digital matrix is the object process

f(i, j) = the grey level of pixel (i, j).

On the basis of image processing, it is necessary to separate objects from images by pattern recognition technology, then to identify and classify these objects through technologies provided by statistical decision theory. Under the conditions that an image includes several objects, the pattern recognition consists of three phasesssed by a computer.

The first phase includes the image segmentation and object separation. In this phase, different objects are detected and separate from other background. The second phase is the feature extraction. In this phase, objects are measured. The measuring feature is to quantitatively estimate some important features of objects, and a group of the features are combined to make up a feature vector during feature extraction. The third phase is classification. In this phase, the output is just a decision to determine which category every object belongs to. Therefore, for pattern recognition, what input are images and what output

are object types and structural analysis of images. The structural analysis is a description of images in order to correctly understand and judge for the important information of images.

Image processing is the application of a set of techniques and algorithms to a digital image to analyse, enhance, or optimize image characteristics such as sharpness and contrast. Most image processing techniques involve treating the image as either a signal or a matrix and applying standard signal-processing or matrix manipulation techniques, respectively, to it. A pixel or "picture element" is the smallest sample of a two-dimensional image that can be programmatically controlled. The number of pixels in an image controls the resolution of the image. The pixel value typically represents its intensity in terms of shades of grey (value 0–255) in a grayscale image or RGB (red, green, blue, each 0–255) values in a colour image. A voxel or "volumetric pixel" is the three-dimensional counterpart of the 2D pixel. It represents a single sample on a three-dimensional image grid. Similar to pixels, the number of voxels in a 3D representation of an image controls its resolution. The spacing between voxels depends on the type of data and its intended use. In a 3D rendering of medical images such as CT scans and MRI scans, the size of a voxel is defined by the pixel size in each image slice and the slice thickness. The value stored in a voxel may represent multiple values. In CT scans, it is often the Hounsfield unit which can then be used to identify the type of tissue represented. In MRI volumes, this may be the weighting factor (T1, T2, T2*, etc.). Image arithmetic is usually performed at pixel level and includes arithmetic as well as logical operations applied to corresponding points on two or more images of equal size. Geometric transformations can be applied to digital images for translation, rotation, scaling, and shearing, as required. Matrix transformation algorithms are typically employed in this case. For binary and grayscale images, various morphological operations such as image opening and closing, skeletonization, dilation, erosion, and so on, may also be employed for pattern matching or feature extraction.

An image histogram represents the distribution of image intensity values for an input digital image. Histogram manipulation is often used to modify image contrast or for image segmentation when the range of values for the desired feature is clearly definable. Some common image processing applications are introduced as follows. Feature extraction is an area of image processing where specific characteristics within an input image are isolated using a set of algorithms. Some commonly used methods for this include contour tracing, thresholding, and template matching. Image segmentation is a common application of feature extraction which is often used with medical imaging to identify anatomical structures. Pattern and

template matching is useful in applications ranging from feature extraction to image substitution. It is also used with face and character recognition and is one of the most commonly used image processing applications. There are several image processing software packages available, from freely distributed ones such as ImageJ to expensive suites such as MATLAB and Avizo which range in functionality and targeted applications. We'll discuss only a few of the commonly used ones within medical physics/clinical engineering here. The image format most commonly used in medical applications is DICOM, providing a standardized structure for medical image management and exchange between different medical applications (see Chapter 8, "DICOM"). ImageJ is an open source, Java-based image processing program developed at the National Institute of Health. It provides various built-in image acquisition, analysis, and processing plugins as well as the ability to build your own using ImageJ's built-in editor and a Java compiler. User-written plugins make it possible to solve many bespoke image processing and analysis problems. ImageJ can display, edit, analyse, process, save, and print 8-bit colour and grayscale, 16-bit integer and 32-bit floating point images.43 It can read many standard image formats as well as raw formats. It is multithreaded, so time-consuming operations can be performed in parallel on multi-CPU hardware. It has built-in routines for most common image manipulation operations in the medical field including processing of DICOM images and image stacks such as those from CT and MRI.

Mimics44 is an image processing software for 3D design and modelling, developed by Materialise NV. It is used to create 3D surface models from stacks of 2D image data. These 3D models can then be used for a variety of engineering applications. Mimics calculates surface 3D models from stacked image data such as CT, micro-CT, CBCT, MRI, confocal microscopy, and ultrasound, through image segmentation. The region of interest (ROI) selected in the segmentation process is converted to a 3D surface model using an adapted marching cubes algorithm that takes the partial volume effect into account, leading to very accurate 3D models. The 3D files are represented in the STL format.45

The most common input format is DICOM, but other image formats such as TIFF, JPEG, BMP, and raw are also supported. Output file formats differ, depending on the subsequent application, but common 3D output formats include STL, VRML, PLY, and DXF. Mimics provides a platform to bridge stacked image data to a variety of different medical engineering applications such as finite element analysis (FEA; see the next section), computer-aided design (CAD), rapid prototyping, and so on.

MATLAB is a programming environment for algorithm development, data analysis, visualization, and numerical computation.46 It has a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modelling and analysis, and computational biology. The MATLAB Image Processing Toolbox™ provides a comprehensive set of reference-standard algorithms and graphical tools for image processing, analysis, visualization, and algorithm development. It also has built-in support for DICOM images and provides various functions to manipulate DICOM data sets. This makes it a widely used tool in various medical physics/clinical engineering research groups and related academia.

IDL is a cross-platform vectorised programming language used for interactive processing of large amounts of data including image processing.47 IDL also includes support for medical imaging via the IDL DICOM Toolkit add-on module. The image processing software packages mentioned here are but a few of the commonly used ones within a medical physics/clinical engineering environment partly due to their extensive libraries for medical image processing and partly for historical reasons.48 There are many more free-for-use as well as commercial software packages available providing varying degrees of functionality for different applications and, if there is a choice available, it is advisable to explore the options available for a particular task.

3.2 ARCHITECTURE

System architecture of a road detection from a single image using computer vision consists mainly the image which can be sent to a model and the output which consists of marking of detections of a road. The system architecture starts by selecting a required image which is captured by driving camera of a self-driving car. This should consist of all the details along with the road which should be detected by the computer. This image should be sent to the model. The model mainly consists of Edge detection model and Road Line detection model. The edge detection model occurs after implementing the Canny edge detection algorithm and Road line detection model occur after training the Hough transform space algorithm. Canny edge detection mainly consists of modules such as Gaussian blur algorithm for noise reduction, Smoothening of image, Gradient Calculations, Non-maximum suppression, double threshold and edge tracking by hysteresis. All these algorithms are compounded together to form a edge

detection model by Canny's process. The selected image is first sent to Canny's model and edges are found in an image. This edge detected image is then sent into road line detection model which is formed using Hough Transform Space algorithm. Hough transform space algorithm normalizes the sent image and then changes the value of θ in the normalized trigonometric line equation and thus detects the required road lines in an image. The only image with edges is sent into the Hough Transform Space algorithm, because the image with more noise takes large time for calculation rather than the image consists only edges of an image. The Hough Transform Space uses the Road line dataset to train itself for detecting the calculated line as a road line. This system architecture is thus used to detect the road lines in an image. When a required output for a single selected image is sent into the model, the testing dataset is sent into the model to get actual output of all the images. This architecture is thus used to build the model which could detect the road from an image. When a model is manually evaluated the process stops or will be used for training by improving the dataset used by Hough Transform Space algorithm and then process is continued until required output is observed from a model.

To build the architecture required by a project, we use incremental process model in which we test each prototype and then clubbed with the actual model on observing a correct output. Each Prototype is built along each model and then clubbed together with an actual model. In this way project is built using incremental model.

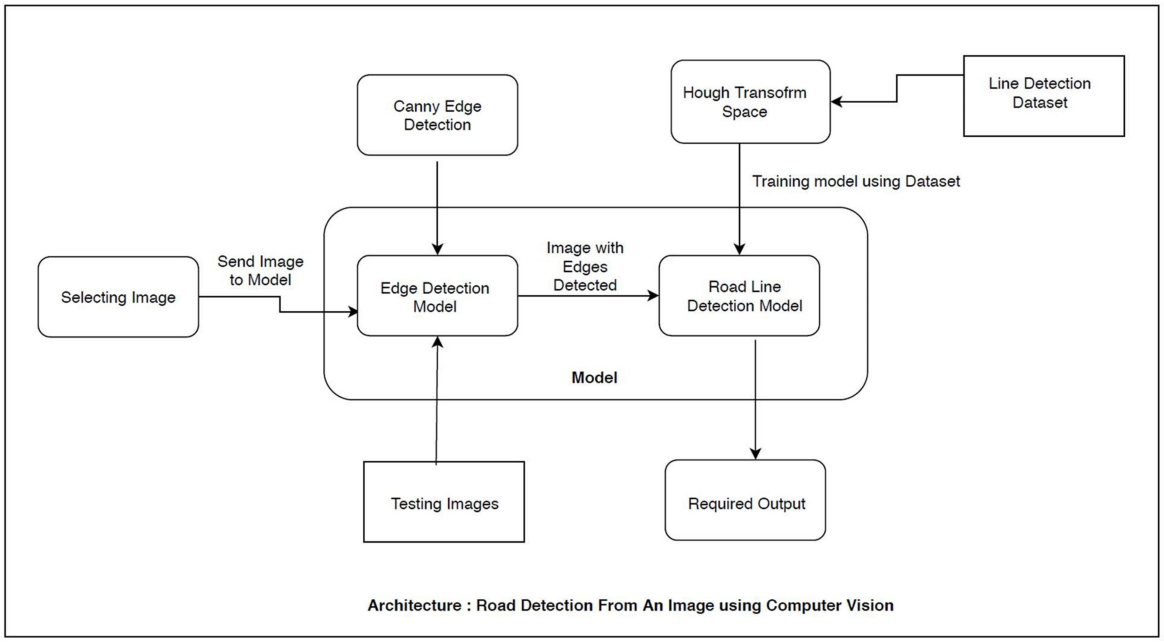


Figure 3.a: Image showing an architecture of a project: Road Detection from an Image using Computer Vision

3.3 CANNY EDGE DETECTION ARCHITECTURE:

This process consists of mainly four functions in it. The image can be of any type which consists of change in intensity in it. The change in intensity of pixels in an image defines the edges in an image. The canny edge detection mainly focuses on change in intensity in an image. The change in pixel's intensities from high intensity to low intensity is known as edge. At first, the color image is changed into black and white image and is passed to smoothening technique. We use Gaussian blur as a smoothening technique followed by gradient calculations, Non-maximum suppression and double threshold. Edge detection mainly use derivatives of pixel intensities in an image and then reduce the complexity of an image. The edge is detected when there is change in intensity from high to low which refers white shades to black shades (in gray scale image) in an image. Gray Scale image is used because it would be easy to process the gray scale image than the colored image. A gradient calculation process is used for calculating the θ through Sobel filters. Non-Maximum suppression is a process of thinning of edges that should be occurred in a required output image. Then a double thresholding is done to intensify the strong pixels of an output image and to close the intensity of weaker pixels in an image. Thus, canny edge detection is used and its architecture is built.

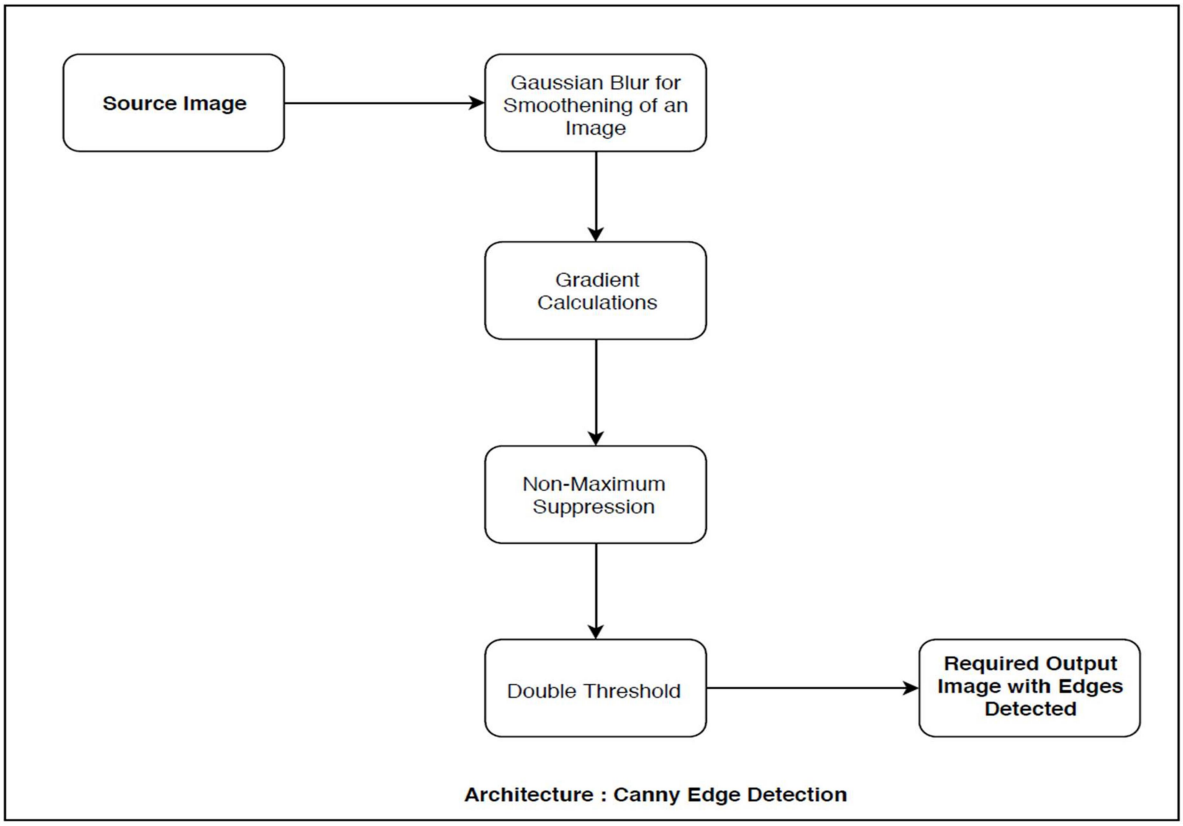


Figure 3.b: Image showing Architecture of Canny Edge Detection

The resultant image occurred through Canny Edge detection is sent to Hough Transform Space algorithmic model as an input and then produces a required output.

3.4 PROJECT MODULES

Road Detection from a single image using Computer Vision consists of image insertion, model building and then testing. The model evaluation is done manually by the developer. We divided the complete project into mainly four modules. They are as follows:

Module 1: Selecting the appropriate testing image

Module 2: Preprocessing the selected image

Module 3: Edge Detection Implementation

Module 4: Hough Transformation

Module 5: Evaluating the output

Module 1: Selecting the appropriate testing image

It is the most important process in the project. Single Image from testing dataset is taken such a way that it reaches our implementation of a model. Each model we implement takes a resultant image as an input and process it further to produce an output. This selection of image is more important because implementation of each model requires an image input for processing. And if the processing is done, then output is produced. If output produced for the testing image is same as required, then the resultant image is sent to next process that we need to develop further. In order to observe the clear output, the best suitable image should be selected such a way that the testing image should be able to produce the clear required output at the end of processes.

Module 2: Preprocessing the selected image

Preprocessing plays a major role in producing the required output in sufficient required amount of time. Preprocessing of selected image mainly undergo the gray scale conversion and smoothening techniques which would be considered as the first process in Canny's process. The selected image is converted into gray scale through the open source computer vision package. And then smoothening is applied by implementing the Gaussian Blur algorithm on the selected gray scale image. A gray scale image mainly consists of change in variants from white to black that represents the color mixes of red, green and blue. The normalization is main process of Gaussian Blur process conversion which is done through multiplying each intensities of a pixels by their corresponding normalized matrix values. Thus, preprocessing is done on the selected image. This conversion of gray scale image and reducing noise in the image can help by reducing the processing time in the next large processes. In machine learning projects in general, you usually go through a data preprocessing or cleaning step. As a machine learning engineer, you'll spend a good amount of your time cleaning up and preparing the data before you build your learning model. The goal of this step is to make your data ready for the ML model to make it easier to analyze and process computationally, as it is with images. Based on the problem you're solving and the dataset in hand, there's some data massaging required before you feed your images to the ML model.

Image processing could be simple tasks like image resizing. In order to feed a dataset of images to a convolutional network, they must all be the same size. Other processing

tasks can take place like geometric and color transformation or converting color to grayscale and many more.

The acquired data are usually messy and come from different sources. To feed them to the ML model (or neural network), they need to be standardized and cleaned up. More often than not, preprocessing is used to conduct steps that reduce the complexity and increase the accuracy of the applied algorithm. We can't write a unique algorithm for each of the condition in which an image is taken, thus, when we acquire an image, we tend to convert it into a form that allows a general algorithm to solve it.

Data preprocessing techniques might include:

- Grey Scale Image Conversion:

  Convert color images to grayscale to reduce computation complexity: in certain problems you'll find it useful to lose unnecessary information from your images to reduce space or computational complexity.

  For example, converting your colored images to grayscale images. This is because in many objects, color isn't necessary to recognize and interpret an image. Grayscale can be good enough for recognizing certain objects. Because color images contain more information than black and white images, they can add unnecessary complexity and take up more space in memory (Remember how color images are represented in three channels, which means that converting it to grayscale reduces the number of pixels that need to be processed).

Module 3: Edge Detection Implementation

The next step in the process if edge detection, which is main part in the program and required to detect the edges in the image irrespective of details present in an image. We use Canny Edge Detection Algorithm to implement the edge detection techniques because the other processes which are also used to find the edges in an image would contain detailed images compared to Canny Edge Detection Technique. Canny Edge Detection technique mainly consists of four processes in it. They are Gaussian Blur which we have performed for smoothening of image as preprocessing technique, Gradient Calculation which is used to calculate θ for boundary selection of an image

followed by Non-Maximum suppression and double threshold required for strengthening the lines occurred in edged image of previous functions. Thus, we get the image with edges which is applied as an input to Hough transformation techniques.

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

The Canny filter is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients. The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude. The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding.

The general criteria for edge detection include:

- Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible.
- The edge point detected from the operator should accurately localize on the center of the edge.
- A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

Module 4: Hough Transformations

Hough Transformations require a Hough transformation space which is used to rotate the angles of a trigonometric line equation and then specify the lines present in an edge detected image. If the trigonometric line in rotation meets the edges in the image, then it may consider for applying the model trained for detecting roads in an image. The training is done in Hough transformation space which is used to detect the actual road lines from an image. When the Hough transformations and training is done, then the road lines are detected on the selected image. The training of model is improvised until the correct output is observed from a selected image.

The Hough transform is a technique which can be used to isolate features of a particular shape within an image. Because it requires that the desired features be specified in some parametric form, the classical Hough transform is most commonly used for the detection of regular curves such as lines, circles, ellipses, etc. A generalized Hough transform can be employed in applications where a simple analytic description of a feature(s) is not possible. Due to the computational complexity of the generalized Hough algorithm, we restrict the main focus of this discussion to the classical Hough transform. Despite its domain restrictions, the classical Hough transform (hereafter referred to without the classical prefix) retains many applications, as most manufactured parts (and many anatomical parts investigated in medical imagery) contain feature boundaries which can be described by regular curves. The main advantage of the Hough transform technique is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise.

The Hough technique is particularly useful for computing a global description of a feature(s) (where the number of solution classes need not be known a priori), given (possibly noisy) local measurements. The motivating idea behind the Hough technique for line detection is that each input measurement (e.g. coordinate point) indicates its contribution to a globally consistent solution (e.g. the physical line which gave rise to that image point).

As a simple example, consider the common problem of fitting a set of line segments to a set of discrete image points (e.g. pixel locations output from an edge detector). Figure 1 shows some possible solutions to this problem. Here the lack of a priori knowledge about the number of desired line segments (and the ambiguity about what constitutes a line segment) render this problem under-constrained.

We can analytically describe a line segment in a number of forms. However, a convenient equation for describing a set of lines uses parametric or normal notion:

$$X \cos t + Y \sin t = r$$

where r is the length of a normal from the origin to this line and t is the orientation of r with respect to the X-axis. (See Figure 2.) For any point (X, Y) on this line, r and t are constant.

In an image analysis context, the coordinates of the point(s) of edge segments (i.e. (X, Y)) in the image are known and therefore serve as constants in the parametric line equation, while r and t are the unknown variables we seek. If we plot the possible (r, t) values defined by each (X, Y), points in Cartesian image space map to curves (i.e. sinusoids) in the polar Hough parameter space. This point-to-curve transformation is the Hough transformation for straight lines. When viewed in Hough parameter space, points which are collinear in the Cartesian image space become readily apparent as they yield curves which intersect at a common (r, t) point.

The transform is implemented by quantizing the Hough parameter space into finite intervals or accumulator cells. As the algorithm runs, each (X, Y) is transformed into a discretized (r, t) curve and the accumulator cells which lie along this curve are incremented. Resulting peaks in the accumulator array represent strong evidence that a corresponding straight line exists in the image.

We can use this same procedure to detect other features with analytical descriptions.

In this case, the computational complexity of the algorithm begins to increase as we now have three coordinates in the parameter space and a 3-D accumulator. (In general, the computation and the size of the accumulator array increase polynomials with the number of parameters. Thus, the basic Hough technique described here is only practical for simple curves.)

Module 5: Evaluating the output

Evaluation of output is done through confusion matrix and the accuracy metrics when a testing dataset is passed. When all the images of testing dataset are passed into the model, then the output is observed on every image and then the true positives, false positives, true negatives and false negatives are noted. This forms a confusion matrix and then accuracy, precision scores can be noted. This process is known as Evaluation. Thus, all the accuracy and precision of a model are noted.

4. EXPERIMENTAL RESULTS

4.1 SYSTEM CONFIGURATION

4.1.1 Software Configuration

These are the Software Configurations that are required.

- Operating System: Windows 10/8/7 (incl. 64-bit), Mac OS, Linux
- Language: Python 3
- IDE: Jupyter Notebook/ Spyder 3
- Framework: Tkinter

4.1.2 Hardware Configuration

These are minimum Hardware configurations that are required.

- Processor: Intel core 2 duo or higher.
- RAM: 1 GB or higher
- HDD: 256 GB or higher
- Monitor: 1024 x 768 minimum screen resolution.
- Keyboard: US en Standard Keyboard.

4.2 SAMPLE CODE

- **Gaussian Blur**

```
import numpy as np
import cv2

img = cv2.imread('images/cat.jpg', cv2.IMREAD_GRAYSCALE)
img_out = img.copy()

height = img.shape[0]
width = img.shape[1]

gauss = (1.0/57) * np.array(
        [[0, 1, 2, 1, 0],
        [1, 3, 5, 3, 1],
        [2, 5, 9, 5, 2],
        [1, 3, 5, 3, 1],
        [0, 1, 2, 1, 0]])
sum(sum(gauss))

for i in np.arange(2, height-2):
    for j in np.arange(2, width-2):
        sum = 0
        for k in np.arange(-2, 3):
            for l in np.arange(-2, 3):
                a = img.item(i+k, j+l)
                p = gauss[2+k, 2+l]
                sum = sum + (p * a)
        b = sum
        img_out.itemset((i,j), b)

cv2.imwrite('images/filter_gauss.jpg', img_out)

cv2.imshow('image',img_out)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Convolution Operation**

```
import numpy as np

# X and F are numpy matrices
def convolve_np(X, F):
    X_height = X.shape[0]
    X_width = X.shape[1]

    F_height = F.shape[0]
    F_width = F.shape[1]

    H = (F_height - 1) / 2
    W = (F_width - 1) / 2

    out = np.zeros((X_height, X_width))

    for i in  np.arange(H,  X_height-H):
        for j  in np.arange(W, X_width-W):
            sum = 0
            for  k in  np.arange(-H, H+1):
                for l in np.arange(-W,  W+1):
                    a = X[i+k,j+l]
                    w = F[H+k,W+l]
                    sum += (w * a)
            out[i,j] = sum

    return out
```

- **Edge Orientation with respect to theta**

```
import numpy as np

def get_orientation_sector(dx,dy):
```

```
    # rotate (dx,dy) by pi/8
    rotation = np.array([[np.cos(np.pi/8), -np.sin(np.pi/8)],
                         [np.sin(np.pi/8), np.cos(np.pi/8)]])
    rotated = np.dot(rotation, np.array([[dx], [dy]]))

    if rotated[1] < 0:
        rotated[0] = -rotated[0]
        rotated[1] = -rotated[1]

    s_theta = -1
    if rotated[0] >= 0 and rotated[0] >= rotated[1]:
        s_theta = 0
    elif rotated[0] >= 0 and rotated[0] < rotated[1]:
        s_theta = 1
    elif rotated[0] < 0 and -rotated[0] < rotated[1]:
        s_theta = 2
    elif rotated[0] < 0 and -rotated[0] >= rotated[1]:
        s_theta = 3

    return s_theta
```

- **Local Maximum Separation For Canny's Process**

```
def is_local_max(E_mag, i, j, s_theta, t_low):
    mC = E_mag[i,j]
    if mC < t_low:
        return False
    else:
        mL = -1
        if s_theta == 0:
            mL = E_mag[i, j-1]
        elif s_theta == 1:
            mL = E_mag[i-1, j-1]
        elif s_theta == 2:
            mL = E_mag[i-1, j]
```

```
    elif s_theta == 3:
        mL = E_mag[i-1, j+1]

    mR = -1
    if s_theta == 0:
        mL = E_mag[i, j+1]
    elif s_theta == 1:
        mL = E_mag[i+1, j+1]
    elif s_theta == 2:
        mL = E_mag[i+1, j]
    elif s_theta == 3:
        mL = E_mag[i+1, j-1]

    return mL <= mC and mC >= mR
```

- **Edge Tracing and Threshold**

```
import numpy as np

def trace_and_threshold(E_nms, E_bin, i, j, t_low):
    E_bin[i,j] = 255

    jL = np.max([j-1, 0])
    jR = np.min([j+1, E_bin.shape[1]])

    iT = np.max([i-1, 0])
    iB = np.min([i+1, E_bin.shape[0]])

    for ii in np.arange(iT, iB):
        for jj in np.arange(jL, jR):
            if E_nms[ii,jj] >= t_low and E_bin[ii,jj] == 0:
                trace_and_threshold(E_nms, E_bin, ii, jj, t_low)

    return
```

- **Integrating the processes**

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import os

def   readImage(imageName):
    img = cv.imread(imageName)
    return img

def showImage(img, msg):
    cv.imshow(msg, img)
    cv.waitKey(0)
    cv.destroyAllWindows()

def grayImage(img):
    gray  =  cv.cvtColor(img,  cv.COLOR_RGB2GRAY)
    showImage(gray, "Gray Scaled Image - Press Any Key to Continue")
    return gray

def GaussianBlur(img):
    gaussianBlur = cv.GaussianBlur(img, (5,5), 0)
    showImage(gaussianBlur, "Gaussian Blured Image - Press Any Key to Continue")
    return gaussianBlur

def edgeDetection(img):
    canny = cv.Canny(img, 50,150)
    showImage(canny, "Edge Detection Image - Press Any Key to Continue")
    return canny

def graphShow(img):
    plt.imshow(img)
    plt.show()
```

```
# Manual technique to find the mask of an image - Finding mask manually
def findingMask(img):
    height = img.shape[0]
    trianglePolygon = np.array([[(200,height), (1100,height), (600,300)]])
    mask = np.zeros_like(img) # creates array same shape as image of zeros
    cv.fillPoly(mask, trianglePolygon, color = 255) #color 255 is white
    return mask

def regionOfInterest(img):
    mask = findingMask(img)
    roadDetectedImage = cv.bitwise_and(img, mask)
    return roadDetectedImage

def displayFoundLines(img, lines):
    #finalImage = np.copy(img)
    onlyLines = np.zeros_like(img)
    print("Lines Detected in an Image usign Hough Transfrom Space are : \n")
    if lines is not None:
        for eachLine in lines:
            print(eachLine)
            x1, y1, x2, y2 = eachLine.reshape(4)
            cv.line(onlyLines, (x1, y1), (x2,y2), (128,0,64), 5) #Drawing Line on an Image
and 255,154,100 is color of the line and thickness of a line
    return onlyLines

def displayFoundLinesOnRealImage(originalImg, linedImg):
    finalImage = cv.addWeighted(originalImg, 0.7, linedImg, 1, 1)
    return finalImage

if __name__=="__main__":
    os.system("pip install numpy")
    os.system("pip install matplotlib")
    os.system("pip install opencv-contrib-python")
```

```
    roadImage = readImage("roadImage.jpg")
    roadImageBackup = np.copy(roadImage)
    showImage(roadImage, "Testing Image - Press Any Key to Continue")
    grayScaleImage = grayImage(roadImageBackup)
    gaussianBlur = GaussianBlur(grayScaleImage)
    EdgeDetection = edgeDetection(gaussianBlur)
    graphShow(EdgeDetection)
    print(7*" "+"Canny Edges plotted on graph"+3*"\n")
    # roadImage.shape[0] = Height (Maximum y) and roadImage.shape[1] = Width (Maximum x)

    detectedImageWithAreaOfInterest = regionOfInterest(EdgeDetection)
    showImage(detectedImageWithAreaOfInterest, "Road Detected Manually by AND operation on masking and image - Press Any Key to Continue")

    lines = cv.HoughLinesP(detectedImageWithAreaOfInterest, 2, np.pi/180, 100, np.array([]), minLineLength = 40, maxLineGap = 5) # r=2, theta and threshold
    #showImage(detectedImageWithAreaOfInterest, "Hough Transform Space for Detecting lines in an Image - Press Any Key to Continue")
    onlyLines = displayFoundLines(roadImageBackup, lines)
    showImage(onlyLines, "Lines detected by hough transform space - Press Any Key to Continue")
    finalImage = displayFoundLinesOnRealImage(roadImageBackup,onlyLines)
    showImage(finalImage, "Final Image with lines detected by hough transform space - Press Any Key to Continue")
```

4.3 SCREENSHOTS AND OUTPUTS

First we will have a look at our projects User Interface. These are the screenshots of project outputs.

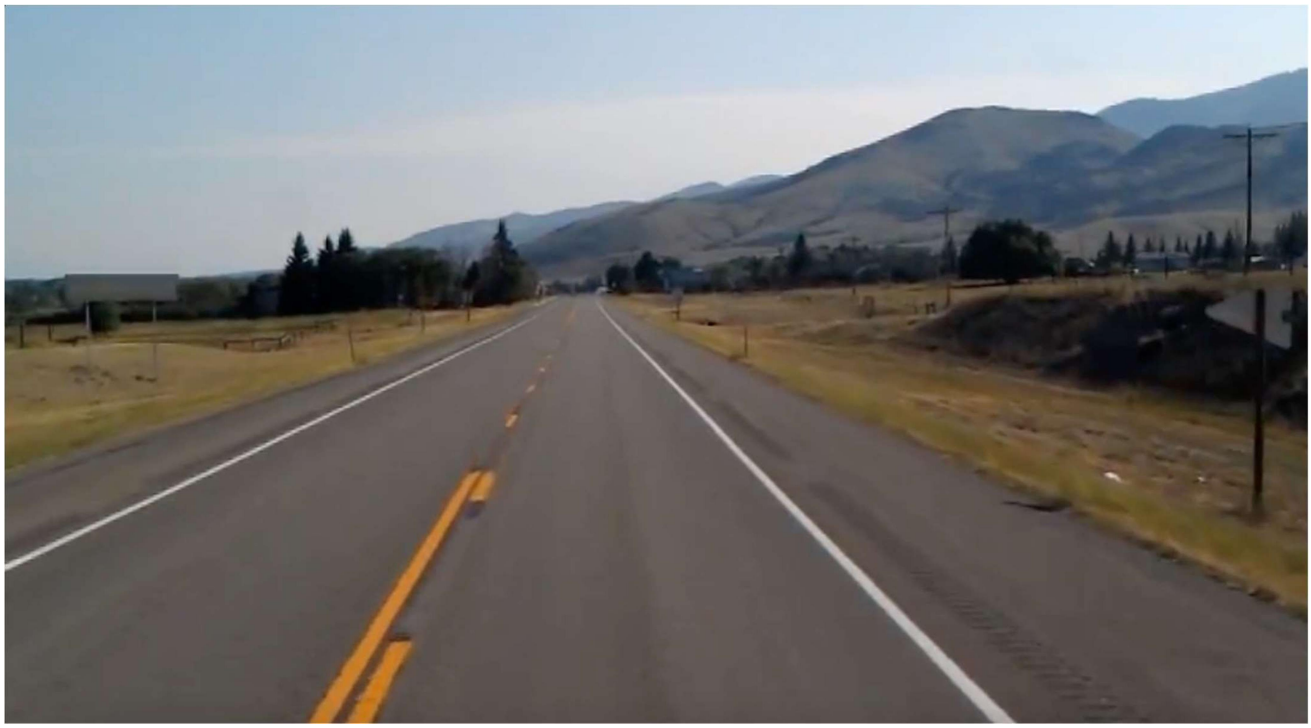


Figure 4.a: Selected testing image so that it undergoes into process

Figure 4.b: Grey scaled image output of a selected image

Figure 4.c: Gaussian Blur applied on grey scaled image

Figure 4.d: Canny Edge Detection output when applied on Gaussian Blurred image

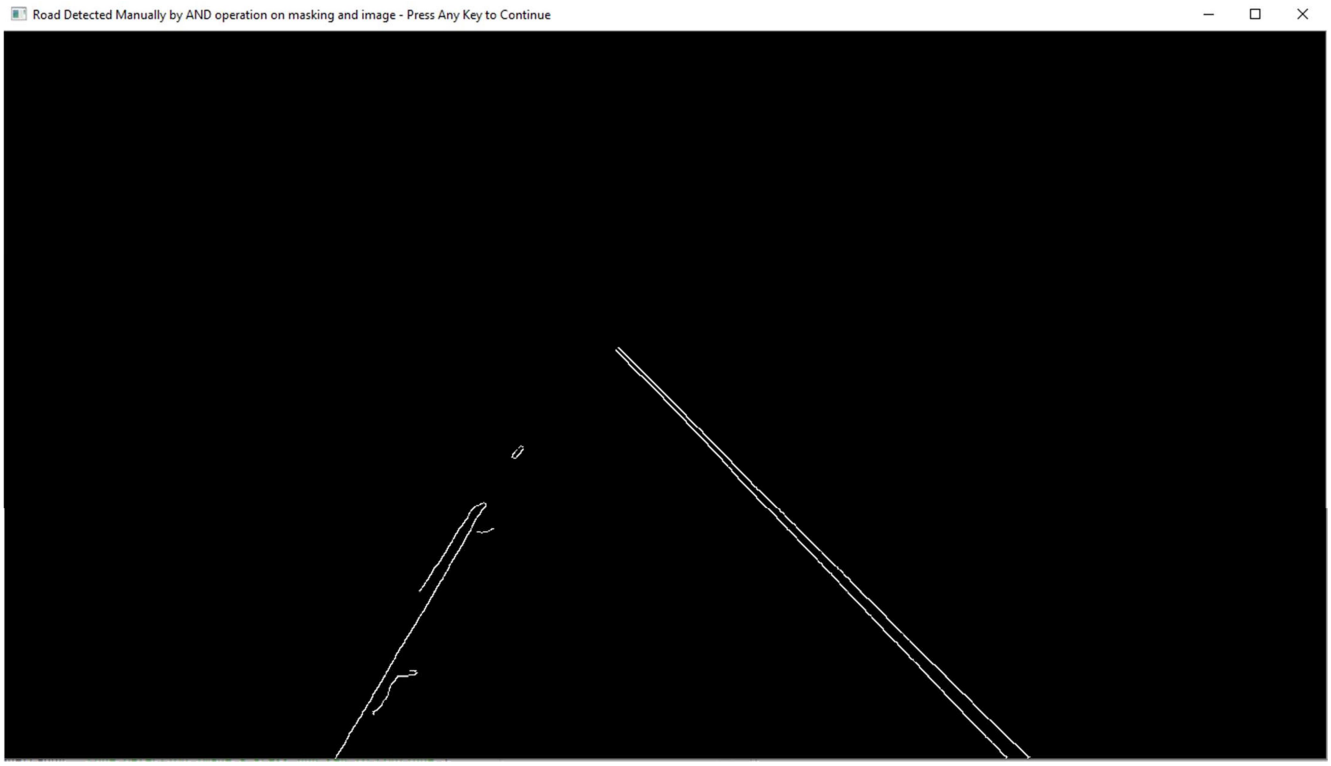


Figure 4.e: Masking applied to edge detected image to get a required part in an image.

Figure 4.f: Lines detected through Hough Transformations by changing the angle theta.

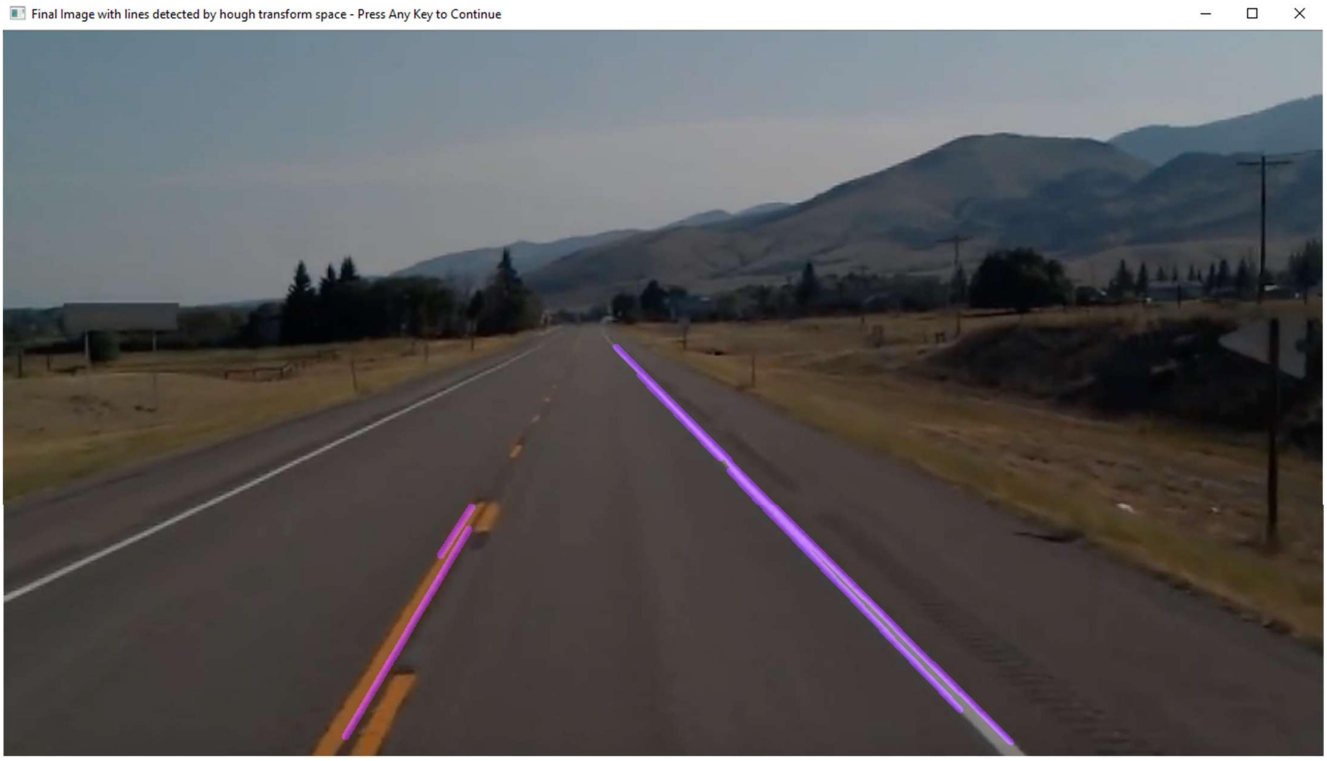


Figure 4.g: Final image with detected lines of Hough Transform Space

5. CONCLUSION AND FUTURE WORKS

5.1 CONCLUSION

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 vehicle is to automatically detect lane lines using an algorithm. The road detection region of interest (ROI), must be flexible. When driving up or down a steep incline, the horizon will change and no longer be a product of the proportions of the frame. This is also something to consider for tight turns and bumper to bumper traffic. This project is entirely based on image processing and road detection in self-driving vehicles in which has a great scope in future. We have completed the entire implementation using specific algorithms to detect the road clearly. If the people's thought hasn't changed about the self-driving cars being safe, these cars are already safe and are becoming safer. Only if they believe and give a try to technology, they get to enjoy the luxury of computerized driving.

Driverless cars appear to be an important next step in transportation technology. They are a new all-media capsule- text to your heart's desire and it's safe. Developments in autonomous cars is continuing and the software in the car is continuing to be updated. Though it all started from a driverless thought to radio frequency, cameras, sensors, more semi-autonomous features will come up, thus reducing the congestion, increasing the safety with faster reactions and fewer errors.

5.2 FUTURE WORKS

The updates in algorithms can be done easily since we do modular implementation and works could be continued in future for change in implementation of model. We use the pickle file of model to insert in required areas and then could be easily transferred onto products. So, this could easily avoid compiling the entire large code every time. We can also improve the project by introducing the new future that the road can be detected in the dark i.e., during night. Driving at night. The colour identification and selection process works very well in day light. Introducing shadows will create some noisy, but it will not provide as rigorous a test as driving in night, or in limited visibility conditions (e.g. heavy fog). And this project can detect the lane only on the bitumen road but not the loamy soil road which is common in Indian villages. So, this project can be further improved to detect the loamy soil roads that present in the villages and prevent the accidents. As we implemented our project in the python we can also implement it in the upcoming computer language Julia which is simple and easily understanding. This road detection system plays a key role in the self-driving vehicles which is the most awaited project. Self-driving cars are poised to revolutionize the transportation industry. There have been many significant shifts in the auto industry since the beginning of commercial auto production roughly eight decades ago, but the basic formula of a human operator guiding a vehicle using a steering wheel and pedals has held pretty steady across that time span. That's changing quickly. Newer cars already have automated features for things like parking and collision detection, and auto and tech companies are hard at work to deliver vehicles that are capable of advanced navigation without input from a human driver. Automated driving systems (ADS) for cars, trucks, and vans

are on the way and set to bring some huge changes and opportunities. Here's a look at what the future might hold, some of the big roadblocks facing driverless technology progression, and what the major breakthroughs from the improvement of the tech could mean. Autonomous car functionality is often referred to and judged on a six-tier scale, with Level 0 representing no autonomous component and a Level 5 ranking signifying an autonomous vehicle that can consistently perform all driving functions without the need for any human input. The table below outlines the basic characteristics of each of the five levels of autonomous cars as outlined by the Society of Automotive Engineers (SAE). People who currently reject self-driving cars would've said no to modern technology and automatic systems. Some reports and experts suggest that ADS vehicles are already safer than human-operated vehicles when it comes to performing some driving functions. Self-driving cars don't suffer from sleep deprivation, and they can't drive under the influence of drugs or alcohol. They also have wider fields of vision and are designed to obey traffic laws, while human drivers will sometimes disregard laws or fail to follow them due to being distracted.

Self-driving technology has the potential to reduce crashes, but some high-profile accidents have raised questions about risks posed by poorly functioning autonomous-driving systems. In January 2016, a man was killed in China after his Tesla crashed into the back of a cleaning vehicle. The Tesla reportedly had its self-driving features activated at the time of the crash. This marked the first reported death in which a vehicle's ADS features were viewed as a potential contributing factor, although the police did find that the Tesla driver had not been paying attention to the road in accordance with the autopilot rules. The public data available for evaluating how safe self-driving cars are remains somewhat limited. Most of the cities and states in which autonomous driving testing is taking place tend to have relatively dry weather conditions and simple road systems that make it easier for driverless vehicles to function. California is also the only state in America that requires companies testing driverless cars to submit reports detailing each accident involving autonomous vehicles on public roads. But even California's reporting requirements do not provide a very detailed view into the performance of driverless vehicle systems.

It's difficult to make a clear across-the-board assessment about the level of safety that autonomous vehicles provide. Competing automated driving systems also rely on different technology platforms, and some systems are likely safer than others, but it does appear that autonomous driving systems can outperform human drivers in ideal operating conditions. A study from Axios found that humans were responsible for most of the accidents in California involving self-driving cars from 2014 to 2018. However, as noted by Waymo's John Krafcik, the technology still needs improvements to make it more functional in a wider range of scenarios.

6. REFERENCES

1. General Road Detection From A Single Image, TIP-05166-2009, ACCEPTED, Hui Kong , Member, IEEE, Jean-Yves Audibert, and Jean Ponce , Fellow, IEEE Willow Team, Ecole Normale Superieure / INRIA / CNRS, Paris, France Imagine team, Ecole des Ponts ParisTech, Paris, France.

2. J. C. McCall and M. M. Trivedi, "Video based lane estimation and tracking for driver assistance: Survey, system, and evaluation," IEEE Trans. on Intelligent Transportation Systems, pp. 20–37, 2006.

3. K.-Y. Chiu and S.-F. Lin, "Lane detection using color-based segmentation," IEEE Intelligent Vehicles Symposium, 2005. 1

4. H. Kong, J.-Y. Audibert, and J. Ponce, "Vanishing point detection for road detection," CVPR, 2009.

5. Y. Wang, E. K. Teoh, and D. Shen, "Lane detection and tracking using b-snake," Image and Vision Computing, pp. 269–280, 2004

6. A. Lookingbill, J. Rogers, D. Lieb, J. Curry, and S. Thrun, "Reverse optical flow for self-supervised adaptive autonomous robot navigation," IJCV, vol. 74, no. 3, pp. 287–302, 2007.

7. A. Broggi, C. Caraffi, R. I. Fedriga, and P. Grisleri, "Obstacle detection with stereo vision for off-road vehicle navigation," IEEE International Workshop on Machine Vision for Intelligent Vehicles, 2005.

8. J. Sparbert, K. Dietmayer, and D. Streller, "Lane detection and street type classification using laser range images," IEEE Proceedings in Intelligent transportation Systems, pp. 456–464, 2001.

9. J. B. Southhall and C. Taylor, "Stochastic road shape estimation," ICCV, pp. 205–212, 2001.

APPENDIX

PYTHON 3.X

To get started working with Python 3, you'll need to have access to the Python interpreter. There are several common ways to accomplish this:

- Python can be obtained from the Python Software Foundation website at python.org. Typically, that involves downloading the appropriate installer for your operating system and running it on your machine.
- Some operating systems, notably Linux, provide a package manager that can be run to install Python.

On macOS, the best way to install Python 3 involves installing a package manager called Homebrew. You'll see how to do this in the relevant section in the tutorial.

On mobile operating systems like Android and iOS, you can install apps that provide a Python programming environment. This can be a great way to practice your coding skills on the go.

Alternatively, there are several websites that allow you to access a Python interpreter online without installing anything on your computer at all.

It is highly unlikely that your Windows system shipped with Python already installed. Windows systems typically do not. Fortunately, installing does not involve much more than downloading the Python installer from the python.org website and running it.

If you are running Windows 10 Creators or Anniversary Update, you actually have another option for installing Python. These versions of Windows 10 include a feature called the Windows Subsystem for Linux, which allows you to run a Linux environment directly in Windows, unmodified and without the overhead of a virtual machine.

For more information, see the Windows Subsystem for Linux Documentation article on the Microsoft website.

For instructions on how to enable the subsystem in Windows 10 and install a Linux distribution, see the Windows 10 Installation Guide.

You can also check out this presentation on YouTube by Sarah Cooley, one of the members of the WSL development team.

Once you have installed the Linux distribution of your choice, you can install Python 3 from a Bash console window, just as you would if you were running that Linux distribution natively.

Unlike most Unix systems and services, Windows does not include a system supported installation of Python. To make Python available, the CPython team has compiled Windows installers (MSI packages) with every release for many years. These installers are primarily intended to add a per-user installation of Python, with the core interpreter and library being used by a single user. The installer is also able to install for all users of a single machine, and a separate ZIP file is available for application-local distributions.

As specified in PEP 11, a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.8 supports Windows Vista and newer. If you require Windows XP support, then please install Python 3.4.

There are a number of different installers available for Windows, each with certain benefits and downsides.

The full installer contains all components and is the best option for developers using Python for any kind of project.

The Microsoft Store package is a simple installation of Python that is suitable for running scripts and packages, and using IDLE or other development environments. It requires Windows 10, but can be safely installed without corrupting other programs. It also provides many convenient commands for launching Python and its tools.

The nuget.org packages are lightweight installations intended for continuous integration systems. It can be used to build Python packages or run scripts, but is not updateable and has no user interface tools.

The embeddable package is a minimal package of Python suitable for embedding into a larger application

ANACONDA CLOUD

Python is a high-level programming language devised by Guido van Rossum & first released in 1991. It's the most popular coding language used by software developers to build, control, manage and for testing.

In Python, compiled code is stored with the file extension .py. For example –

hello.py

It is also an interpreter which executes Python programs. The python interpreter is called python.exe on Windows. To execute hello.py program, type –

python hello.py

Together with a list of Python packages, tools like editors, Python distributions include the Python interpreter. Anaconda is one of several Python distributions. Anaconda is a new distribution of the Python and R data science package. It was formerly known as Continuum Analytics. Anaconda has more than 100 new packages.

This work environment, Anaconda is used for scientific computing, data science, statistical analysis, and machine learning. The latest version of Anaconda 5.0.1 is released in October 2017.

The released version 5.0.1 addresses some minor bugs and adds useful features, such as updated R language support. All of these features weren't available in the original 5.0.0 release.

This package manager is also an environment manager, a Python distribution, and a collection of open source packages and contains more than 1000 R and Python Data Science Packages.

There's no big reason to switch to Anaconda if you are completely happy with you regular python. But some people like data scientists who are not full-time developers, find anaconda much useful as it simplifies a lot of common problems a beginner runs into.

Anaconda can help with –

- Installing Python on multiple platforms
- Separating out different environments
- Dealing with not having correct privileges and
- Getting up and running with specific packages and libraries

The free version of Anaconda distribution community edition can be downloaded directly from Anaconda's website. For the enterprise edition, one need professional support from Anaconda's sales team.

Conda treats Python the same as any other package, so it is easy to manage and update multiple installations.

Anaconda supports Python 2.7, 3.4, 3.5 and 3.6. The default is Python 2.7 or 3.6, depending on which installer you used:

- For the installers "Anaconda" and "Miniconda," the default is 2.7.
- For the installers "Anaconda 3" or "Miniconda 3," the default is 3.6.

Once downloaded the .exe file, run through the installer. Do accept the terms, and finish the installation. To check, close the browser and pull up the terminal.

Once the installation is complete, it should have automatically added that to the path. To test this, go ahead and type 'python'.

The version of python i.e. 3 will show and also the anaconda distribution will be seen. If you install the 4 versions of Anaconda, then all the packages, which are there in the packages, can also be imported easily.

- To check type import numpy or import matplotlib and run that.

JUPYTER NOTEBOOK

The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects. This article will walk you through how to set up Jupyter Notebooks on your local machine and how to start using it to do data science projects.

First, though: what is a "notebook"? A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media. This intuitive workflow promotes iterative and rapid development, making notebooks an increasingly popular choice at the heart of contemporary data science, analysis, and increasingly science at large.

Best of all, as part of the open source Project Jupyter, they are completely free. The Jupyter project is the successor to the earlier IPython Notebook, which was first published as a prototype in 2010. Although it is possible to use many different programming languages within Jupyter Notebooks, this article will focus on Python as it is the most common use case. (Among R users, R Studio tends to be a more popular choice).

To get the most out of this tutorial you should be familiar with programming, specifically Python and pandas specifically. That said, if you have experience with another language, the Python in this article shouldn't be too cryptic, and will still help you get Jupyter Notebooks set up locally. Jupyter Notebooks can also act as a flexible platform for getting to grips with pandas and even Python, as will become apparent in this article.

On Windows, you can run Jupyter via the shortcut Anaconda adds to your start menu, which will open a new tab in your default web browser. This isn't a notebook just yet, but don't panic! There's not much to it. This is the Notebook Dashboard, specifically designed for managing your Jupyter Notebooks. Think of it as the launchpad for exploring, editing and creating your notebooks.

Be aware that the dashboard will give you access only to the files and sub-folders contained within Jupyter's start-up directory; however, the start-up directory can be changed. It is also possible to start the dashboard on any system via the command prompt (or terminal on Unix systems) by entering the command jupyter notebook; in this case, the current working directory will be the start-up directory.

The astute reader may have noticed that the URL for the dashboard is something like http://localhost:8888/tree. Localhost is not a website, but indicates that the content is being served from your local machine: your own computer. Jupyter's Notebooks and dashboard are web apps, and Jupyter starts up a local Python server to serve these apps to your web browser, making it essentially platform independent and opening the door to easier sharing on the web.

The dashboard's interface is mostly self-explanatory — though we will come back to it briefly later. So what are we waiting for? Browse to the folder in which you would like to create your first notebook, click the "New" drop-down button in the top-right and select "Python 3" (or the version of your choice).

Ipynb File

It will be useful to understand what this file really is. Each. ipynb file is a text file that describes the contents of your notebook in a format called JSON. Each cell and its contents, including image attachments that have been converted into strings of text, is listed therein along with some metadata. You can edit this yourself — if you know what you are doing! — by selecting "Edit > Edit Notebook Metadata" from the menu bar in the notebook.

You can also view the contents of your notebook files by selecting "Edit" from the controls on the dashboard, but the keyword here is "can"; there's no reason other than curiosity to do so unless you really know what you are doing.

SPYDER INTEGRATED DEVELOPMENT ENVIRONMENT

t is always necessary to have interactive environments to create software applications and this fact becomes very important when you work in the fields of Data Science, engineering, and scientific research. The Python Spyder IDE has been created for the same purpose. In this article, you will be learning how to install and make use of Spyder or the Scientific Python and Development IDE.

An integrated development environment (IDE) facilitates computer programmers by integrating fundamental tools (e.g., code editor, compiler, and debugger) into a single software package. Users do not need to install the language's compiler/interpreter on their machines; an IDE provides the environment itself.

Spyder is a dedicated IDE for Python. It incorporates some useful features that make it a popular IDE.

Spyder is an open-source cross-platform IDE. The Python Spyder IDE is written completely in Python. It is designed by scientists and is exclusively for scientists, data analysts, and engineers. It is also known as the Scientific Python Development IDE and has a huge set of remarkable features which are discussed below.

Features of Spyder

Some of the remarkable features of Spyder are:

- Customizable Syntax Highlighting
- Availability of breakpoints (debugging and conditional breakpoints)
- Interactive execution which allows you to run line, file, cell, etc.
- Run configurations for working directory selections, command-line options, current/ dedicated/ external console, etc.
- Can clear variables automatically (or enter debugging)
- Navigation through cells, functions, blocks, etc. can be achieved through the Outline Explorer
- It provides real-time code introspection (The ability to examine what functions, keywords, and classes are, what they are doing and what information they contain)
- Automatic colon insertion after if, while, etc.
- Supports all the IPython magic commands
- Inline display for graphics produced using Matplotlib

It also provides features such as help, file explorer, find files, etc.

Writing code in Spyder becomes very easy with its multi-language code editor and a number of powerful tools. As mentioned earlier, the editor has features such as syntax highlighting, real-time analysis of code, style analysis, on-demand completion, etc. When you write your code, you will also notice that it gives a clear call stack for methods suggesting all the arguments that can be used along with that method.

Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It offers a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool

with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

Beyond its many built-in features, its abilities can be extended even further via its plugin system and API. Furthermore, Spyder can also be used as a PyQt5 extension library, allowing developers to build upon its functionality and embed its components, such as the interactive console, in their own PyQt software.

OPENCV – LIBRARY FOR COMPUTER VISION

Computer vision is a process by which we can understand the images and videos how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence. Computer-Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

Lots of pieces of information that are present in the original image can be obtained. Like in the above image there are two faces available and the person(I) in the images wearing a bracelet, watch, etc. so by the help of OpenCV we can get all these types of information from the original image.

APPLICATIONS OF OPENCV

There are lots of applications which are solved using OpenCV, some of them are listed below.

- face recognition
- Automated inspection and surveillance
- number of people – count (foot traffic in a mall, etc)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anamoly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching
- Video/image search and retrieval
- Robot and driver-less car navigation and control
- object recognition
- Medical image analysis
- Movies – 3D structure from motion

- TV Channels advertisement recognition

OpenCV Functionality

- Image/video I/O, processing, display (core, imgproc, highgui)
- Object/feature detection (objdetect, features2d, nonfree)
- Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo, video, superres)
- Machine learning & clustering (ml, flann)
- CUDA acceleration (gpu)