AUTONOMOUS VECHILE LANE DETECTION

A PROJECT REPORT

Submitted by

DHEV DARSHAN R [Reg No: RA1911003010997]
ILAN SURYA [Reg No: RA1911003010954]

Under the guidance of

Mrs. G. MALARSELVI

Assistant Professor, Department of Computing Technologies

in partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

MAY 2023

SRM

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled "**AUTONOMOUS VEHICLES LANE DETECTION**" is the bonafide work of **DHEV DARSHAN R [RA1911003010997]** and **ILAN SURYA [RA1911003010954]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Mrs.G.MALARSELVI
SUPERVISOR
Assistant Professor
Department of Computing Technologies

Dr. SUBALALITHA C.N.
PANEL HEAD
Associate Professor
Department of Computing Technologies

Department of Computing Technologies
Kattankulathur
603 203
SRM IST

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Professor
Department of Computing Technologies

17/5/23

INTERNAL EXAMINER

17/5/23

EXTERNAL EXAMINER

SRM

Department of Computing Technologies

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course	: B.Tech in Computer Science and Engineering
Student Names	: DHEV DARSHAN R , ILAN SURYA
Registration Number	: RA1911003010997, RA1911003010954
Title of Work	: AUTONOMOUS VEHICLE LANE DETECTION.

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.
We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

Student 1 Signature: R. Dhev Darshan
Student 2 Signature: Ilan Surya

Date: 17/05/2023

If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalatha,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinator, **Mrs G. Malarselvi**, Assistant Professor, Panel Head, **Dr. Subalalitha C.N.**, Associate Professor and Panel Members, **Dr. R. Anita,** Assistant Professor and **Dr. R. Srinivasan,** Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support..

We register our immeasurable thanks to our Faculty Advisor, **Dr. M. KOWSIGAN**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Mrs. G. Malarselvi**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for problem solving and making a difference in the world has always been inspiring.

We sincerely thank the Department of Computing Technologies staff, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support, and encouragement.

DHEV DARSHAN R [RA1911003010997]

ILAN SURYA [RA1911003010954]

TABLE OF CONTENTS

ABSTRACT

Self-driving vehicles are not only a reality today, but this technology also provides a preview of what complicated technology can look like in the future. To build the greatest autonomous cars possible, many professionals from diverse industries have collaborated. Lane detection is already a standard function in automobiles, and training models for it needs a lot of real-world data. This information must take into account all possible weather conditions, road topographies, driver behaviours, and vulnerable road users in order to construct completely autonomous cars. It is difficult to simulate every probable circumstance in the actual world, and the majority of firms lack the funding necessary to test their vehicles across lengthy distances. Utilising vehicle racing games, which provide a wealth of data, is one way to train models more effectively.real-world privacy and security issues by simulating genuine driving conditions. With the eventual objective of employing this system to create data for training models in self-driving cars, the suggested method involves developing a system for recognising road lanes in a car racing game using Python and OpenCV. To enable the model to focus on recognising lanes, the system will employ Canny edge detection to recognise the margins of the lane and a masking function to hide undesirable elements in pictures like trees, rocks, and power lines. The game's lanes are identified and drawn using the Hough Transform.The suggested approach successfully locates lanes in a racing automobile operating in real-time. This strategy enables the development offering a variety of driving scenarios that could be challenging or costly to reproduce in the real world, while also giving a significant volume of data for lane detecting systems in autonomous cars in a cost-effective and efficient manner.

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
PC	Personal Computer
AI	Artificial Intelligence
ML	Machine Learning
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
RGB	Red Green Blue
BGR	Blue Green Red
GPU	Graphics Processing Unit
RAM	Random-Access Memory
ROS	Robotic Operating System
ROI	Region Of Interst
CNN	Convolutional Neural Network
CADP	Car Accident Detection and Prediction
VIRAT	Video Image Retrieval and Analysis Tool
OpenCV	Open source Computer Vision library

CHAPTER 1

INTRODUCTION

1.1. General

Every day, people make hundreds of decisions, most of which are based on information we gather from the world around us. The majority of this perception when it comes to driving is visual. Autonomous vehicles that are capable of recognising items around them and making quick judgements in response to these impulses are self-driving automobiles. As a result, there are several computer vision-based issues with autonomous cars. Examples include locating things like traffic lights, people, other vehicles, and lanes. This essay focuses on lane detection, one of the prerequisites for vehicle motion planning that is most essential.

With the advancement of autonomous driving technology, lane detection has become an important study topic. At the same time, lane detection serves a crucial role in the lane departure warning system. The first stage in teaching a computer to comprehend the road situation is lane recognition. Several lane characteristics exist that we may use. The parallel lane lines are either white or yellow. The majority of them are continuous and have a uniform width all along the route. Finding the white or yellow horizontal marks on the painted road's surface that indicate the limits of the lanes is known as lane detection. Utilising temporal coherence and the previously identified lane markers, lane tracking modifies itself in accordance with the motion model.

When driving on a highway with a homogeneous road texture and seeing very clear and straight lane lines for a short distance ahead. When driving on an urban or inner-city road, obstructions from parked and moving cars as well as tree shadows, bright lighting, changing weather, different times of day, broken lanes, and common lane shapes provide obstacles. Lane detection appears to be a challenging issue that can be resolved with computer vision.

Before lane detection using the Hough Transform, the first processing step in image pre-processing is the conversion of an RGB picture to a grayscale one. A review of the literature on autonomous vehicle technology is done, and several autonomous vehicle techniques are taken into account. The selected method should be effective, have a high processing speed, low complexity, good computational efficiency, need little memory, and be simple to apply in light of the aforementioned challenges and downsides. The method should also be adaptable for future growth and improvement.

1.2. Purpose

Intelligent vehicles work together with smart infrastructure in intelligent transportation systems to create a safer environment and improve traffic flow.Although enhancing safety by fully or partially automating driving responsibilities is a more compelling argument for the development of intelligent cars.Road detection was one of these duties, and it played a significant part in driving assistance systems, which give information such lane structure and vehicle location in relation to the lane. However, the demand for safety is the most compelling argument for adding autonomous capabilities to automobiles. In Malaysia and other Asian nations, vehicle accidents continue to be the largest source of accident fatalities and injuries, resulting in tens of thousands of fatalities and millions of injuries each year. Most of these traffic-related fatalities and injuries take place

The majority of these traffic-related fatalities and injuries take place on the country's roadways. With an average of 4.5 fatalities per 10,000 registered cars, Malaysia is ranked 30th among the nations with the highest number of fatal road accidents by the UN [1]. A device that can alert the motorist to the risk therefore has the potential to save a sizable number of lives. One of the key technologies utilised in these is computer vision, which has developed into a potent instrument for perceiving the surroundings and has been extensively employed by the intelligent transport systems (ITS) in numerous applications. Many suggested systems'[2] lane recognition is accomplished by localising certain primitives, such painted road surfaces' surface road markers.

Although this constraint makes the detection process easier, it can still be complicated in two ways: The presence of other cars in the same lane somewhat obscured the road markings in front of the car, as did the shadows cast by nearby trees, buildings, etc. This work provides a vision-based method that can detect and track structured road borders (lane markers painted or unpainted) with a minor curvature in real time and is sufficiently resilient in the face of shadowy circumstances. The edges of the 82 are fitted with parallel hyperbola pairs to identify road borders.After using edge detection and Hough transformation, you may view metadata, citations, and related publications at core.ac.uk, which is offered to you by CORE and The International Islamic University Malaysia Repository channel.

This constraint makes the identification procedure easier, but it might be complicated in two situations: the prese transform. The intended route for the car is either a flat, straight road or one with a gentle curve.

The simulation framework for autonomous cars and real-time road lane recognition and tracking is presented in this study. The study is divided into five sections, the first of which is devoted to research on the detection of painted lines and road limits. The core scenario, which was developed using Pro-SiVIC, a simulator of embedded sensors, dynamic infrastructure, and automobiles, is described in the second part. The third section focuses primarily on algorithms for detecting road lanes, using the Canny edge detector and Hough Transform to initialise a suggested approach that uses an adaptive region of interest, the least-square method, and a Kalman filter to forecast the parameters of the next position of the road boundaries and lines. The control technique based on road lane centre and fuzzy logic is described in the fourth part.

1.3. Motivation

On a daily basis, humans make hundreds of judgements, most of which are based on perceptions gained from the environment. When it comes to driving, the majority of this sense is visual. Autonomous vehicles, such as self-driving automobiles, should be able to recognise the things around them and make quick judgements to respond to these urges. As a result, autonomous cars have a number of computer vision-based issues. Road signs, traffic lights, pedestrians, other vehicles, and lanes are a few examples of items that can be identified.

One of the most important prerequisites for vehicle motion planning, lane detection, is the subject of this research.

The essential technology and most difficult issue in autonomous driving is motion planning. Motion planning, according to The Robotics Institute at the University of Michigan, is the act of breaking down a desired movement job into discrete movements that adhere to movement limitations and may even optimise a particular feature of the movement. [1]. In the early stages of motion planning, self-driving cars look for lanes, then follow the trajectory of the lines, look for signs on the lanes, and finally look for obstacles to identify closed lanes. The primary goal of the study is to determine which detection techniques are most appropriate for the job of lane line and lane colour identification.

1.4. Machine Learning

Machine Learning (ML), though often confused and interchangeably used with Artificial Intelligence, is a subset of artificial intelligence and is the science of developing machine intelligent algorithms that help the machine learn from existing and past data. The algorithms achieve this by identifying and learning the various relationships between individual features of the data, looking for common patterns, responding to different situations outside of their programming restrictions and makes predictions accordingly.

Machine Learning can be broadly classified into three categories (figure 1.4.1) based on the kind of data they use to learn and the type of outputs produced by them. They are:

- Supervised Learning: In this type of learning, the datasets that the machine utilises to learn are entirely structured and labelled. The datasets have a set of input features and their corresponding outputs and are given to the ML algorithm as inputs for training. While training, the model learns by mapping its predictions to the true predictions present in the dataset, evaluates its performance, and automatically updates its learning curve to achieve better scores of the performance metrics. For example, classification and regression tasks such as image classification and commodity price prediction respectively.

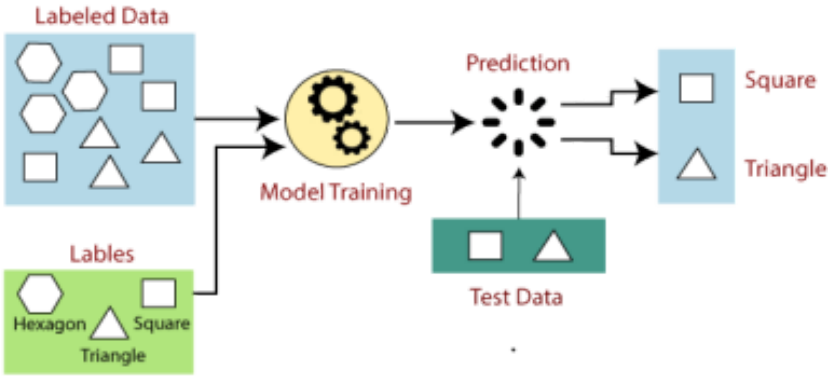


Figure 1.4.1: Supervised Learning

- Unsupervised Learning: In this type of learning, the datasets are not mapped with their outputs. The models are trained on a dataset consisting of several features and are expected to produce outputs by learning and identifying common patterns without conforming to a certain "correct answer". For example, clustering algorithms used for customer segmentation, DNA pattern recognition and grouping in evolutionary.

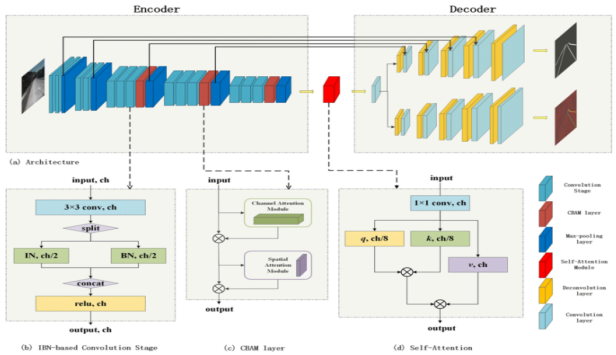


Fig 1.4.2: Unsupervised Learning

1.5. Computer Vision

Traditional computer vision (CV) methods were used to interpret visual data for a very long time before the invention of deep learning algorithms. Since the widespread use of information extraction from visual data approximately 60 years ago, the field has been around. The guy who is frequently credited as being the inventor of CV, Lawrence Gilman Roberts,

originally experimented with building and displaying a three-dimensional array of solid objects from a two-dimensional image.

"These assumptions enable a computer to obtain reasonable, three-dimensional description from the edge formation in a photograph by means of topological mathematical process," he writes in his 1965 article. [2]. Academics were subsequently able to categorise CV into additional fields as a result of the improvement in picture quality; the most often used fields for lane detection now are image segmentation and edge detection.

A subset of artificial intelligence known as computer vision contains techniques created to allow machines to analyse visual data such as pictures, movies, and information from other visual inputs in order to extract useful information from them. These techniques assist computers in learning the context and information necessary to distinguish between objects, comprehend the many elements and characteristics included in a picture, and interpret sequences of slices of 3D images and movies.

Convolution neural networks (CNNs), a type of deep learning model used in computer vision, are used to learn, evaluate, and interpret visual input similarly to humans. Computer vision is mostly pattern recognition at its most fundamental level. Therefore, feeding a computer vision algorithm a huge amount of tagged visual data and then using various approaches and strategies to aid the machine in learning these patterns is the most typical strategy to boost its efficiency. Based on its uses, computer vision may be further divided into a number of fields. Just a few examples include scene reconstruction, object identification, object recognition, event detection, motion estimates, 3D pose estimation, 3D scene modelling, and picture restoration.

John Canny first introduced the Canny edge detection technique in his paper “A computational approach to Edge detection” [4]. The process for the Canny Algorithm can be shown with the following flow chart:

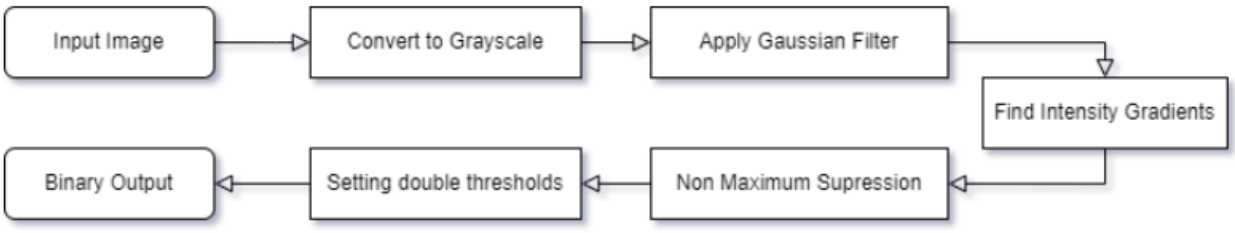


Fig 1.5.1 : computer vision

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.

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, Nonmaximum 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.

.

CHAPTER 2

LITERATURE REVIEW

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 edg

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

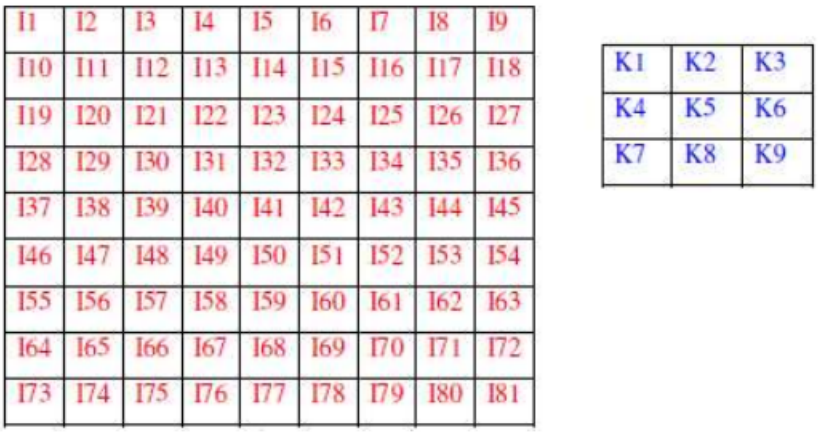


Figure 2.1.1: Convolution operation

2.2. 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.

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

Fig 2.2.1 : Gaussian filtering to remove noise

2.3. 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.

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 and G masks shown below each one estimates gradient x direction and y direction respectively:

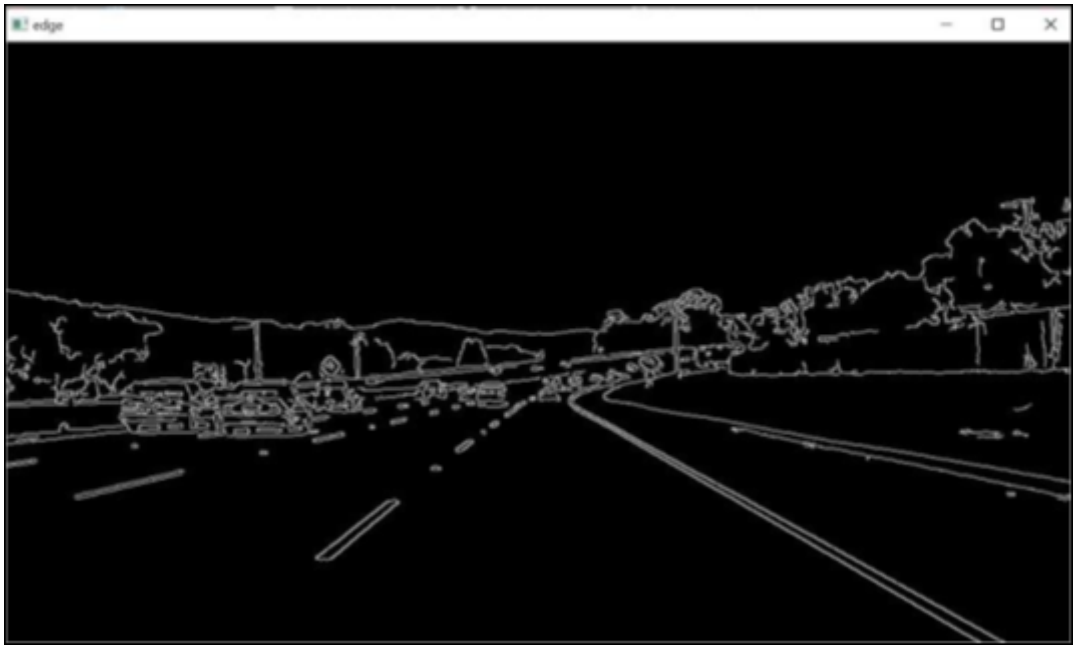


fig 2.3.1 : Sobel Operator

fig 2.3.2 : 2-D spatial gradient

2.4. 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 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.

The 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.

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 degre

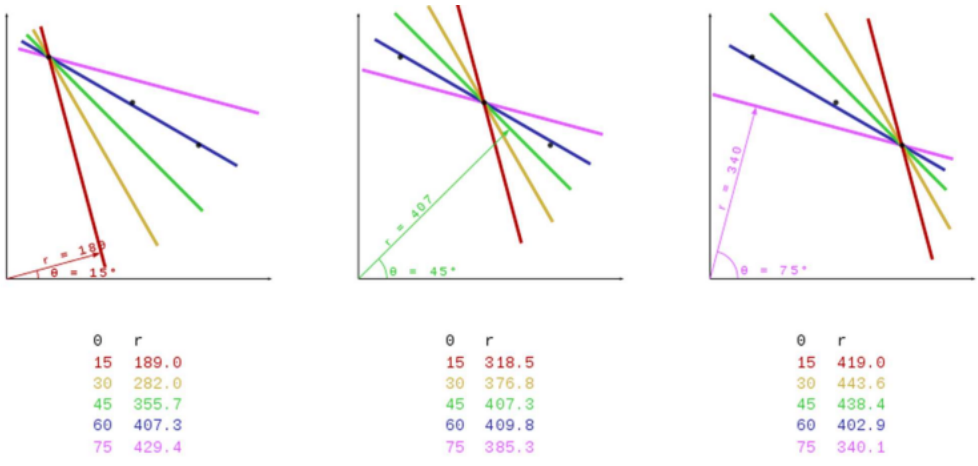


Figure 2.4.1: Tracing the edge in the image using $\boldsymbol{\theta}$(angle)

2.4.2 Non maximum Suppression

After the edge directions are known, non-maximum 16 | P a g e 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).

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.

Fig2.4.2 : Non maximum Suppression

2.5. Sample Input and Output of Canny Edge Detection Algorithm:

Hausdorff distance is used to measure the distance between two subsets of a metric space. It is most commonly used as a performance metric for comparing segmented images with the original dataset. A small magnitude indicates higher precision of segmented images. In a recent study [10, 11] detecting crowd group structures, Hausdorff distance was used as a measure of inter-group closeness and locomotion similarity that takes into consideration the variables of proximity and velocity, having shown to be used for crowd movement and trajectory analyze

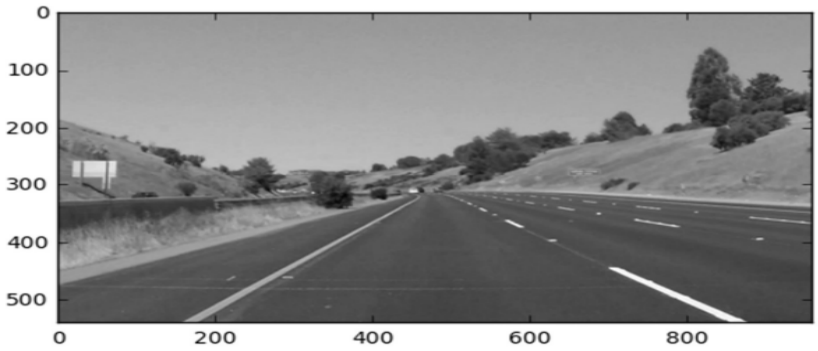


Fig 2.5.1: Sample Input and Output of Canny Edge Detection Algorithm

Fig 2.5.2 : higher precision of segmented images

CHAPTER 3

PROPOSED 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

3.1. Straight Lane Detection

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 × 512 × 8 ≈ 2 × 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 28 | P a g e 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.

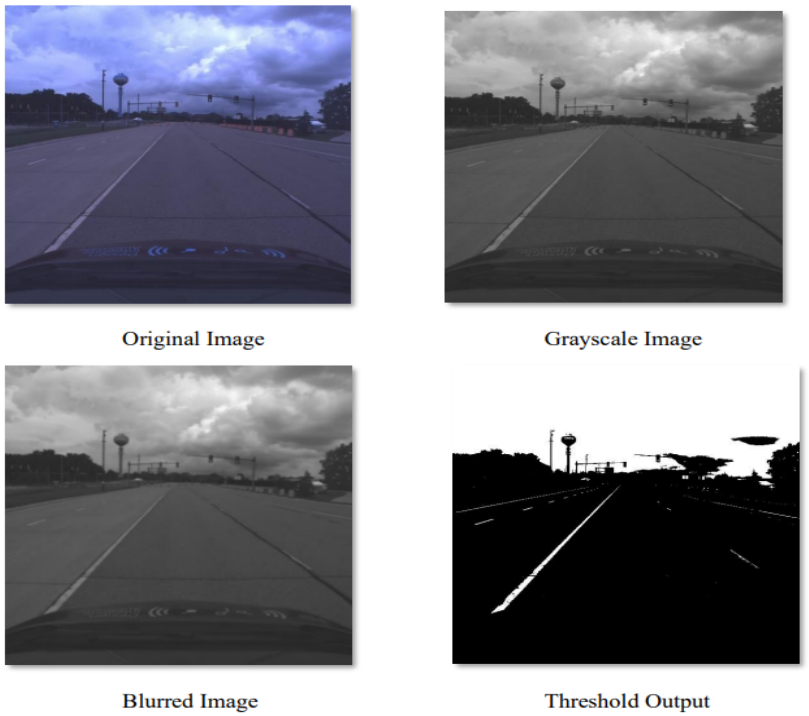


Fig 3.1.1: Straight Lane Detection

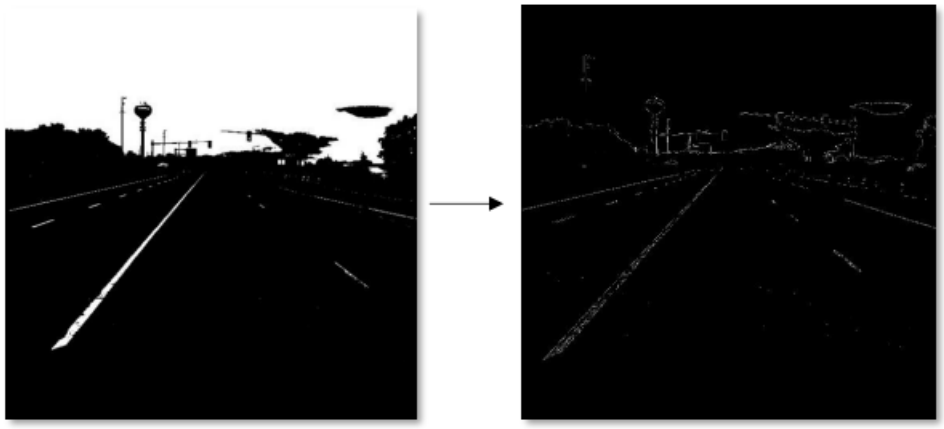


Fig 3.1.2: Application of Canny and Sobel Methods

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.

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.

The image obtained after applying the Canny and Sobel filters shows all edges needed in the picture for lane detection. The next important step is to identify our region of interest to avoid unnecessary edges from being detected. We thus, figure out a smaller area of interest. We perform this by calculating the vanishing point of the image and then forming a trapezoid with lines parallel to the location of the vanishing point. This concept of vanishing point was first introduced by Filippo Brunelleschi in the early 15th century and is one of the predominant laws of image spaces [9]. For our purposes, we manually calculate the vanishing point in our scenario since this is a challenging computer vision problem on its own.

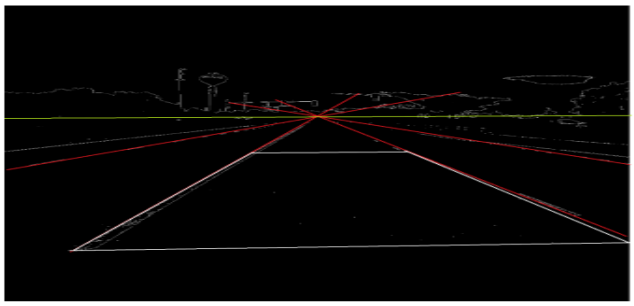


Fig 3.1.3: Vanishing Point and Region of Interest (ROI)

3.2. Hough Line Transform

After identifying the ROI, we use Hough Line Transform as a feature extraction technique, to extract the lines we are interested in which are the lanes on the road in our case [11]. We utilized Probabilistic Hough Transform for our use case to obtain the pixels where lanes are located inside the ROI [12]. Fig. 2.4 below shows the final output of our ROS pipeline for Straight Lane Detection.

A combination of two state-of-the-art convolution neural networks pre-trained on the ImageNet dataset, the EfficientNet-B7 and the ResNet-50, were used to extract features from the frames of the video clips. Since the EfficientNet-B7 has proved to be more accurate and efficient as compared to the ResNet-50, it is used in the beginning to extract the initial, more important features. The architectural details of the EfficientNet-B7 and the ResNet-50 are depicted in figure 3.2.1 and 3.2.2 respectively.

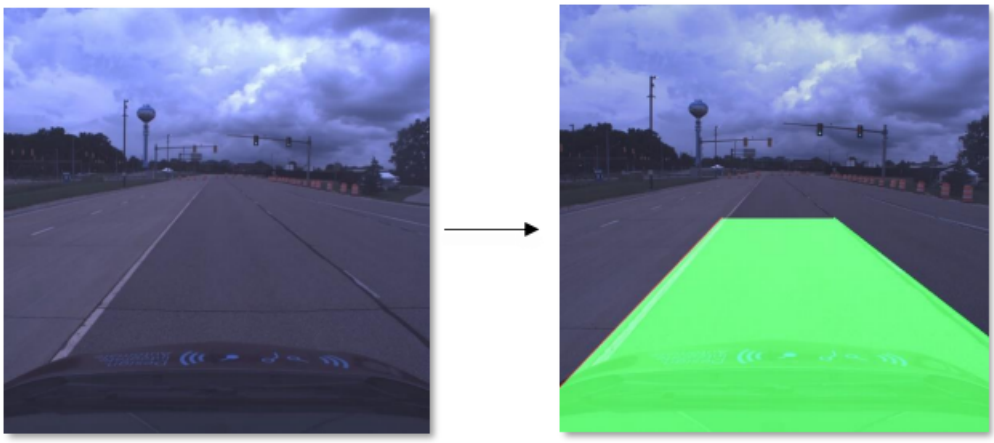


fig3.2.1.: Architectural Curved Lane Detection

The Curved Lane detection problem is more difficult given the complexity of detecting curved lines. Several methods have been used for curved lane detection using computer vision. The method we choose for the thesis is called a Sliding Histogram Method for Lane Detection, He et al. visualized in Fig. 2.5, is the process we used for Curved Lane Detection.

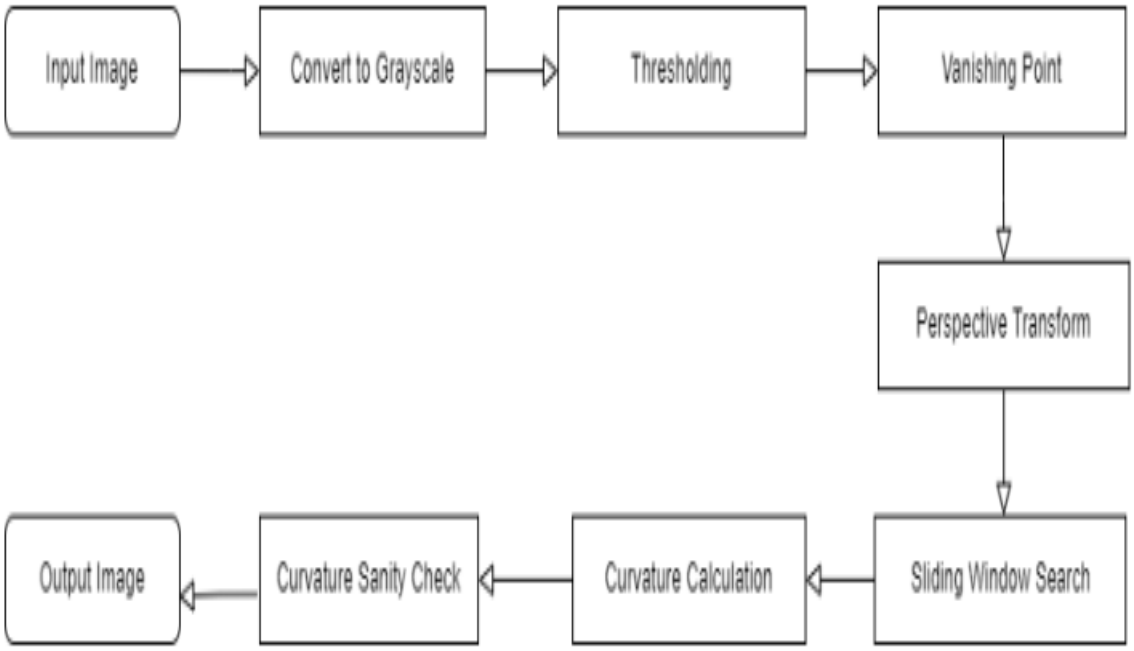


Fig 3.2.2.: Thresholding

Thresholding is a common technique utilized in most computer vision systems, wherein a given threshold is set for pixels, and a pixel's RGB values above the threshold are converted to white and the rest to black. This method of thresholding is often referred to as Binary thresholding and helps reduce noise in the image. An advanced method of thresholding called Adaptive thresholding was utilized in our process. Adaptive thresholding uses thresholding based on small regions around the pixel being threshold thus ensuring more uniform thresholding even with nonuniform lighting conditions [14]. We used OpenCV's open-source implementation for this calculation.

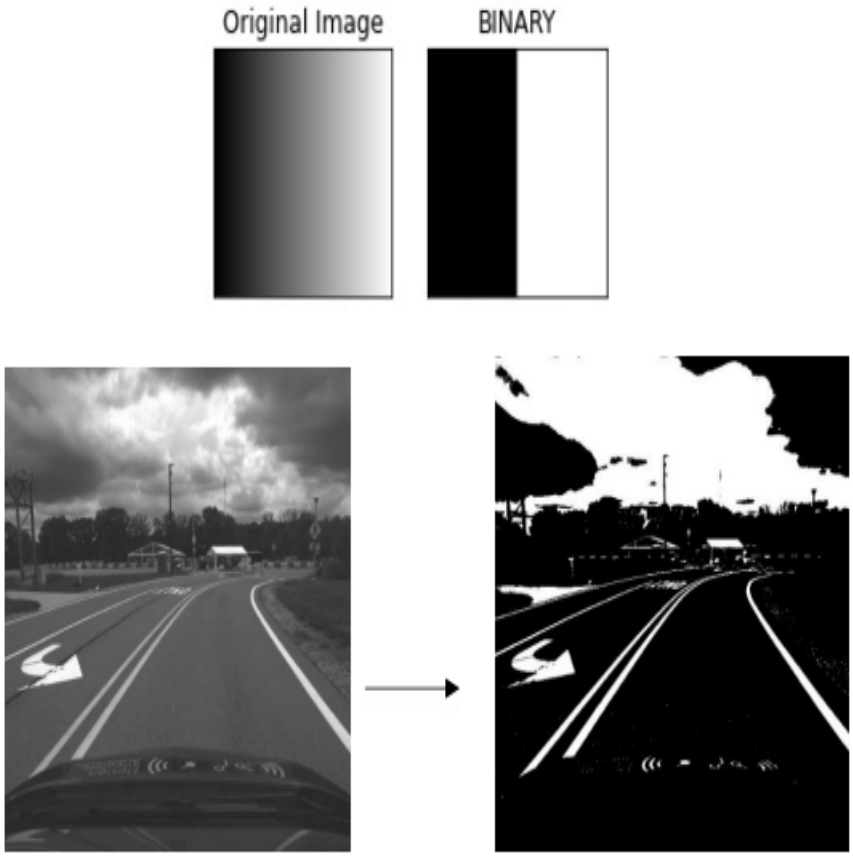


fig 3.2.3: Adaptive Thresholding

Furthermore, the ResNet-50 model, like the EfficientNet, is loaded with the ImageNet weights. Prior to sending the images and features as the inputs to the EfficientNet and the ResNet respectively, the inputs must be pre-processed. Thus, layers to pre-process the inputs are defined and introduced before the models. Since the pre-processing layers and methods are included in the main model architecture for the EfficientNet as the Rescaling layer, the preprocess_input layer for this model is essentially a pass-through function. However, for the ResNet, the preprocess_input layer converts the RGB images to BGR and performs zero centre normalization on the colour channels without scaling.

Pooling is used to reduce the dimensions of the features produced by the network. By reducing the dimensions and the features, it reduces the number of parameters to be learnt by the model during training, thereby reducing the computation cost of training the network, and increases its efficiency. There are two types of pooling that can be performed on the output of features:

- Max Pooling: in this type of pooling, it returns the maximum value of the pixels that are covered by the kernel. Since the maximum value among a set of pixels is towards the lighter end of the colour spectrum, Max Pooling selects the brighter and lighter-coloured pixels of the image. This can be used for tasks where the background of the object to be segmented is dark or if this object needs to be distinctively identified.

- Average Pooling: in this type of pooling, it computes and returns the average of the pixel values that are covered by the kernel. In general, Average Pooling smoothens out the image, and prevents the sharp features of the images such as edges, lines, and grains from being retained.

The extracted features of the EfficientNet are reshaped to an appropriate size containing three colour channels, thus having the input shape compatible with that of the ResNet and simultaneously preventing the loss or transformation of the extracted features by using additional Convolution or Pooling layers. Both Max-Pooling and Average Pooling are first applied across the extracted features of the ResNet to obtain a total of 2048 features for each frame. The pooling producing the higher score of accuracy during training is then retained.

Below shows the next step in the process, i.e., detecting the vanishing point and identifying the ROI as we did in Straight Lane Detection

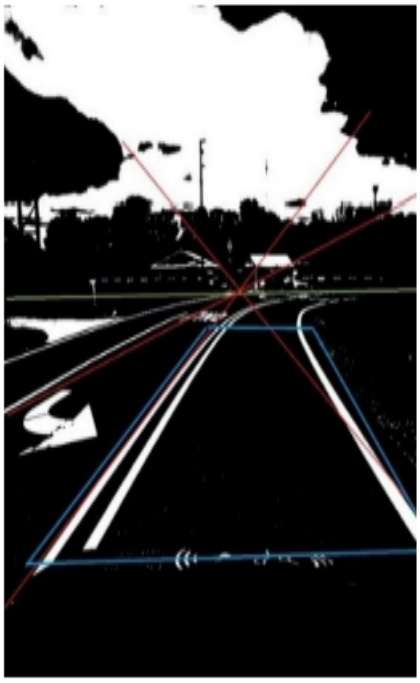


Fig 3.2.4.: Perspective Transform

3.3. Sliding Window Search

The next step in the process is developing a histogram of the pixels found in our ROI and looking at the peaks of the histogram. Fig. 2.9 shows 3 lane lines, and the line histogram shows two peaks on the left and the right respectively. This gives us two arrays with pixels and their locations

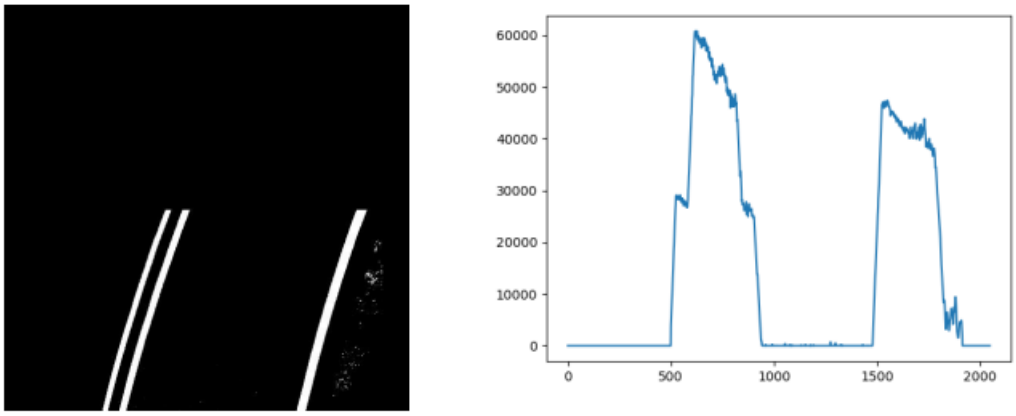


Fig 3.3.1.: Curvature Calculation

The final step in the detection process is to create windows of dx width and slide them across these arrays to determine the shift in curvature. We then fit a second order polynomial curve on these pixels and the curvature value is calculated using the formula for radius of curvature from a curve.The equation of curvature for our sample image and diagram for radius of curvature generated using Desmos .Thus, we obtain the curves that fit these pixels, and these curves give the direction the road is turning towards

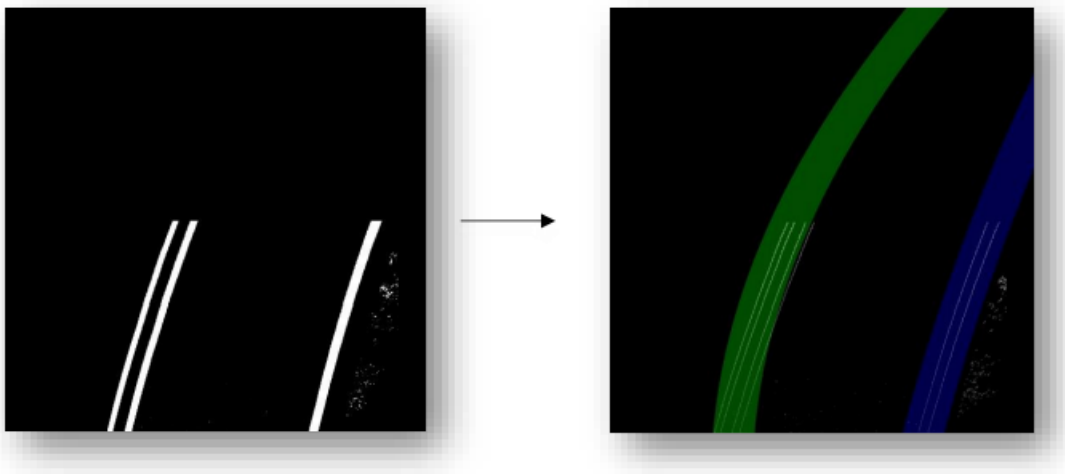


Fig 3.3.2.: Lane Type Detection

The lane type detection can be complex to solve using Deep learning since it would require accurate labeling to ensure correct detection, and as explained earlier this can be an expensive process. To our knowledge, there are no existing deep learning models which classify the lane marking type, making computer vision an essential method for this process. We utilized the output for Straight Lane detection pipeline as input for this subsystem. The model density as shown in the histogram below shows a vast difference in the number of pixels between Solid and Dashed Lines.

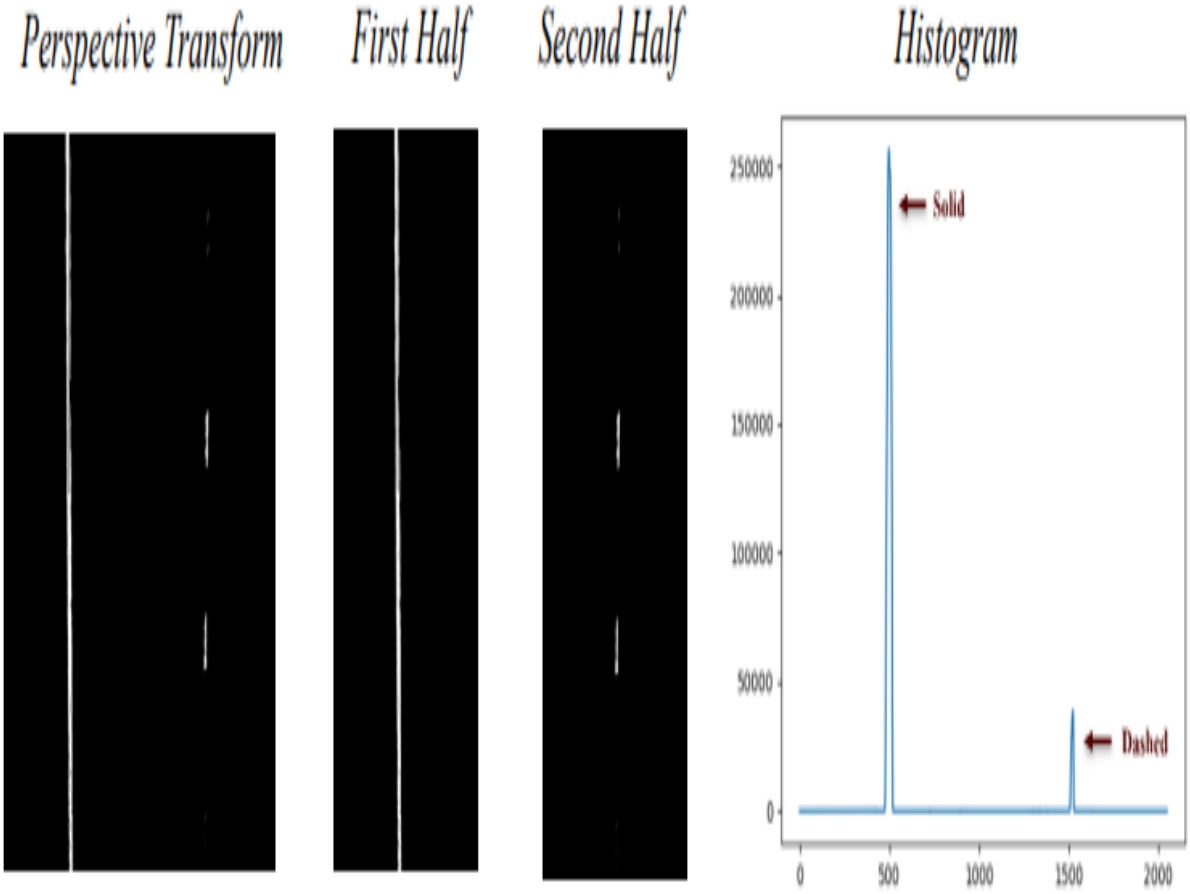


Fig 3.3.3: Lane Color Detection

Lane color detection is not typically solved using computer vision as there are other methods to determine where traffic is flowing in your surroundings using mapping and localization.

We wanted to try to solve this problem with computer vision itself. The main challenge in identifying the lane color was caused by constantly changing light conditions. To account for this, we used a range of colors for both the yellow and the white lanes to ensure accuracy. The steps of the process for Lane Color Detection are shown in Fig 2.13 below. We first apply gaussian blur to reduce pixelation after threshold masking. The next step in the process is to apply threshold masks in the yellow and white ranges. Mask values can typically be represented as RGB values for the respective pixels, mapped from 0 to 255.The range we selected for the yellow mask was from [130, 60, 200] to [240, 240, 255]. Values for the White Mask ranged from [0, 0, 185] to [255, 70, 255]. As seen in Fig. 2.13, the yellow and white masks still have pixelation, which gets removed once we combine both masks, thus giving us access to nothing but the yellow and white pixels in the image. We then apply an ROI mask and obtain the output image showing yellow and white lanes. These can be filtered later to distinguish between the two colors.

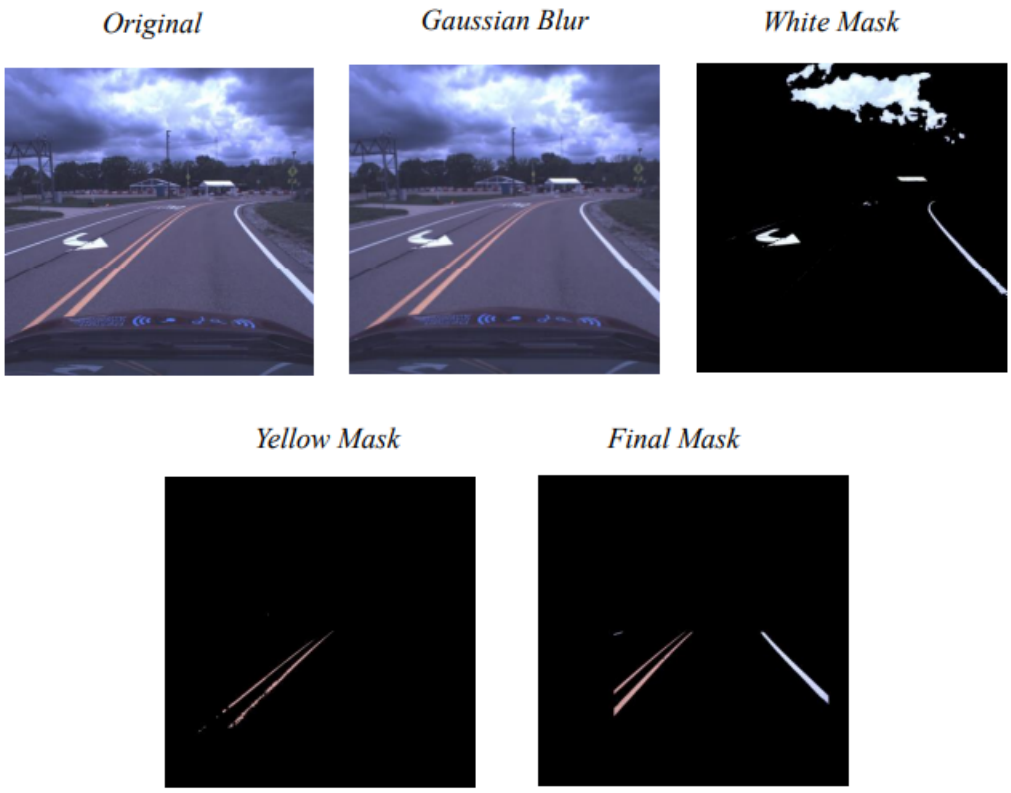


Fig 3.3.4.: Distortion correction

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.

- 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.
- 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.

- 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=r2) 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.
- 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=axx+ayy+d for which we can use a 3D Hough space corresponding to ax, ay 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 ax and ay 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 24 | P a g e 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

The average prediction of the model and the ground truth values. Whereas variance is the model's sensitivity to and variation in predictions for different portions of the training dataset. There is a trade-off between these two parameters when the model is required to simultaneously minimize both.

3.3.5.: Important Parameters

True Positive (TP): Detects True when the actual value is True

True Negative (TN): Detects False when the actual value is False

False Positive (FN): Detects True when the actual value is False

False Negative (FN): Detects False when the actual value is True\

IoU is commonly used as a similarity index for object detection problems in Computer Vision. As the name suggests we take intersection and unions of the expected output and the actual output and divide to get the score (Eqn. 3.3). This score ensures that the model is attributed for correct identification, while penalizing for incorrect identification.

3.4. Discussion

Computer Vision detectors, i.e., the Straight Lane Detector and the Curved Lane Detector performed surprisingly well in most test cases in our dataset. The performance of these methods however is dependent on various aspects of driving conditions. Some edge cases that we encountered where Computer Vision didn't perform well were as follows: 1. Presence of Shadows: Encountering shadows tend to mess up the CV algorithms we used since, shadows can be mistaken for gradient changes, and thus get detected as edges. This causes discrepancies in the model, thus giving unexpected results. There are however some algorithms for detection and removal of shadows that could be used for preprocessing [21]. 2. Presence of Road Sign markings or Crosswalks: We also noticed that the Sliding Histogram method tends to mess up when crosswalks are encountered, since these crosswalks are the same color as the lane lines. 3. Double Yellow/White Lines: The histogram method can falsely detect the lane as being present between the two line of the left rather than the entire lane. We intend to resolve this issue by specifying a minimum distance between the detected lane lines,

which should be wider than the car itself. 4. Unclear Lane Markings: Some edge cases included lane marking that were either worn out or covered by tar thus confusing the Computer Vision models into not detecting those as lines. The Deep Learning model we utilized also performed surprisingly well given the fact that it was not specifically trained on our dataset. We intend to perform some transfer learning on these models to make them more specific for our use case. The drawbacks discussed above for the Computer Vision methods have been solved using Deep Leaning methods, given that you can label objects, essentially reducing the noise the model looks at to begin with.

We collected samples from a few different categories to make a test set shown in Fig. 3.1.

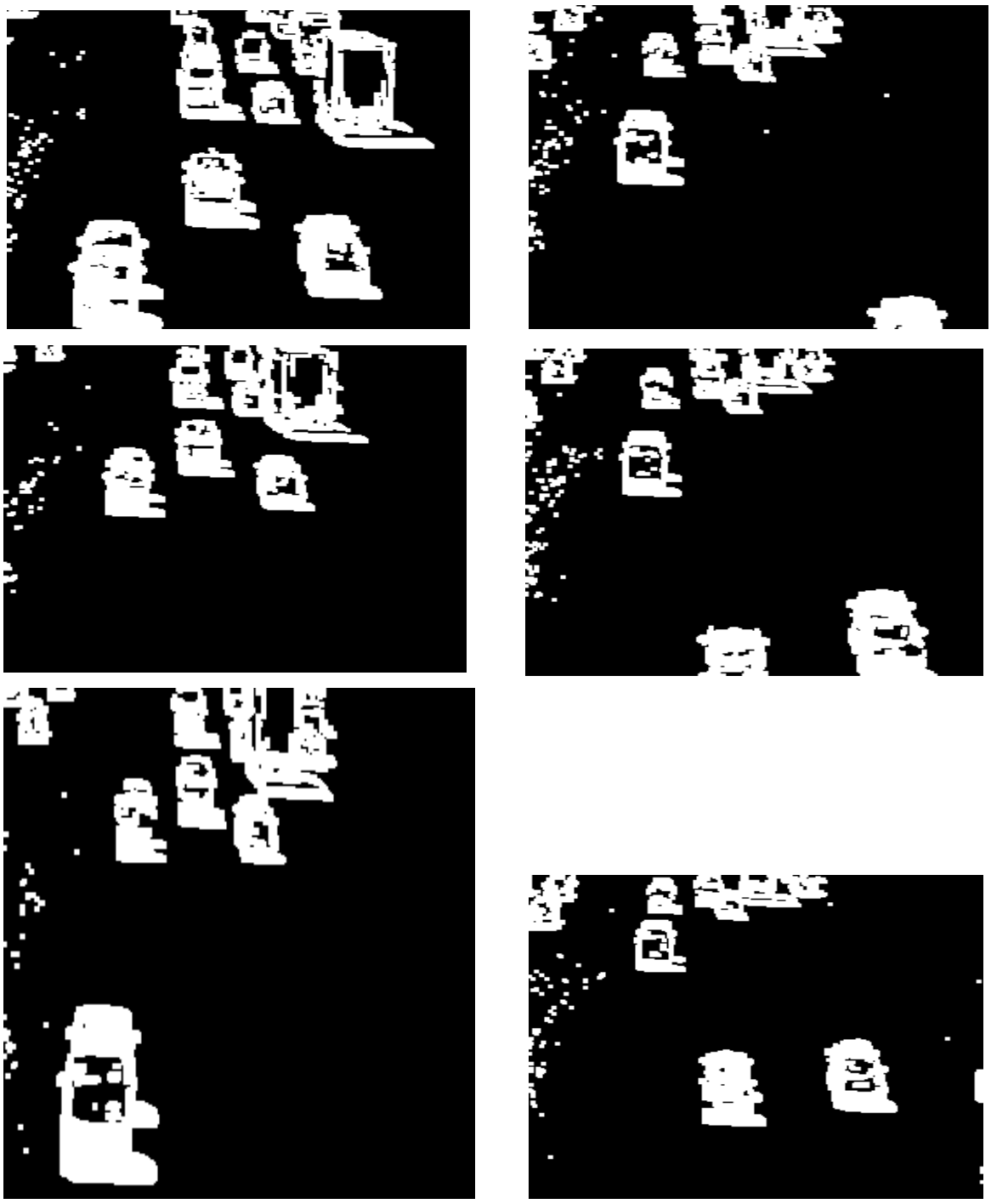


Fig 3.4.1.: Frames Post Thresholding and Dilation

3.5. 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 32 | P a g e 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.

CHAPTER 4

RESULTS

The outcomes of the suggested models and approaches to locate and recognise instances of reckless driving from video footage are covered in this section. We built the project utilising the Google Colab Nvidia K80 / T4 GPU (0.82 - 1.59 GHz), with a minimum of 12GB of RAM, and neural networks built using TensorFlow Keras [15] [16] layers. Additionally, OpenCV's computer vision techniques were used to develop the detection module [17]. First, the model's performance is compared to its numerous designs and parameter initializations, and then comparisons are made between the model's performance, accuracy, efficiency, and model complexity using pre-existing architectures.

On the dataset, the RNN classifier was initially trained without augmentation or parameter tuning. The weights were set to the available default values in the constructors of the corresponding layers classes. The model, however, appeared to be severely overfitting the data (figure 4.1). Given that the dataset for this project was manually constructed, it was possible that the limited and imbalanced dataset was to blame for the model's poor performance.

The model was trained on a subset of the UCF-101 dataset in order to evaluate whether the issue was with the model or the dataset. Realistic action videos are used in the UCF-101 dataset for action recognition. There are five different activity categories that may be separated: athletics, human-object interaction, bodily motion, human-human interaction, and performing musical instruments. The model was trained using 594 films, which were used for training, and 224 videos, which were used for testing, from a subset sampled with five distinct actions. Figure 4.2 shows that the model overfit this dataset, suggesting that the model's parameters needed to be adjusted.

We utilised Rand Augment using the settings given forth in the original research [9] to convert and enlarge the original dataset in order to address the issue of having a small-scale, imbalanced dataset. In training, data augmentation minimises overfitting. For the SimpleRNN layers with an L2

0.01 , the optimizer for Adam. The GlorotNormal [18] and Orthogonal distributions were chosen as the kernel and recurrent initializers, respectively, since they were shown to be the most effective in reducing the vanishing gradient problem when compared to the Lecun, random, and GlorotUniform distributions. The dataset was divided into a training and validation set at a percentage of 20%, and training took place over 40 iterations with 10 steps each. The accuracy and loss curves for training and validation are displayed against the epochs in Figure 4.3. The DriftNet [5] model architecture, the most recent state-of-the-art model architecture presented for the same goal, has been shown to be more effective and accurate than previous models with training and validation accuracy of 93.75% and 92.5%, respectively, using 4.69M trainable parameters.

0.02 ConvLSTM, DenseNet, and C3D models, among others, were employed to identify aggressive driving behaviours. With 144,026 trainable parameters, our model achieved training and validation accuracy of 90.62% and 90.17%, respectively. Information on the many parameters tweaked to improve the performance of the model is provided

0.03 With a modification in the mix and strength of the RandAugment transformations used before and during feature extraction, different accuracy and loss curves may be shown. Our architecture is computationally less complex than other networks', which lowers computation costs and eliminates the need for high-performance computer servers despite the fact that it performs somewhat worse than other networks. Additionally, the fewer epochs, smaller batches of training data, and careless dataset construction can all be blamed for the noisy training.

0.04 The efficacy of this approach to recognise the cars depends on the model's capacity to segment and detect the vehicles engaged in a reckless driving event utilising frame markers, moving object identification, and contour processing.

0.05 Straight Lane Detector and Curved Lane Detector are two computer vision detectors. performed surprisingly well across the majority of our dataset's test situations. These strategies' effectiveness, however, depends on a number of driving-related factors. some unusual situations that we The following are examples of situations when computer vision didn't function well:

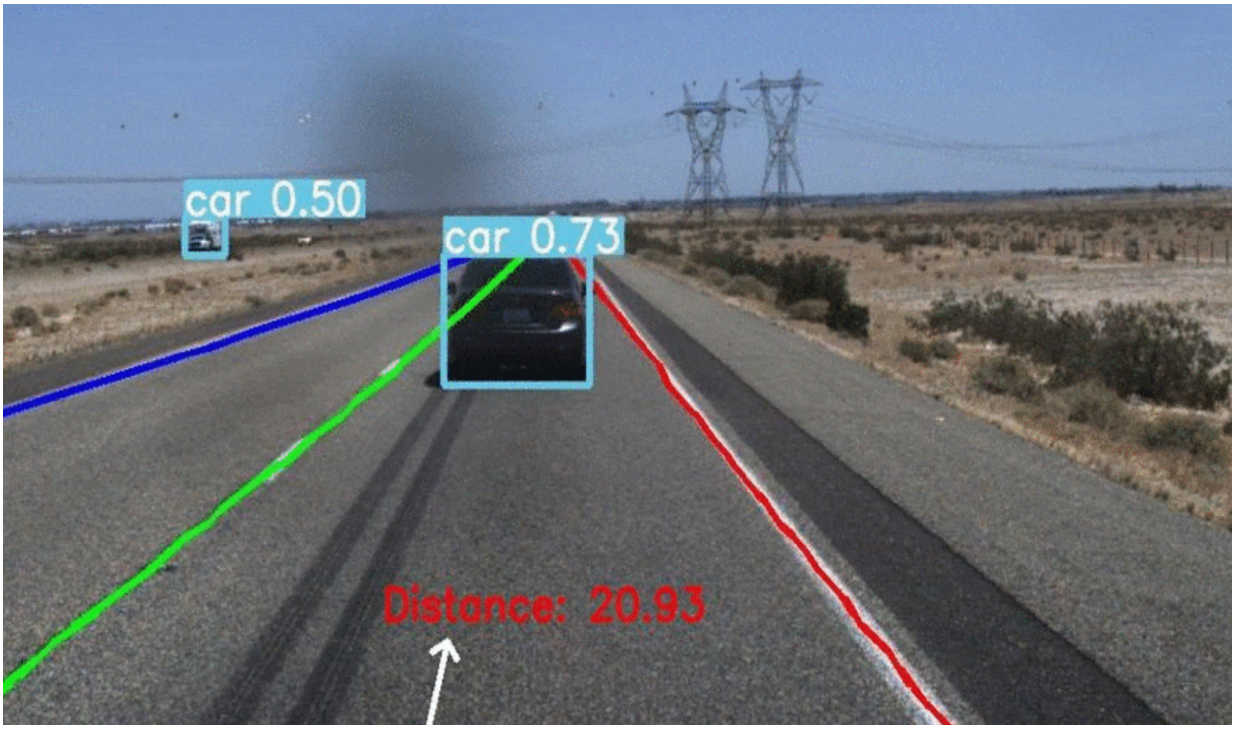
1. Presence of Shadows: Encountering shadows tend to mess up the CV algorithms we used since, shadows can be mistaken for gradient changes, and thus get detected as edges. This causes discrepancies in the model, thus giving unexpected results. There are however some algorithms for detection and removal of shadows that could be used for preprocessing .

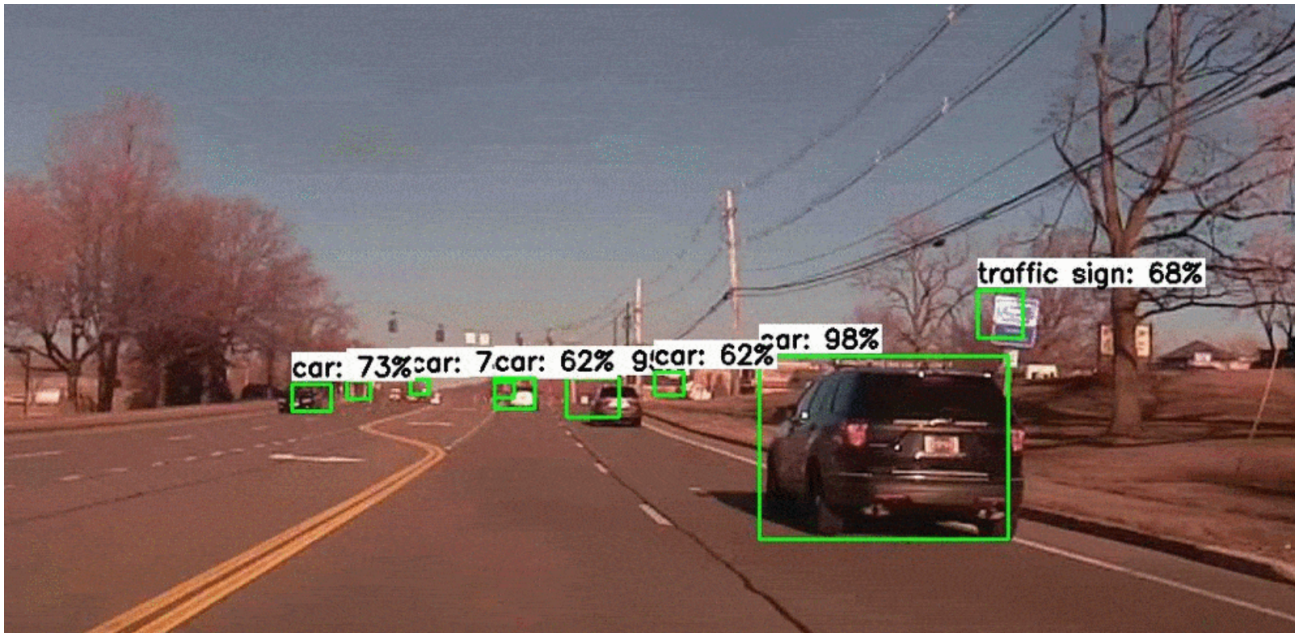
2. Presence of Road Sign markings or Crosswalks: We also noticed that the Sliding Histogram method tends to mess up when crosswalks are encountered, since these crosswalks are the same color as the lane lines.

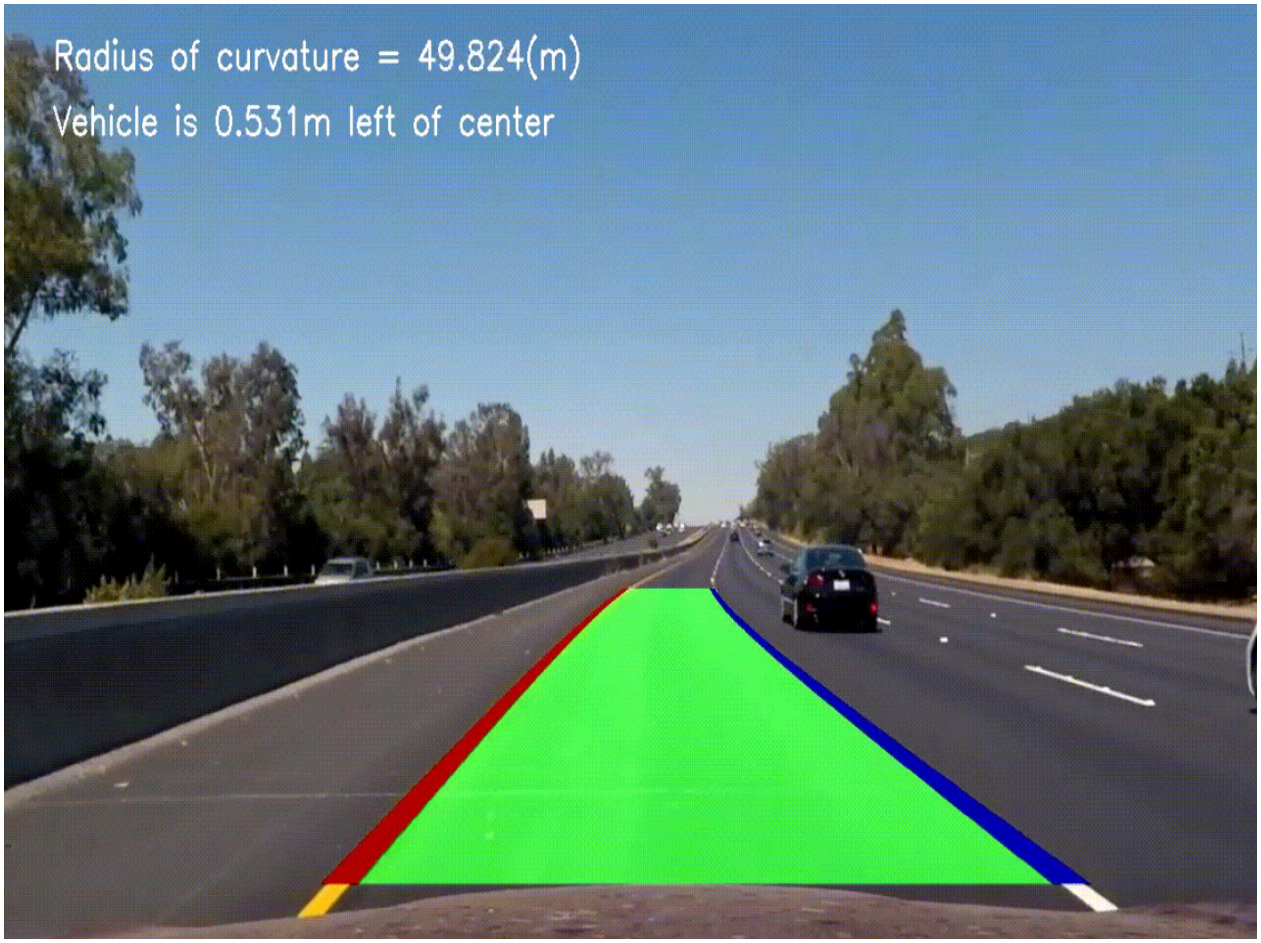
3. Double Yellow/White Lines: The histogram method can falsely detect the lane as being present between the two line of the left rather than the entire lane. We intend to resolve this issue by specifying a minimum distance between the detected lane lines, which should be wider than the car itself.

4. Unclear Lane Markings: Some edge cases included lane marking that were either worn out or covered by tar thus confusing the Computer Vision models into not detecting those as lines.

The Deep Learning model we utilized also performed surprisingly well given the fact that it was not specifically trained on our dataset. We intend to perform some transfer learning on these models to make them more specific for our use case. The drawbacks discussed above for the Computer Vision methods have been solved using Deep Leaning methods, given that you can label objects, essentially reducing the noise the model looks at to begin with


car 0.50
car 0.73
Distance: 20.93


traffic sign: 68%
car: 98%
car: 73%
car: 62%
car: 62%


Radius of curvature = 49.824(m)
Vehicle is 0.531m left of center

Considering that the Deep Learning model we used was not specially trained on our dataset, it also performed very well. To make these models more applicable to our use case, we want to apply some transfer learning to them. Given that you can label objects, thus lowering the noise the model looks at to begin with, the problems with computer vision methods outlined above have been resolved utilising deep learning approaches. These problems were as follows:

Since there are other ways, such as using mapping and localization, to determine where traffic is moving around you, lane colour detection is typically not solved using computer vision. We intended to attempt a computer vision-based solution to this issue. The continually shifting lighting conditions were the greatest obstacle to correctly determining the lane colour. We selected a variety of colours for both the yellow and the white lanes to guarantee accuracy in order to take this into consideration.

CHAPTER 5

CONCLUSION

The research compared Deep Learning with Traditional CV, and found assumptions, disadvantages, and benefits for each of these approaches. The most important lessons learned from the Computer Vision segment were that CV is very useful in situations when speedy processing was required, such as when performing tasks like Lane Colour and Type identification. On the other hand, deep learning is very helpful when computer vision struggles in the face of complex illumination or shadows. The amount and calibre of labelled data required for Deep Learning, however, is one of its main disadvantages.

Of course, the study is still ongoing, and with the aid of some transfer learning, we anticipate more rigorous results on the deep learning side.

The experiment carried out as part of our study was designed to demonstrate that analysing and recognising aggressive driving habits may be accomplished using low-cost techniques. We have been able to obtain significant training and validation accuracy to support the success of our project. Our model was validated with a high training accuracy of 90.17% and a high validation accuracy of 90.62%. With only 144,026 trainable parameters, these values were and are attainable. We have used computer vision technologies including thresholding, frame differencing, processing of contours utilising areas, and Hausdorff distance to enable the identification of automobiles participating in reckless driving, meet our objectives, and have a high percentage of positive metrics.

Through this project, we were able to introduce cutting-edge convolution neural network architectures like ResNet-50 and the EfficientNet-B7 to enable feature extraction, along with a straightforward Recurrent Neural Network model to enable classification, to create an efficient method that incorporates various modules like driving pattern classification and vehicle detection.

CHAPTER 6

FUTURE

SCOPE

In order to categorise aggressive driving and identify the cars involved, this study suggests a computationally efficient, economical, and successful mix of feature extraction, model architecture design, and detection algorithm from video footage. On the dataset we created and compiled, the experimental results were obtained using the specified regularisation and optimisation techniques. The foundation for recurrent neural networks may be further enhanced if huge, evenly distributed, and pre-processed datasets, as well as additional RAM and powerful servers, are available. The algorithms for feature extraction, picture segmentation, and Hausdorff distance can also be changed depending on the issue statement's domain [19] [20]. This project may be expanded to incorporate driver information like age, gender, stress levels, and number of crashes. This project may be expanded upon to incorporate driver information such as age, gender, stress levels, passenger counts, and other traffic-related factors including foreign object interference, broken traffic signals, climatic conditions, and improper sign board placement. These characteristics can be added to the classification process in order to classify the episodes into several subcategories of violent or non-aggressive conduct, or to create a multi-class classification issue. The video dataset may be expanded to incorporate footage from dashboard cameras, drones, multi-pivoted CCTV cameras, aerial top views of traffic, and more. The main strategy used in this project is enabling small, independent businesses to use our approaches in conjunction with their existing video surveillance software.

using the videos that were recorded by them as a dynamically expanding dataset to regularly train the model, specialising the algorithms and models according to their use case.

CHAPTER 7

REFERENCE

S

[1] "Canny edge detector," Wikipedia, 28-Dec-2021. [Online]. Available: https://en.wikipedia.org/wiki/Canny_edge_detector. [Accessed: 21-Jan-2022].

[2] "A computational approach to edge detection," IEEE Xplore, Nov-1986. [Online]. Available: https://ieeexplore.ieee.org/document/4767851. [Accessed: 21-Jan-2022].

[3] Z. Wang, W. Ren, and Q. Qiu, "LaneNet: Real-time Lane Detection Networks for autonomous driving," arXiv.org, 04-Jul-2018. [Online]. Available: https://arxiv.org/abs/1807.01726. [Accessed: 23-Jan-2022]..

[4] D. Shinar and R. Compton, "Aggressive Driving: An Observational Study of Driver, Vehicle, and Situational Variables," *Accident; analysis and prevention,* vol. 36, pp. 429-437, 2004.

[5] "Feature detection," OpenCV. [Online]. Available: https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga8618180a5948286384e3b7ca02f6feeb. [Accessed: 24-Mar-2022].

[6] J. He, S. Sun, D. Zhang, G. Wang and C. Zhang, "Lane Detection for Track-following Based on Histogram Statistics," 2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), 2019, pp. 1-2, doi: 10.1109/EDSSC.2019.8754094.

[7] Chen, H. Wang, "A Real-Time Lane Detection Algorithm Based on a Hyperbola-Pair," Proceedings of IV2006. IEEE Intelligent Vehicles Symposium, 2006

[8] Farid Bounini, Denis Gingras "Autonomous Vehicle and Real Time Road Lanes Detection and Tracking" 2015 IEEE Vehicle Power and Propulsion Conference (VPPC) 19-22 Oct. 2015 Montreal, QC, Canada

[9] Thomas Churack "Improved Road-Edge Detection and Path Planning For an Autonomous Sae Electric Race Car" 26th October 2015

[10] Kodi Balasriram, Manimozhi M "Curve Path Detection in Autonomous Vehicle using Deep Learning" May 2018 IEEE International Conference on Robotics and Automation (ICRA)

[11] Suvarna Shirke, Dr.C.Rajabhushanam "A Study Of Lane Detection Techniques And Lane Departure System" Published in June 2018

[12] Balakrishna Yadav Peddagolla, Santhosh Sivan Kathiresan, Nikhil S. Prabhu, Ninad C. Doshi, Mehrdad H. Zadeh, Ph.D., "On the Lane Detection for Autonomous Driving: A Computational Experimental Study on Performance of Edge Detectors" Published in July 2015, South African Transport Conference at Pretoria

[13] Sukriti Srivastava Manisha Lumb Ritika Singal , LCET Katani Kalan "Improved Lane Detection Using Hybrid Median Filter and Modified Hough Transform" Published in 2016

[14] Ajit Danti, Jyoti Y. Kulkarni, and P. S. Hiremath "An Image Processing Approach to Detect Lanes, Pot Holes and Recognize Road Signs in Indian Roads" Published in Jan 2012

[15] Guangqian Lu "A Lane Detection, Tracking and Recognition System for Smart Vehicles" Published 2014 in MSWiM.

[16] Sunil Kumar Vishwakarma, Akash, Divakar Singh Yadav "Analysis of Lane Detection Techniques using OpenCV" Conference: 2015 Annual IEEE India Conference (INDICON)

[17] Gowdham Prabhakar, Binsu Kailath, Sudha Natarajan, Rajesh Kumar "Obstacle Detection and Classification using Deep Learning for Tracking in High-Speed Autonomous Driving" Published in 2017 IEEE Region 10 Symposium (TENSYMP).

[18] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track,* vol. 9, pp. 249-256, 2010.

[19] Xu Yang, Zhang Ling "Research on Lane Detection Technology Based on OpenCv" 2015 3rd International Conference on Mechanical Engineering and Intelligent Systems

[20] L. Zhu, S. Arik, Y. Yang and T. Pfister, *Learning to Transfer Learn: Reinforcement Learning-Based Selection for Adaptive Transfer Learning,* 2019.

[21] Xin Liu, Bin Dai, Hangen He "Real-time On-Road Vehicle Detection Combining Specific Shadow Segmentation and SVM Classification" 2011 Second International Conference on Digital Manufacturing & Automation

[22] Teddy Ort, Liam Paull, Daniela Rus "Autonomous Vehicle Navigation in Rural Environments without Detailed Prior Maps" 2018 IEEE International Conference on Robotics and Automation (ICRA)

[23] Larry S. Davis, Todd Kushner "Road boundary detection for autonomous vehicle navigation" 1 March 1986

[24] M. Elarbi Boudihir, R. Nourine, D. Ziou "Visual Guidance of Autonomous Vehicle Based on Fuzzy Perception". January 1998

APPENDIX 1

Information about the language, programmes, and packages utilised in our project is provided in this area.

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language that was used to create this project. It provides code that is clear and readable. The AI and ML algorithms, although being extremely complicated with flexible processes, may aid developers in building solid and dependable machine intelligent systems when they are built in Python. Below is a list of the Python packages applied to our project:

- **Numpy:** It is a Python library that offers a multidimensional array object, various derived objects (like masked arrays and matrices), and a variety of routines for quick operations on arrays, such as discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and so forth. It is the cornerstone Python module for scientific computing.

- **Pandas:** It is a Python package that offers quick, adaptable, and expressive data structures designed to make it simple and natural to deal with "relational" or "labelled" data. It seeks to serve as the essential, high-level building block for using Python for actual, useful data analysis. Its overarching objective is to develop into the most potent and adaptable open source data analysis and manipulation tool accessible in any language.

- **Imgaug:** It is a library for machine learning experiments that use picture augmentation. It has a simple yet effective stochastic interface, supports a wide variety of augmentation techniques, makes it simple to combine them and execute them in random order or on multiple CPU cores, and can enhance not only images but also keypoints/landmarks, bounding boxes, heatmaps, and segmentation maps.

- **OpenCV-python:** A computer vision and machine learning software library called OpenCV is available for free use. OpenCV was created to offer a to speed the adoption of machine perception in commercial goods, there should be a standard infrastructure for computer vision applications. More than 2500 optimised algorithms are available in the collection, including a wide range of both traditional and cutting-edge computer vision and machine learning techniques. These algorithms can be used to find similar images from an image database, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch together images to produce high-resolution images of entire scenes, detect and recognise faces, and more.

- **PIL:** Together with other contributors, Fredrik Lundh created the Python Imaging Library. It enhances the Python interpreter's image processing capabilities and is made for quick access to data kept in a few fundamental pixel formats.

- **Matplotlib:** For building static, animated, and interactive visualisations in Python, this is a thorough library. It can construct interactive figures with zoom, pan, update, and customizable visual style and layout, export to a variety of file formats, and be incorporated in JupyterLab and graphical user interfaces. It can also produce plots suitable for publishing.

- **Keras:** It is a Python interface for an open-source software library that supports artificial neural networks. The TensorFlow library interface is provided by Keras. TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML were just a few of the backends that Keras supported up until version 2.3.

- **Tensorflow:** It is a machine learning and artificial intelligence software library that is open-source and free. Although it can be applied to many different tasks, deep neural network training and inference are given special attention. The Google Brain team created TensorFlow for use in internal Google research and production. A large number of programming languages, including Python, Javascript, C++, and Java, may use TensorFlow. It features a vast, adaptable ecosystem of resources, libraries, and community tools that enables researchers to advance the state-of-the-art in machine learning

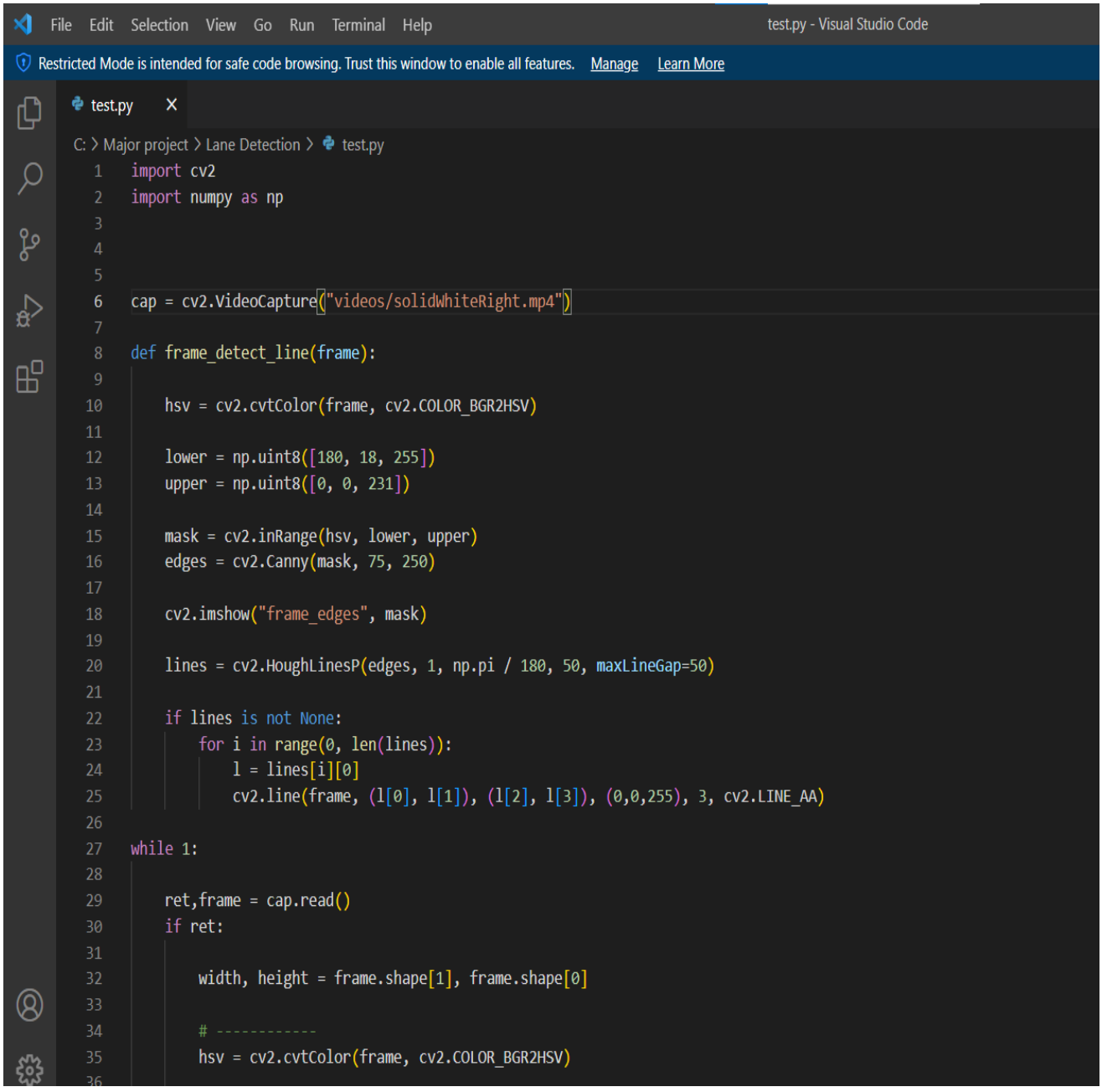
- **Scikit-learn:** It is a Python programming language machine learning package that is available for free. Support-vector machines, random forests, gradient boosting, k-means, and DBSCAN are just a few of the classification, regression, and clustering methods it offers. It is also made to work with Python's NumPy and SciPy libraries.

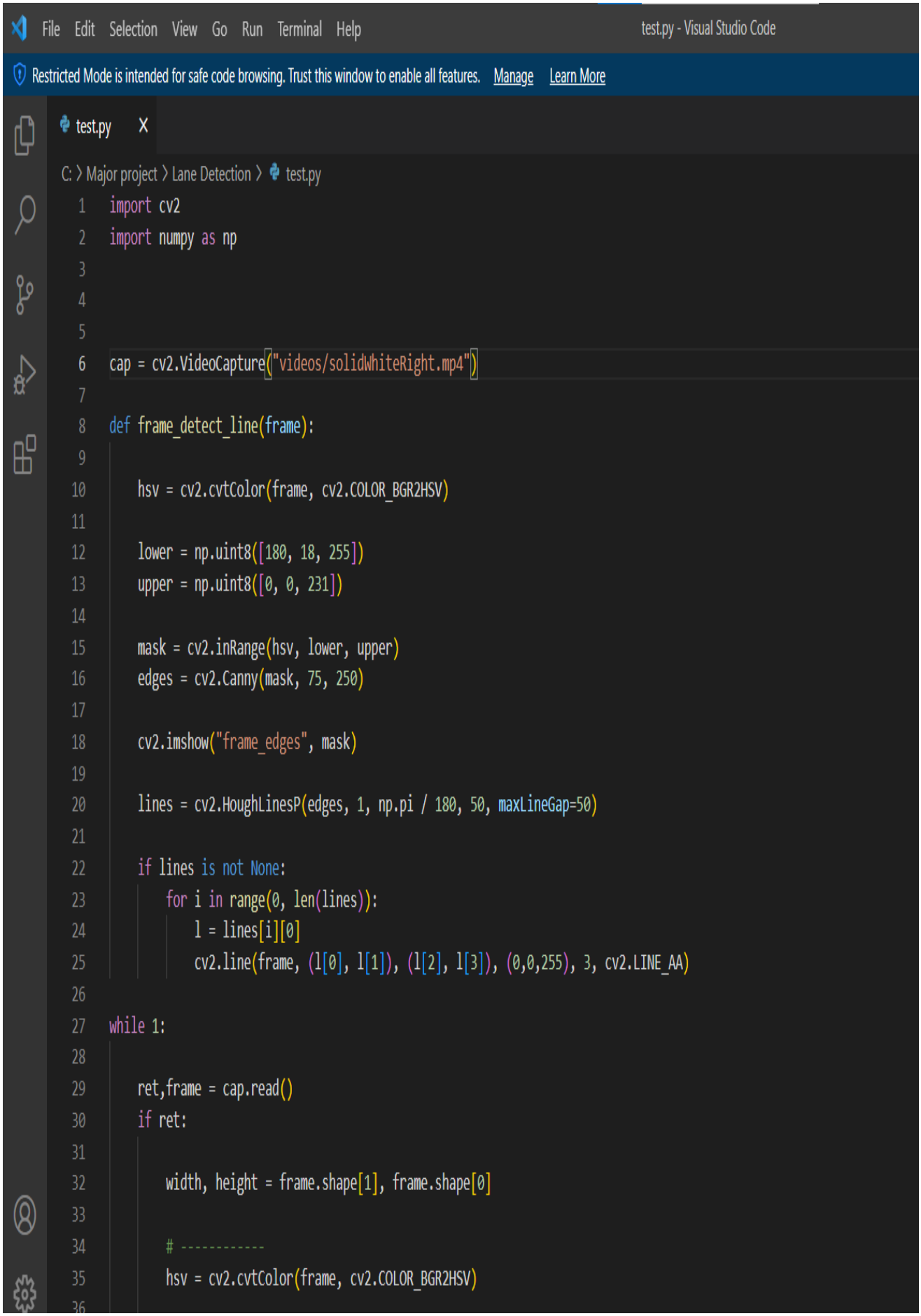
- **SciPy:** It is a Python library that is open-source, free, and used for technical and scientific computing. It includes modules for signal and image processing, interpolation, integration, special functions, and optimisation. SciPy uses a multidimensional array as its fundamental data structure, which is made available through the NumPy module.

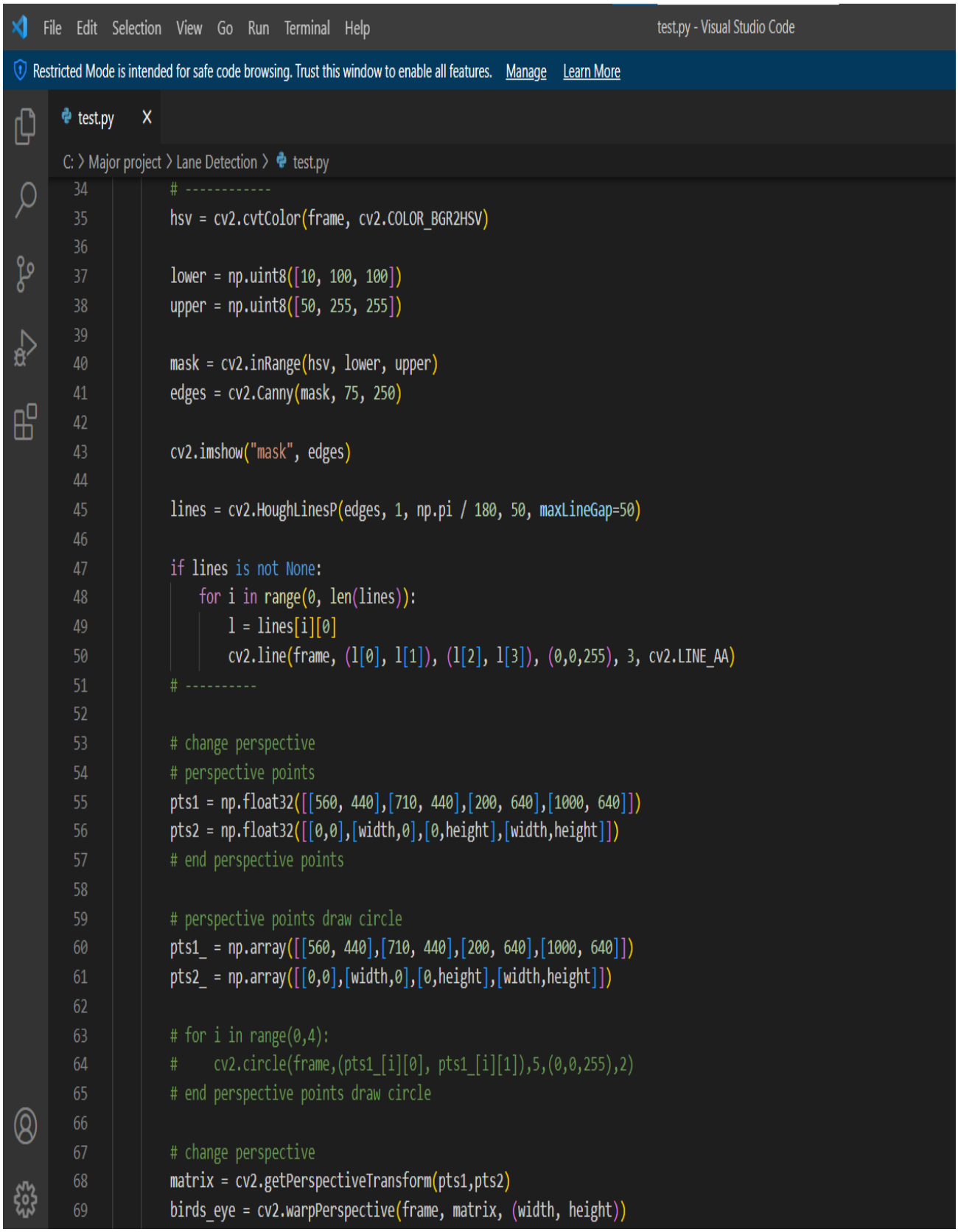
- **Tkinter:** It is the typical Python GUI library. The combination of Python with Tkinter makes it quick and simple to develop GUI apps. An effective object-oriented interface for the Tk GUI toolkit is provided by Tkinter. This embedded interpreter receives Tcl commands that are translated from Tkinter calls, allowing Python and Tcl to coexist in the same application.

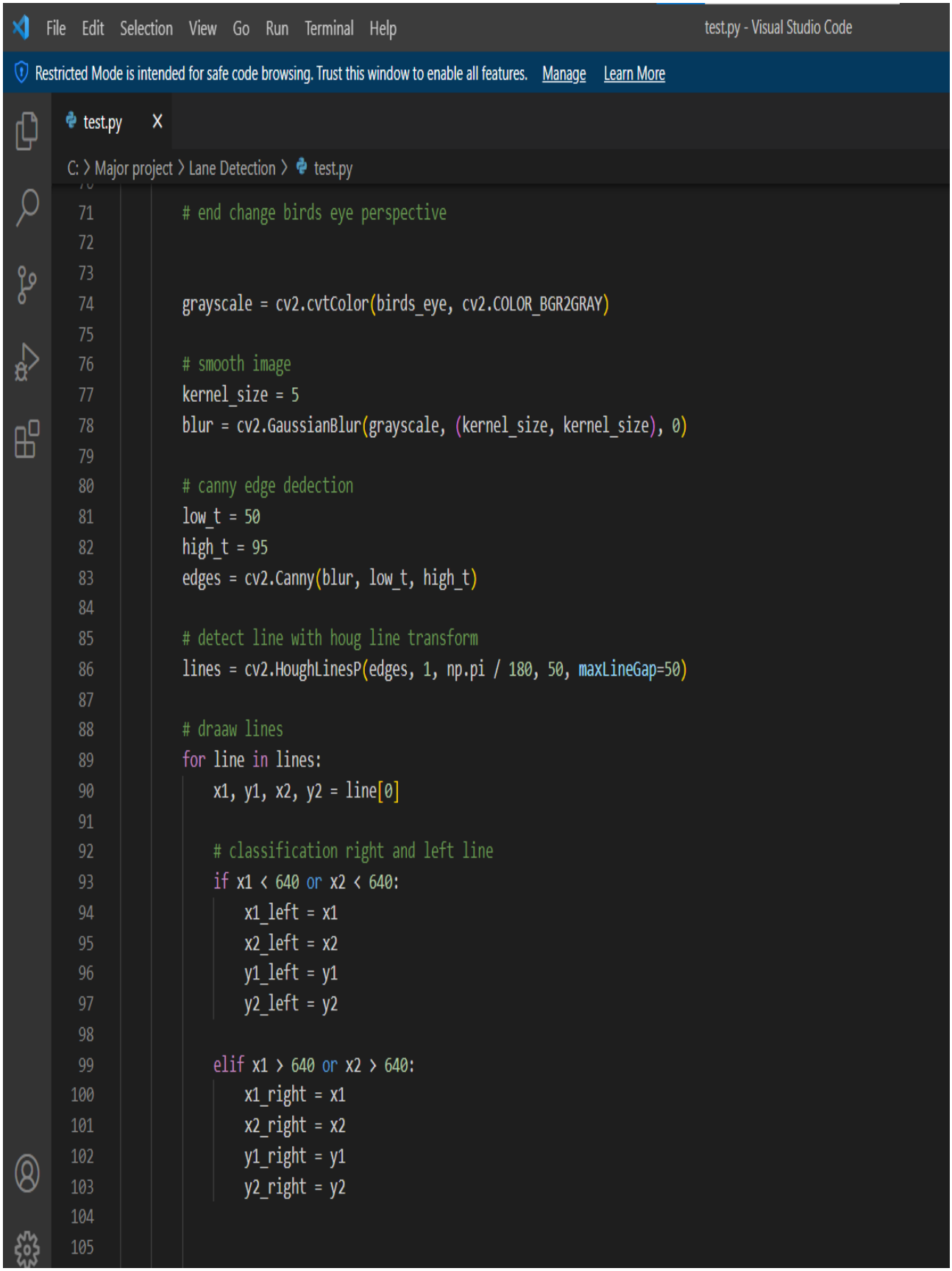
APPENDIX 2

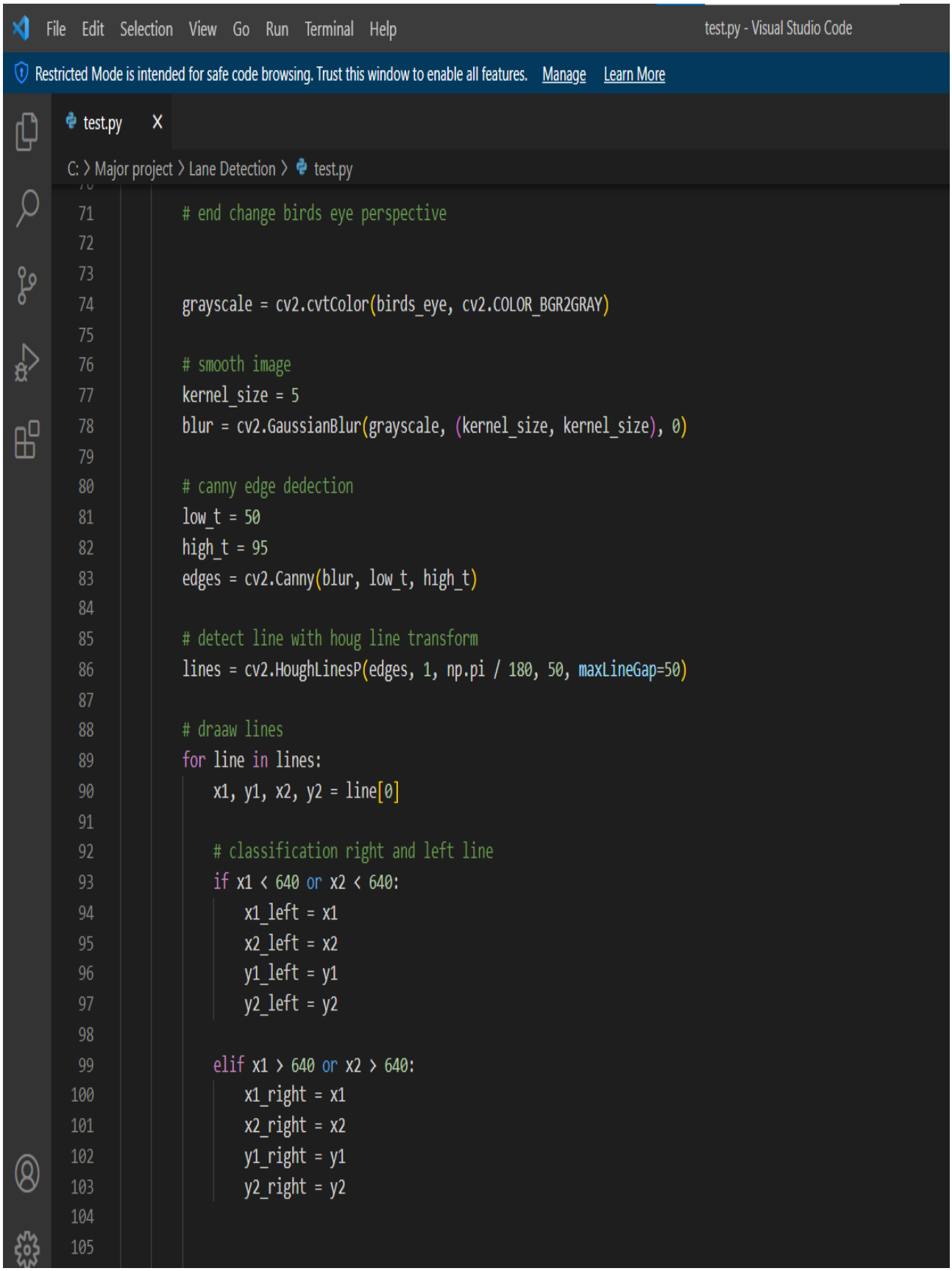
Model training and lane detection:

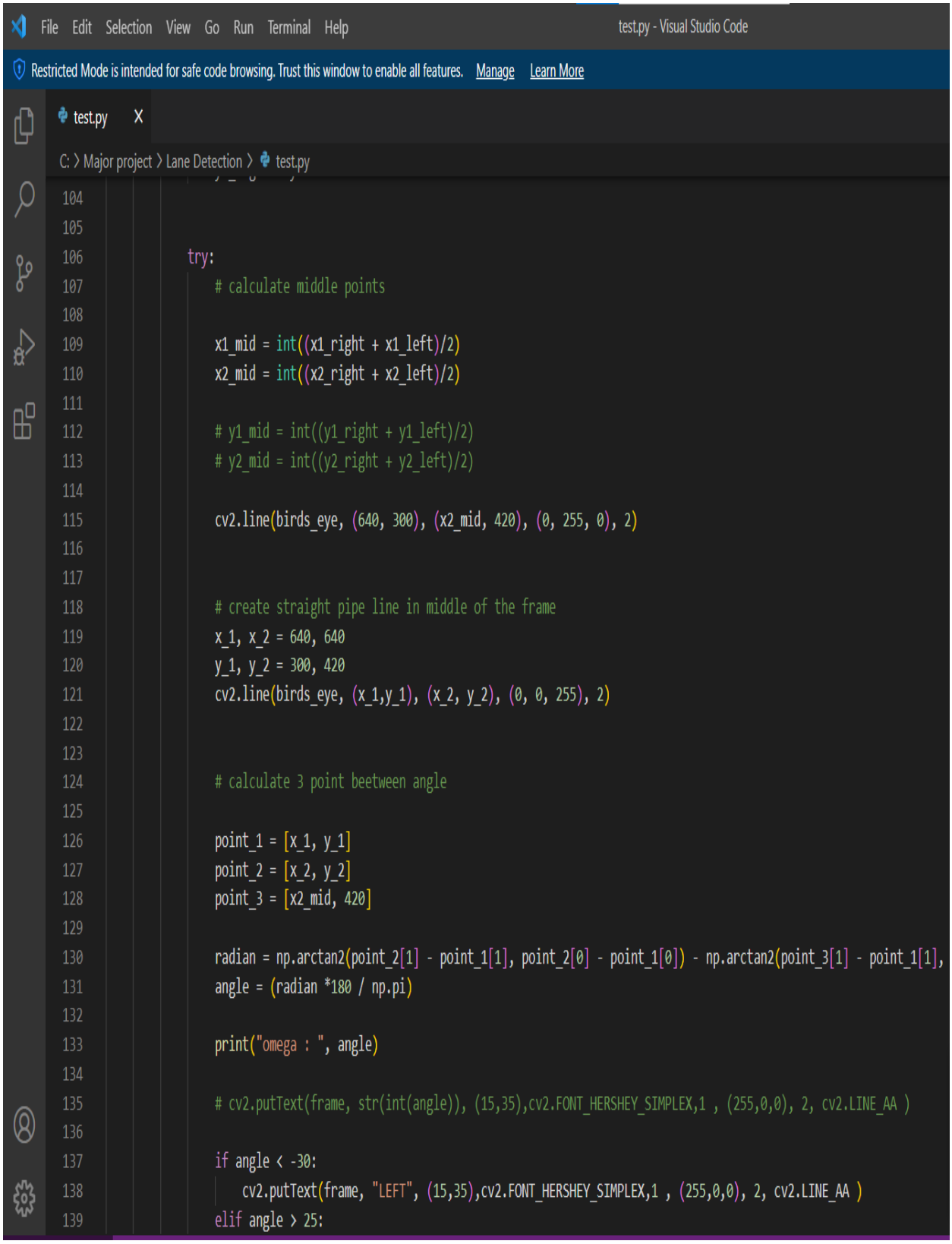


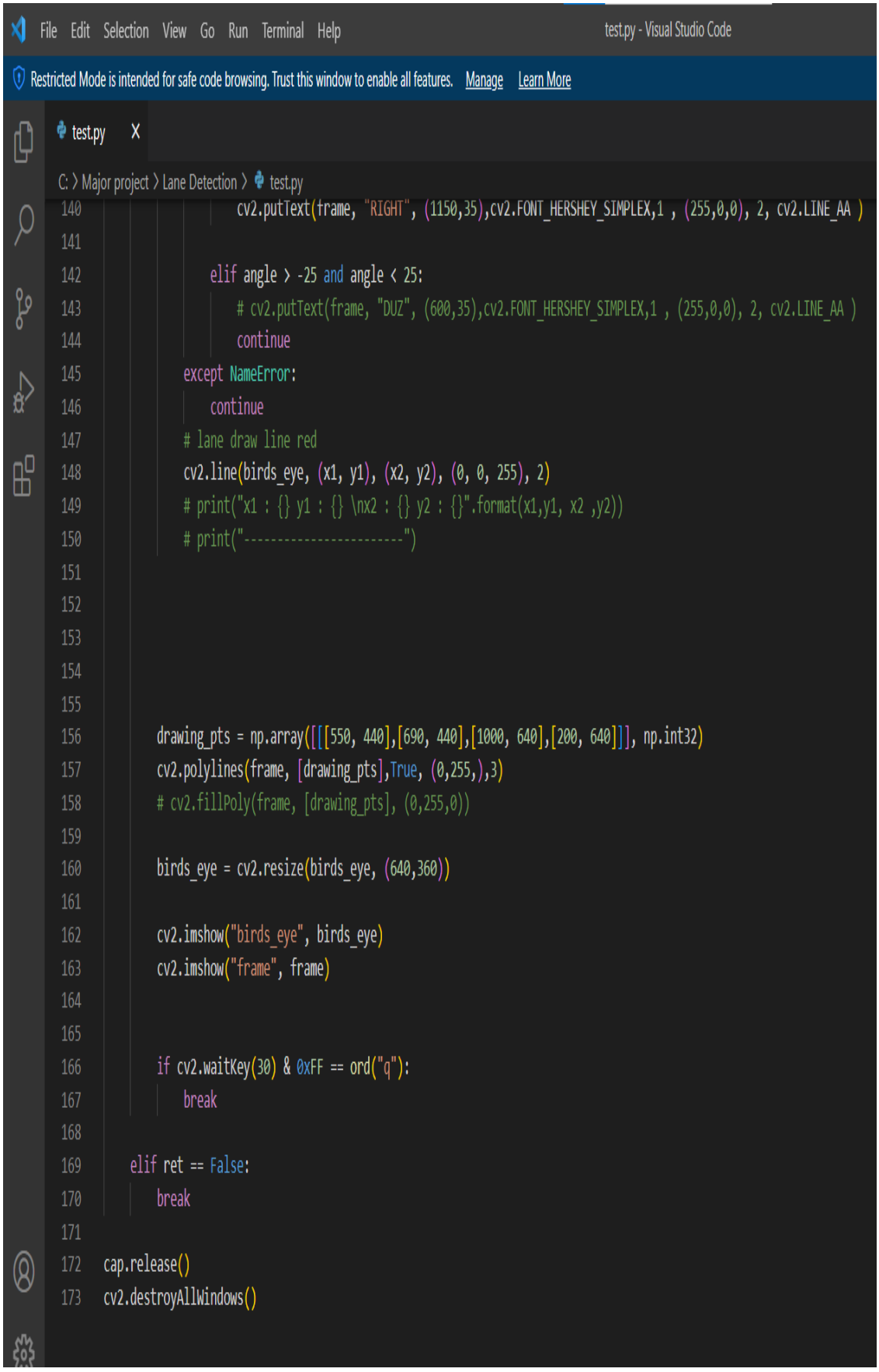

File Edit Selection View Go Run Terminal Help
test.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
C: > Major project > Lane Detection > test.py
import cv2
import numpy as np
cap = cv2.VideoCapture("videos/solidWhiteRight.mp4")
def frame_detect_line(frame):
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
lower = np.uint8([180, 18, 255])
upper = np.uint8([0, 0, 231])
mask = cv2.inRange(hsv, lower, upper)
edges = cv2.Canny(mask, 75, 250)
cv2.imshow("frame_edges", mask)
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, maxLineGap=50)
if lines is not None:
for i in range(0, len(lines)):
l = lines[i][0]
cv2.line(frame, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv2.LINE_AA)
while 1:
ret,frame = cap.read()
if ret:
width, height = frame.shape[1], frame.shape[0]
# ------------
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

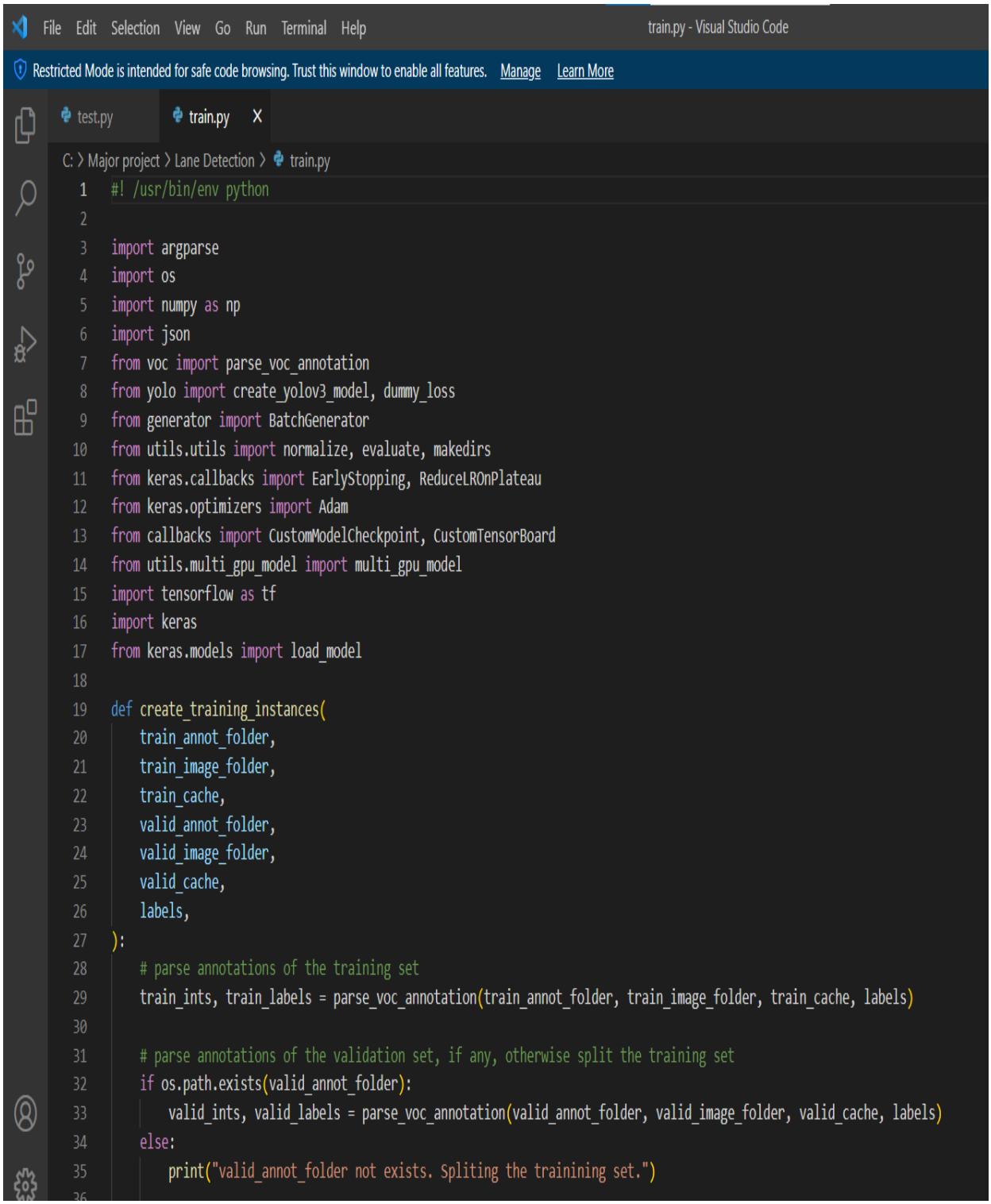

File Edit Selection View Go Run Terminal Help
test.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
C: > Major project > Lane Detection > test.py
# ------------
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
lower = np.uint8([10, 100, 100])
upper = np.uint8([50, 255, 255])
mask = cv2.inRange(hsv, lower, upper)
edges = cv2.Canny(mask, 75, 250)
cv2.imshow("mask", edges)
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, maxLineGap=50)
if lines is not None:
for i in range(0, len(lines)):
l = lines[i][0]
cv2.line(frame, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv2.LINE_AA)
# ----------
# change perspective
# perspective points
pts1 = np.float32([[560, 440],[710, 440],[200, 640],[1000, 640]])
pts2 = np.float32([[0,0],[width,0],[0,height],[width,height]])
# end perspective points
# perspective points draw circle
pts1_ = np.array([[560, 440],[710, 440],[200, 640],[1000, 640]])
pts2_ = np.array([[0,0],[width,0],[0,height],[width,height]])
# for i in range(0,4):
# cv2.circle(frame,(pts1_[i][0], pts1_[i][1]),5,(0,0,255),2)
# end perspective points draw circle
# change perspective
matrix = cv2.getPerspectiveTransform(pts1,pts2)
birds_eye = cv2.warpPerspective(frame, matrix, (width, height))

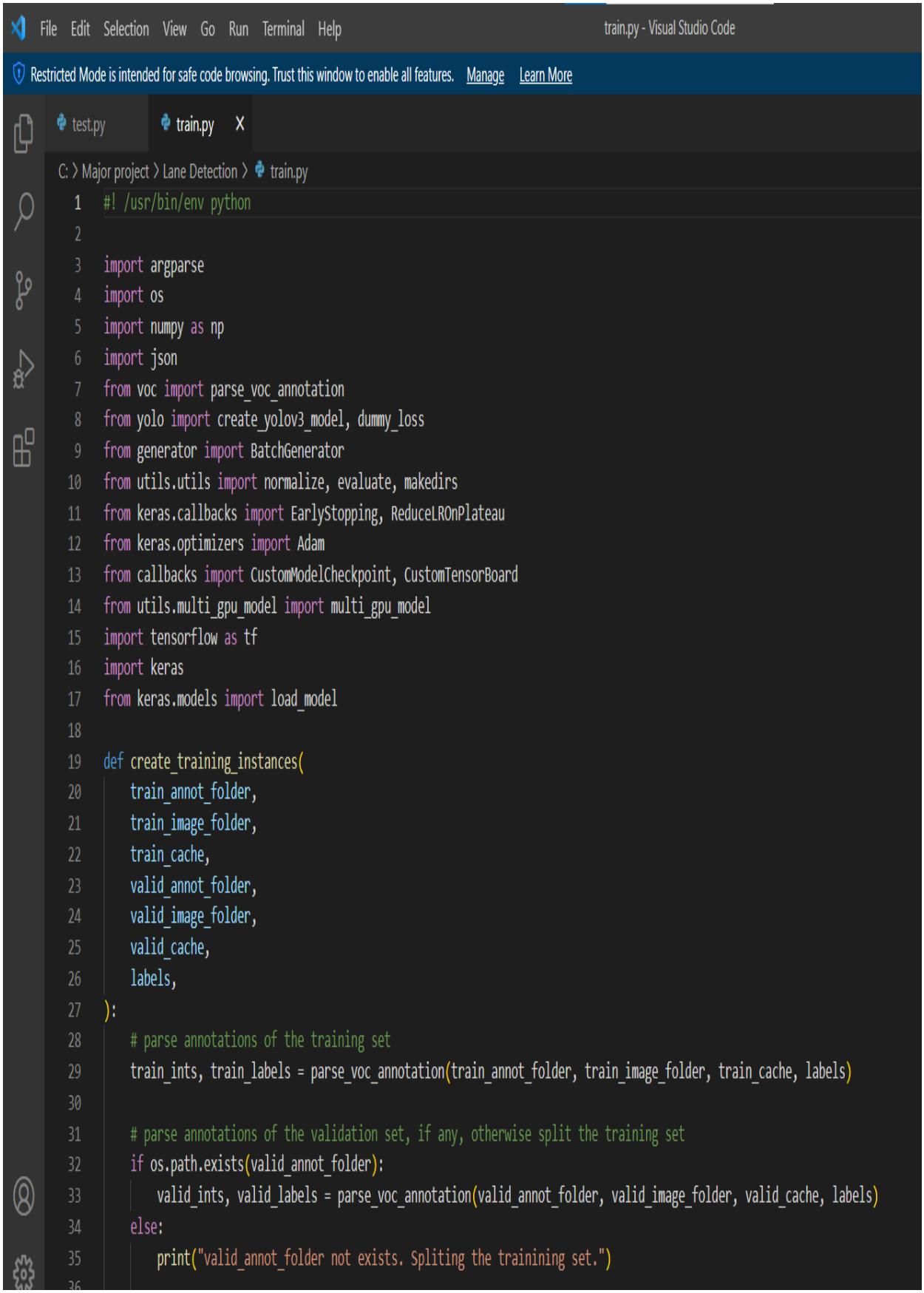

File Edit Selection View Go Run Terminal Help
test.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
C: > Major project > Lane Detection > test.py
# end change birds eye perspective
grayscale = cv2.cvtColor(birds_eye, cv2.COLOR_BGR2GRAY)
# smooth image
kernel_size = 5
blur = cv2.GaussianBlur(grayscale, (kernel_size, kernel_size), 0)
# canny edge dedection
low_t = 50
high_t = 95
edges = cv2.Canny(blur, low_t, high_t)
# detect line with houg line transform
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, maxLineGap=50)
# draaw lines
for line in lines:
x1, y1, x2, y2 = line[0]
# classification right and left line
if x1 < 640 or x2 < 640:
x1_left = x1
x2_left = x2
y1_left = y1
y2_left = y2
elif x1 > 640 or x2 > 640:
x1_right = x1
x2_right = x2
y1_right = y1
y2_right = y2

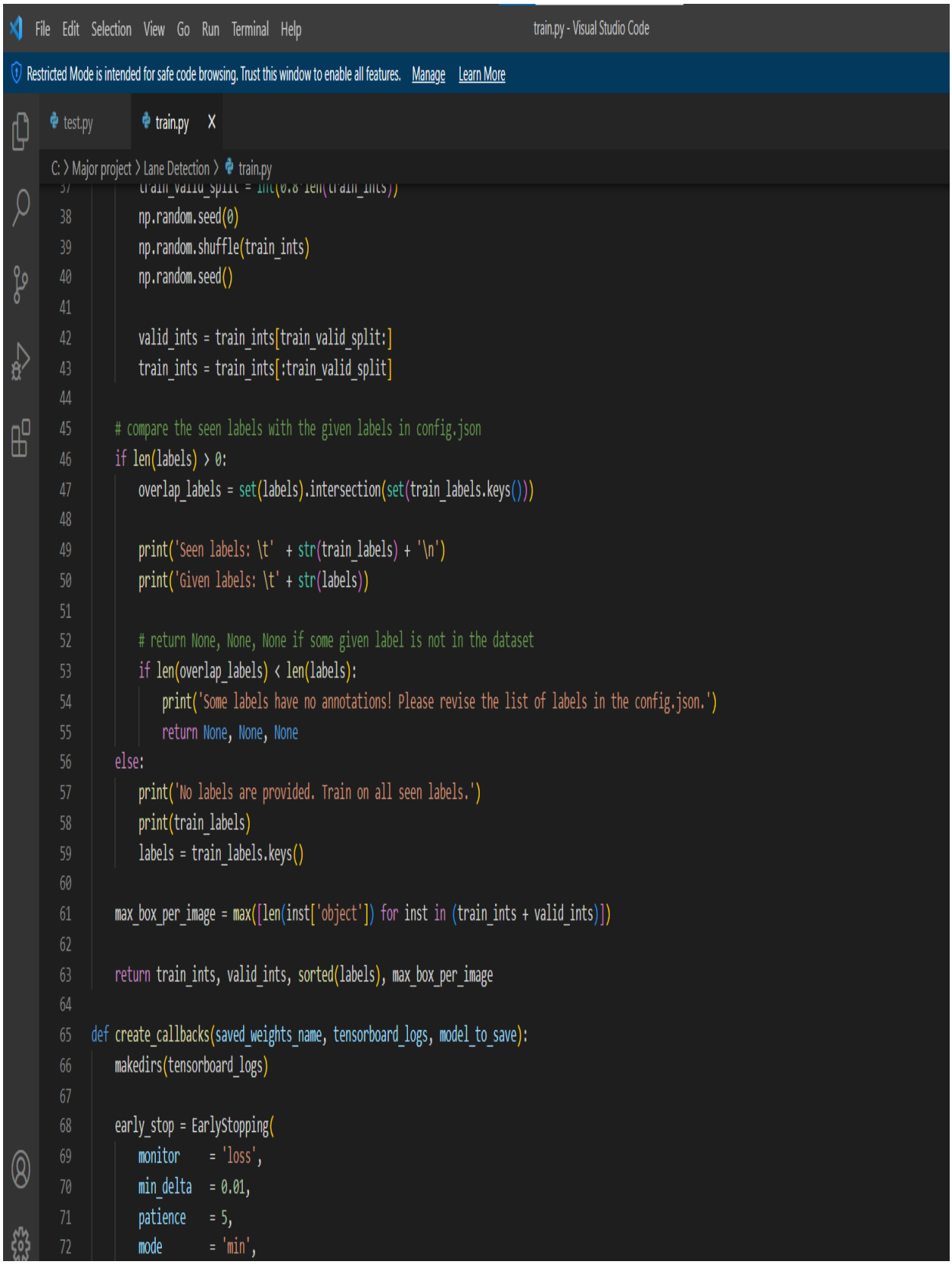

File Edit Selection View Go Run Terminal Help
test.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
C: > Major project > Lane Detection > test.py
# end change birds eye perspective
grayscale = cv2.cvtColor(birds_eye, cv2.COLOR_BGR2GRAY)
# smooth image
kernel_size = 5
blur = cv2.GaussianBlur(grayscale, (kernel_size, kernel_size), 0)
# canny edge dedection
low_t = 50
high_t = 95
edges = cv2.Canny(blur, low_t, high_t)
# detect line with houg line transform
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, maxLineGap=50)
# draaw lines
for line in lines:
x1, y1, x2, y2 = line[0]
# classification right and left line
if x1 < 640 or x2 < 640:
x1_left = x1
x2_left = x2
y1_left = y1
y2_left = y2
elif x1 > 640 or x2 > 640:
x1_right = x1
x2_right = x2
y1_right = y1
y2_right = y2

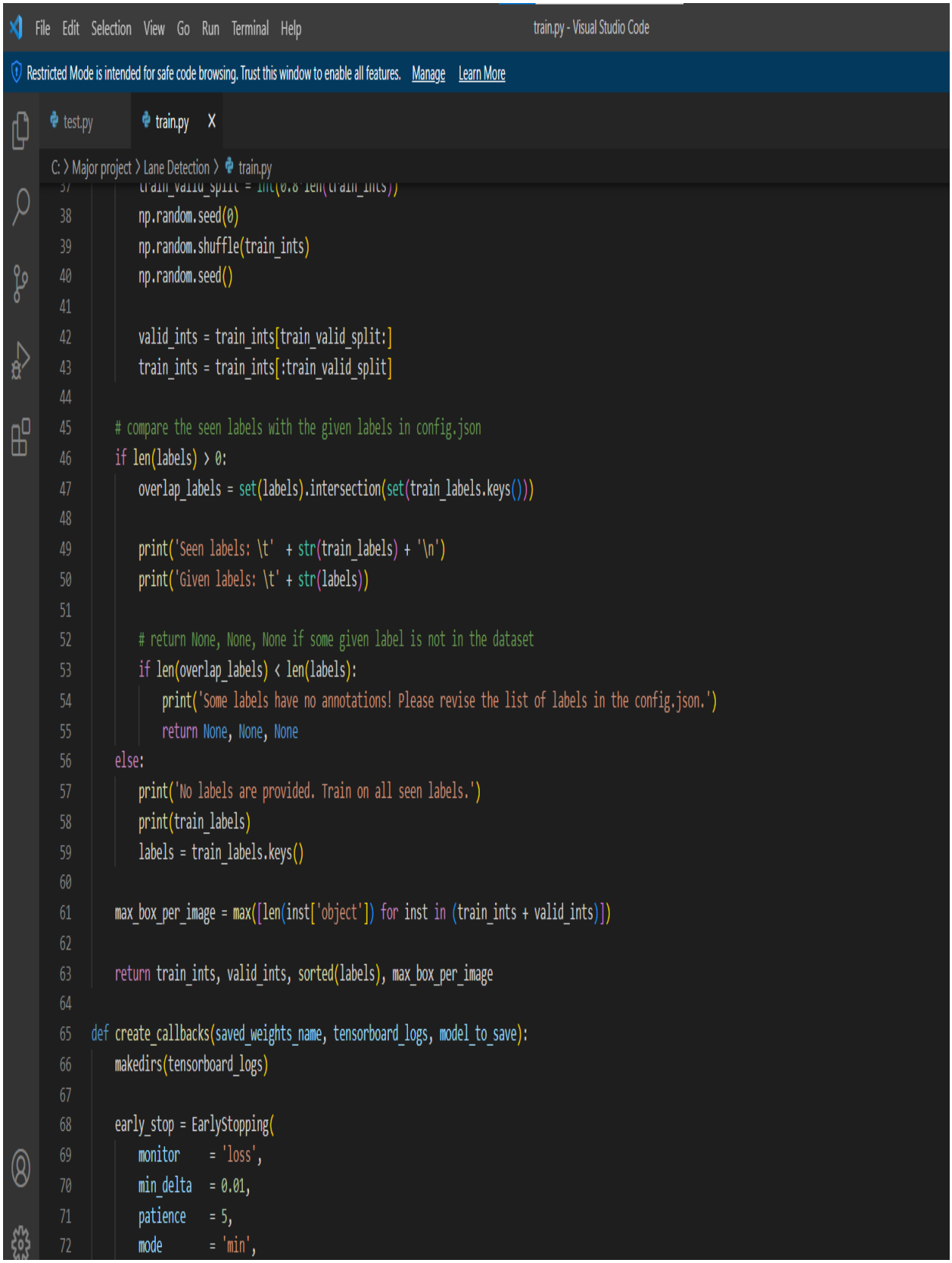

File Edit Selection View Go Run Terminal Help
test.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
C: > Major project > Lane Detection > test.py
104
105
106 try:
107 # calculate middle points
108
109 x1_mid = int((x1_right + x1_left)/2)
110 x2_mid = int((x2_right + x2_left)/2)
111
112 # y1_mid = int((y1_right + y1_left)/2)
113 # y2_mid = int((y2_right + y2_left)/2)
114
115 cv2.line(birds_eye, (640, 300), (x2_mid, 420), (0, 255, 0), 2)
116
117
118 # create straight pipe line in middle of the frame
119 x_1, x_2 = 640, 640
120 y_1, y_2 = 300, 420
121 cv2.line(birds_eye, (x_1,y_1), (x_2, y_2), (0, 0, 255), 2)
122
123
124 # calculate 3 point beetween angle
125
126 point_1 = [x_1, y_1]
127 point_2 = [x_2, y_2]
128 point_3 = [x2_mid, 420]
129
130 radian = np.arctan2(point_2[1] - point_1[1], point_2[0] - point_1[0]) - np.arctan2(point_3[1] - point_1[1],
131 angle = (radian *180 / np.pi)
132
133 print("omega : ", angle)
134
135 # cv2.putText(frame, str(int(angle)), (15,35),cv2.FONT_HERSHEY_SIMPLEX,1 , (255,0,0), 2, cv2.LINE_AA)
136
137 if angle < -30:
138 cv2.putText(frame, "LEFT", (15,35),cv2.FONT_HERSHEY_SIMPLEX,1 , (255,0,0), 2, cv2.LINE_AA)
139 elif angle > 25:

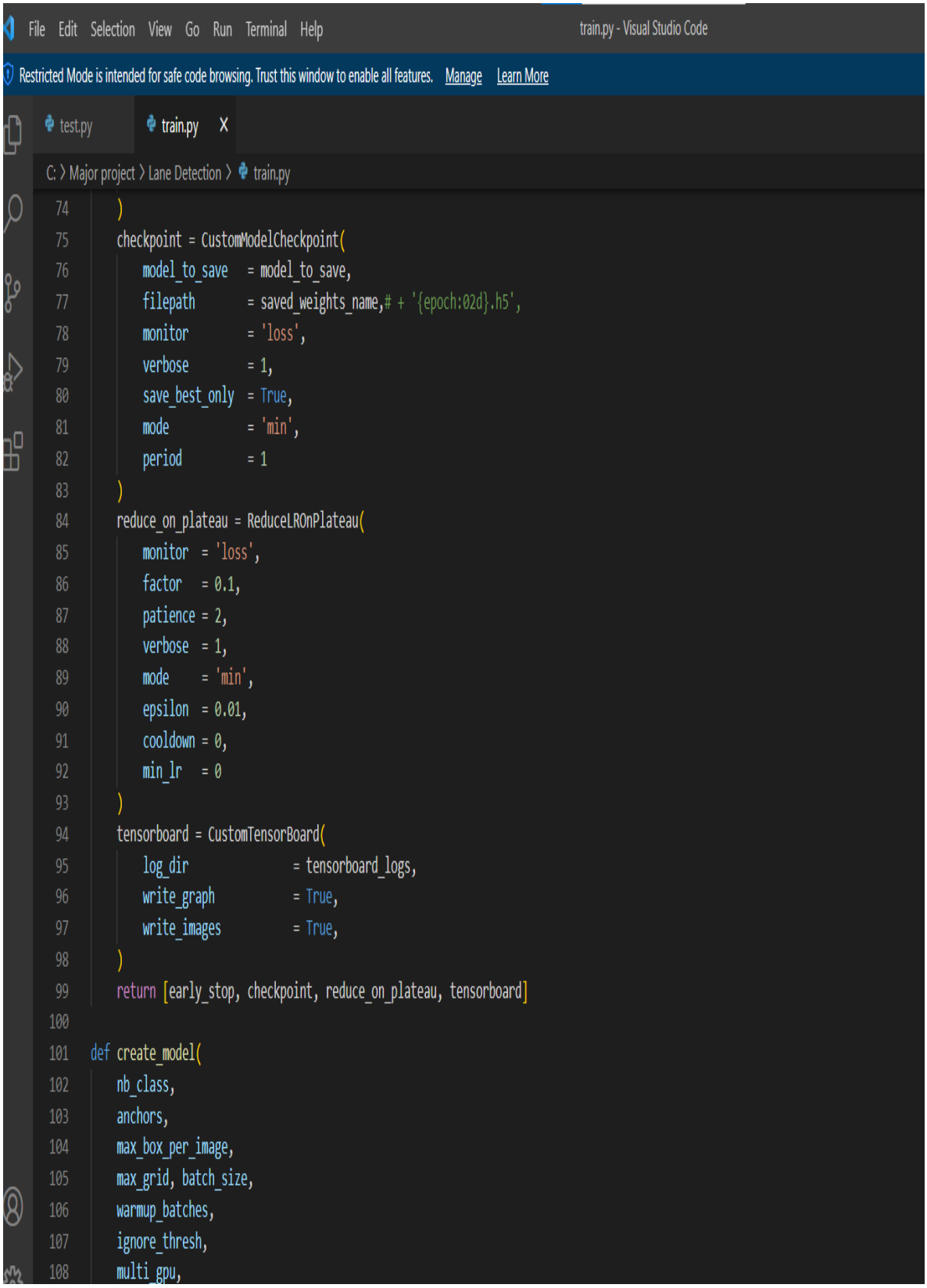

File Edit Selection View Go Run Terminal Help
test.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
C: > Major project > Lane Detection > test.py
cv2.putText(frame, "RIGHT", (1150,35),cv2.FONT_HERSHEY_SIMPLEX,1 , (255,0,0), 2, cv2.LINE_AA)
elif angle > -25 and angle < 25:
# cv2.putText(frame, "DUZ", (600,35),cv2.FONT_HERSHEY_SIMPLEX,1 , (255,0,0), 2, cv2.LINE_AA)
continue
except NameError:
continue
# lane draw line red
cv2.line(birds_eye, (x1, y1), (x2, y2), (0, 0, 255), 2)
# print("x1 : {} y1 : {} \nx2 : {} y2 : {}".format(x1,y1, x2 ,y2))
# print("----------")
drawing_pts = np.array([[[550, 440],[690, 440],[1000, 640],[200, 640]]], np.int32)
cv2.polylines(frame, [drawing_pts],True, (0,255,),3)
# cv2.fillPoly(frame, [drawing_pts], (0,255,0))
birds_eye = cv2.resize(birds_eye, (640,360))
cv2.imshow("birds_eye", birds_eye)
cv2.imshow("frame", frame)
if cv2.waitKey(30) & 0xFF == ord("q"):
break
elif ret == False:
break
cap.release()
cv2.destroyAllWindows()

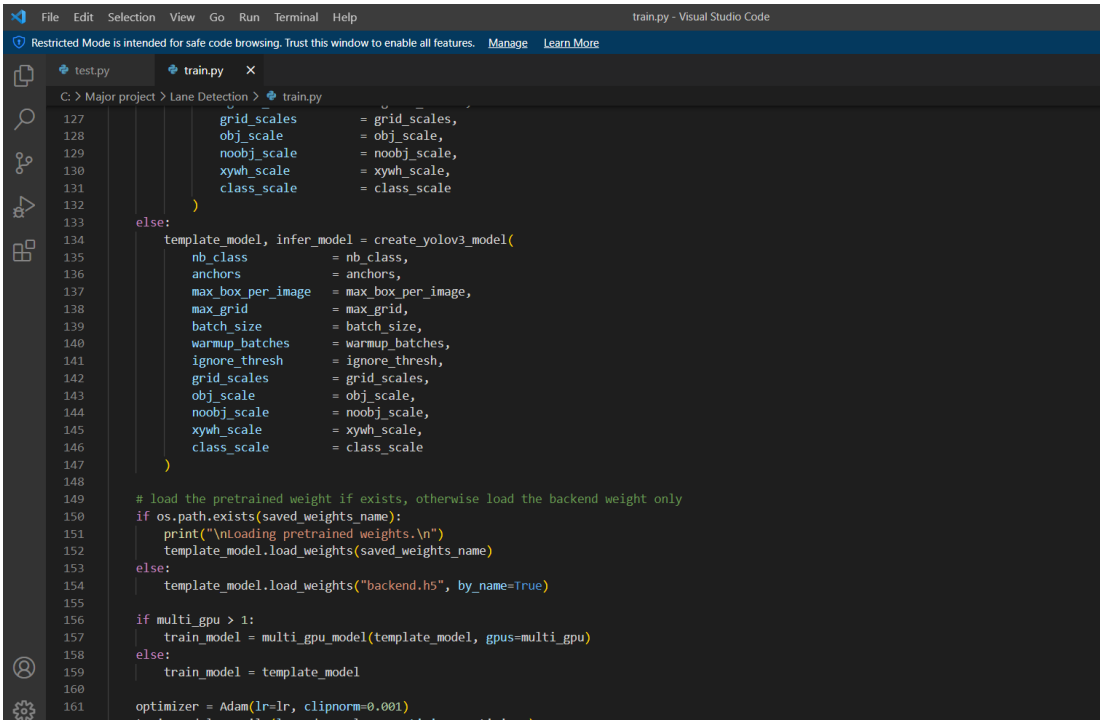

File Edit Selection View Go Run Terminal Help
train.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
train.py
C: > Major project > Lane Detection > train.py
#! /usr/bin/env python
import argparse
import os
import numpy as np
import json
from voc import parse_voc_annotation
from yolo import create_yolov3_model, dummy_loss
from generator import BatchGenerator
from utils.utils import normalize, evaluate, makedirs
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam
from callbacks import CustomModelCheckpoint, CustomTensorBoard
from utils.multi_gpu_model import multi_gpu_model
import tensorflow as tf
import keras
from keras.models import load_model
def create_training_instances(
train_annot_folder,
train_image_folder,
train_cache,
valid_annot_folder,
valid_image_folder,
valid_cache,
labels,
):
parse annotations of the training set
train_ints, train_labels = parse_voc_annotation(train_annot_folder, train_image_folder, train_cache, labels)
# parse annotations of the validation set, if any, otherwise split the training set
if os.path.exists(valid_annot_folder):
valid_ints, valid_labels = parse_voc_annotation(valid_annot_folder, valid_image_folder, valid_cache, labels)
else:
print("valid_annot_folder not exists. Spliting the trainining set.")

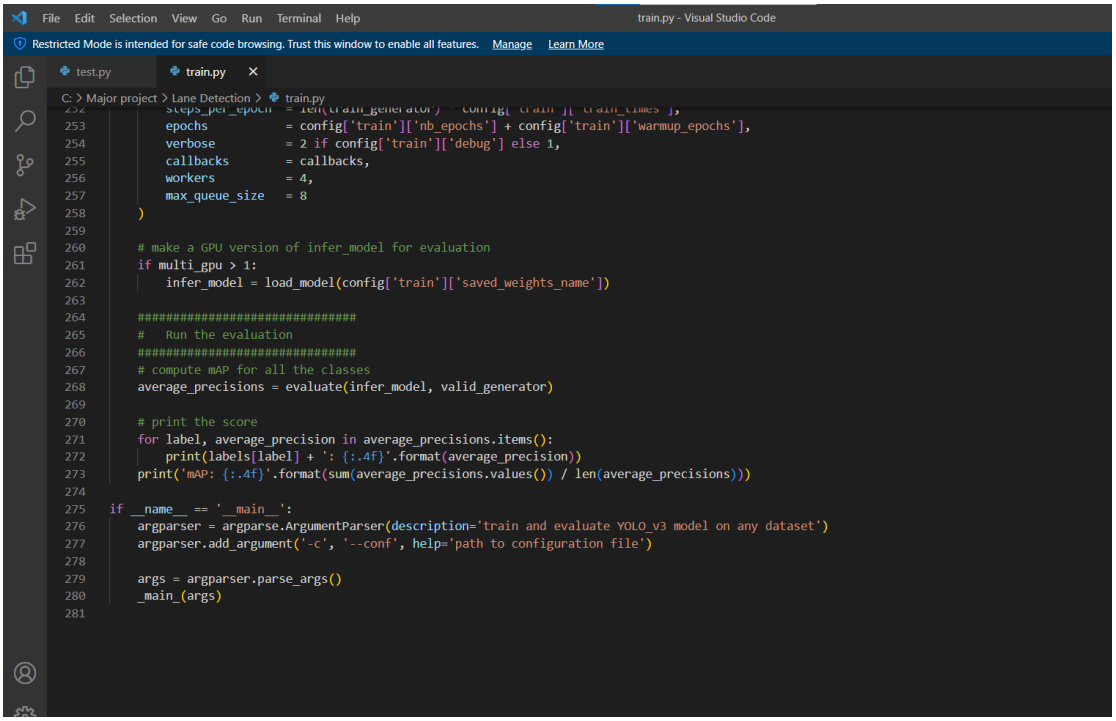

File Edit Selection View Go Run Terminal Help
train.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
train.py
C: > Major project > Lane Detection > train.py
#! /usr/bin/env python
import argparse
import os
import numpy as np
import json
from voc import parse_voc_annotation
from yolo import create_yolov3_model, dummy_loss
from generator import BatchGenerator
from utils.utils import normalize, evaluate, makedirs
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam
from callbacks import CustomModelCheckpoint, CustomTensorBoard
from utils.multi_gpu_model import multi_gpu_model
import tensorflow as tf
import keras
from keras.models import load_model
def create_training_instances(
train_annot_folder,
train_image_folder,
train_cache,
valid_annot_folder,
valid_image_folder,
valid_cache,
labels,
):
parse annotations of the training set
train_ints, train_labels = parse_voc_annotation(train_annot_folder, train_image_folder, train_cache, labels)
# parse annotations of the validation set, if any, otherwise split the training set
if os.path.exists(valid_annot_folder):
valid_ints, valid_labels = parse_voc_annotation(valid_annot_folder, valid_image_folder, valid_cache, labels)
else:
print("valid_annot_folder not exists. Spliting the trainining set.")


File Edit Selection View Go Run Terminal Help
train.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
train.py
C: > Major project > Lane Detection > train.py
train_valid_split = int(0.8*len(train_ints))
np.random.seed(0)
np.random.shuffle(train_ints)
np.random.seed()
valid_ints = train_ints[train_valid_split:]
train_ints = train_ints[:train_valid_split]
# compare the seen labels with the given labels in config.json
if len(labels) > 0:
overlap_labels = set(labels).intersection(set(train_labels.keys()))
print('Seen labels: \t' + str(train_labels) + '\n')
print('Given labels: \t' + str(labels))
# return None, None, None if some given label is not in the dataset
if len(overlap_labels) < len(labels):
print('Some labels have no annotations! Please revise the list of labels in the config.json.')
return None, None, None
else:
print('No labels are provided. Train on all seen labels.')
print(train_labels)
labels = train_labels.keys()
max_box_per_image = max([len(inst['object']) for inst in (train_ints + valid_ints)])
return train_ints, valid_ints, sorted(labels), max_box_per_image
def create_callbacks(saved_weights_name, tensorboard_logs, model_to_save):
makedirs(tensorboard_logs)
early_stop = EarlyStopping(
monitor = 'loss',
min_delta = 0.01,
patience = 5,
mode = 'min',


File Edit Selection View Go Run Terminal Help
train.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
train.py
C: > Major project > Lane Detection > train.py
train_valid_split = int(0.8*len(train_ints))
np.random.seed(0)
np.random.shuffle(train_ints)
np.random.seed()
valid_ints = train_ints[train_valid_split:]
train_ints = train_ints[:train_valid_split]
# compare the seen labels with the given labels in config.json
if len(labels) > 0:
overlap_labels = set(labels).intersection(set(train_labels.keys()))
print('Seen labels: \t' + str(train_labels) + '\n')
print('Given labels: \t' + str(labels))
# return None, None, None if some given label is not in the dataset
if len(overlap_labels) < len(labels):
print('Some labels have no annotations! Please revise the list of labels in the config.json.')
return None, None, None
else:
print('No labels are provided. Train on all seen labels.')
print(train_labels)
labels = train_labels.keys()
max_box_per_image = max([len(inst['object']) for inst in (train_ints + valid_ints)])
return train_ints, valid_ints, sorted(labels), max_box_per_image
def create_callbacks(saved_weights_name, tensorboard_logs, model_to_save):
makedirs(tensorboard_logs)
early_stop = EarlyStopping(
monitor = 'loss',
min_delta = 0.01,
patience = 5,
mode = 'min',


File Edit Selection View Go Run Terminal Help
train.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
train.py X
C: > Major project > Lane Detection > train.py
74)
75 checkpoint = CustomModelCheckpoint(
76 model_to_save = model_to_save,
77 filepath = saved_weights_name,# + '{epoch:02d}.h5',
78 monitor = 'loss',
79 verbose = 1,
80 save_best_only = True,
81 mode = 'min',
82 period = 1
83)
84 reduce_on_plateau = ReduceLROnPlateau(
85 monitor = 'loss',
86 factor = 0.1,
87 patience = 2,
88 verbose = 1,
89 mode = 'min',
90 epsilon = 0.01,
91 cooldown = 0,
92 min_lr = 0
93)
94 tensorboard = CustomTensorBoard(
95 log_dir = tensorboard_logs,
96 write_graph = True,
97 write_images = True,
98)
99 return [early_stop, checkpoint, reduce_on_plateau, tensorboard]
100
101 def create_model(
102 nb_class,
103 anchors,
104 max_box_per_image,
105 max_grid, batch_size,
106 warmup_batches,
107 ignore_thresh,
108 multi_gpu,


File Edit Selection View Go Run Terminal Help
train.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
train.py
C: > Major project > Lane Detection > train.py
grid_scales = grid_scales,
obj_scale = obj_scale,
noobj_scale = noobj_scale,
xywh_scale = xywh_scale,
class_scale = class_scale
)
else:
template_model, infer_model = create_yolov3_model(
nb_class = nb_class,
anchors = anchors,
max_box_per_image = max_box_per_image,
max_grid = max_grid,
batch_size = batch_size,
warmup_batches = warmup_batches,
ignore_thresh = ignore_thresh,
grid_scales = grid_scales,
obj_scale = obj_scale,
noobj_scale = noobj_scale,
xywh_scale = xywh_scale,
class_scale = class_scale
)
# load the pretrained weight if exists, otherwise load the backend weight only
if os.path.exists(saved_weights_name):
print("\nLoading pretrained weights.\n")
template_model.load_weights(saved_weights_name)
else:
template_model.load_weights("backend.h5", by_name=True)
if multi_gpu > 1:
train_model = multi_gpu_model(template_model, gpus=multi_gpu)
else:
train_model = template_model
optimizer = Adam(lr=lr, clipnorm=0.001)


File Edit Selection View Go Run Terminal Help
train.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
test.py
train.py
C: > Major project > Lane Detection > train.py
epochs = config['train']['nb_epochs'] + config['train']['warmup_epochs'],
verbose = 2 if config['train']['debug'] else 1,
callbacks = callbacks,
workers = 4,
max_queue_size = 8
)
# make a GPU version of infer_model for evaluation
if multi_gpu > 1:
infer_model = load_model(config['train']['saved_weights_name'])
################################
Run the evaluation
################################
# compute mAP for all the classes
average_precisions = evaluate(infer_model, valid_generator)
# print the score
for label, average_precision in average_precisions.items():
print(labels[label] + ': {:.4f}'.format(average_precision))
print('mAP: {:.4f}'.format(sum(average_precisions.values()) / len(average_precisions)))
if __name__ == '__main__':
argparser = argparse.ArgumentParser(description='train and evaluate YOLO_v3 model on any dataset')
argparser.add_argument('-c', '--conf', help='path to configuration file')
args = argparser.parse_args()
main(args)

Output module:

```
 import cv2
import numpy as np


cap =
cv2.VideoCapture("v
ideos/solidWhiteRig
ht.mp4")

def
frame_detect_line(fr
ame):

   hsv =
cv2.cvtColor(frame,
cv2.COLOR_BGR2
HSV)

   lower =
np.uint8([180, 18,
255])
   upper =
np.uint8([0, 0, 231])

   mask =
cv2.inRange(hsv,
lower, upper)
   edges =
cv2.Canny(mask, 75,
250)


cv2.imshow("frame_
edges", mask)

   lines =
cv2.HoughLinesP(ed
ges, 1, np.pi / 180,
50,
maxLineGap=50)

   if lines is not
None:
```

```
        for i in range(0,
len(lines)):
            l = lines[i][0]

cv2.line(frame, (l[0],
l[1]), (l[2], l[3]),
(0,0,255), 3,
cv2.LINE_AA)

while 1:

    ret,frame =
cap.read()
    if ret:

        width, height =
frame.shape[1],
frame.shape[0]

        # ------------
        hsv =
cv2.cvtColor(frame,
cv2.COLOR_BGR2
HSV)

        lower =
np.uint8([10, 100,
100])
        upper =
np.uint8([50, 255,
255])

        mask =
cv2.inRange(hsv,
lower, upper)
        edges =
cv2.Canny(mask, 75,
250)


cv2.imshow("mask",
edges)

        lines =
cv2.HoughLinesP(ed
ges, 1, np.pi / 180,
50,
maxLineGap=50)

        if lines is not
None:
            for i in
```

```
range(0, len(lines)):
                l =
import cv2
import numpy as np



cap =
cv2.VideoCapture("v
ideos/solidWhiteRig
ht.mp4")

def
frame_detect_line(fr
ame):

    hsv =
cv2.cvtColor(frame,
cv2.COLOR_BGR2
HSV)

    lower =
np.uint8([180, 18,
255])
    upper =
np.uint8([0, 0, 231])

    mask =
cv2.inRange(hsv,
lower, upper)
    edges =
cv2.Canny(mask, 75,
250)


cv2.imshow("frame_
edges", mask)

    lines =
cv2.HoughLinesP(ed
ges, 1, np.pi / 180,
50,
maxLineGap=50)

    if lines is not
None:
        for i in range(0,
len(lines)):
```

```
  # calculate 3 point
beetween angle

            point_1 =
[x_1, y_1]
            point_2 =
[x_2, y_2]
            point_3 =
[x2_mid, 420]

            radian =
np.arctan2(point_2[1
] - point_1[1],
point_2[0] -
point_1[0]) -
np.arctan2(point_3[1
] - point_1[1],
point_3[0] -
point_1[0])
            angle =
(radian *180 / np.pi)


print("omega : ",
angle)

            #
cv2.putText(frame,
str(int(angle)),
(15,35),cv2.FONT_
HERSHEY_SIMPL
EX,1 , (255,0,0), 2,
cv2.LINE_AA )

            if angle <
-30:

cv2.putText(frame,
"LEFT",
(15,35),cv2.FONT_
HERSHEY_SIMPL
EX,1 , (255,0,0), 2,
cv2.LINE_AA )
            elif angle
> 25:

cv2.putText(frame,
"RIGHT",
```

```
lines[i][0]

cv2.line(frame, (l[0],
l[1]), (l[2], l[3]),
(0,0,255), 3,
cv2.LINE_AA)
        # ----------

        # change
perspective
        # perspective
points
        pts1 =
np.float32([[560,
440],[710,
440],[200,
640],[1000, 640]])
        pts2 =
np.float32([[0,0],[wi
dth,0],[0,height],[wi
dth,height]])
        # end
perspective points

        # perspective
points draw circle
        pts1_ =
np.array([[560,
440],[710,
440],[200,
640],[1000, 640]])
        pts2_ =
np.array([[0,0],[widt
h,0],[0,height],[widt
h,height]])

        # for i in
range(0,4):
        #
cv2.circle(frame,(pts
1_[i][0],
pts1_[i][1]),5,(0,0,25
5),2)
        # end
perspective points
draw circle

        # change
perspective
```

```
        matrix =
cv2.getPerspectiveTr
ansform(pts1,pts2)
        birds_eye =
cv2.warpPerspective
(frame, matrix,
(width, height))

        # end change
birds eye perspective


        grayscale =
cv2.cvtColor(birds_e
ye,
cv2.COLOR_BGR2
GRAY)

        # smooth image
        kernel_size = 5
        blur =
cv2.GaussianBlur(gr
ayscale,
(kernel_size,
kernel_size), 0)

        # canny edge
dedection
        low_t = 50
        high_t = 95
        edges =
cv2.Canny(blur,
low_t, high_t)

        # detect line
with houg line
transform
        lines =
cv2.HoughLinesP(ed
ges, 1, np.pi / 180,
50,
maxLineGap=50)

        # draaw lines
        for line in lines:
            x1, y1, x2, y2
= line[0]

            #
classification right
and left line
            if x1 < 640
```

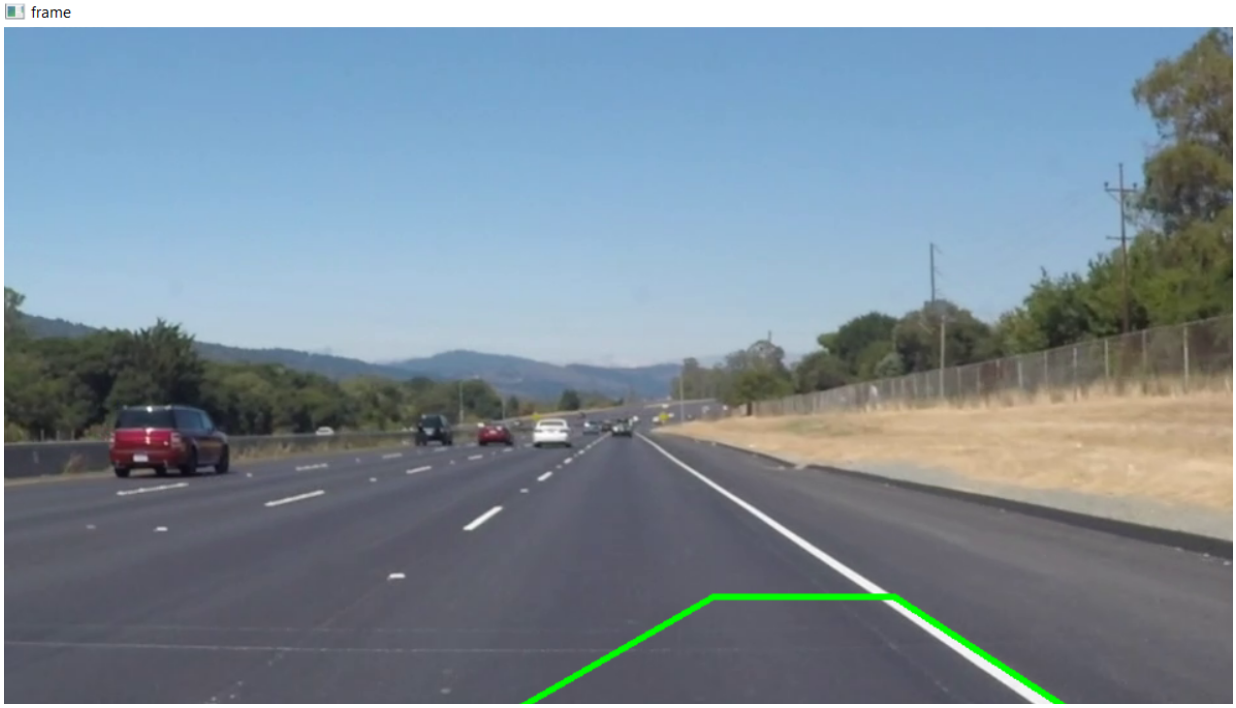
```
or x2 < 640:
                x1_left =
x1
                x2_left =
x2
                y1_left =
y1
                y2_left =
y2

            elif x1 > 640
or x2 > 640:
                x1_right =
x1
                x2_right =
x2
                y1_right =
y1
                y2_right =
y2


            try:
                # calculate
middle points

                x1_mid =
int((x1_right +
x1_left)/2)
                x2_mid =
int((x2_right +
x2_left)/2)

                # y1_mid
= int((y1_right +
y1_left)/2)
                # y2_mid
= int((y2_right +
y2_left)/2)


cv2.line(birds_eye,
(640, 300), (x2_mid,
420), (0, 255, 0), 2)


                # create
straight pipe line in
middle of the frame
                x_1, x_2 =
640, 640
                y_1, y_2 =
```

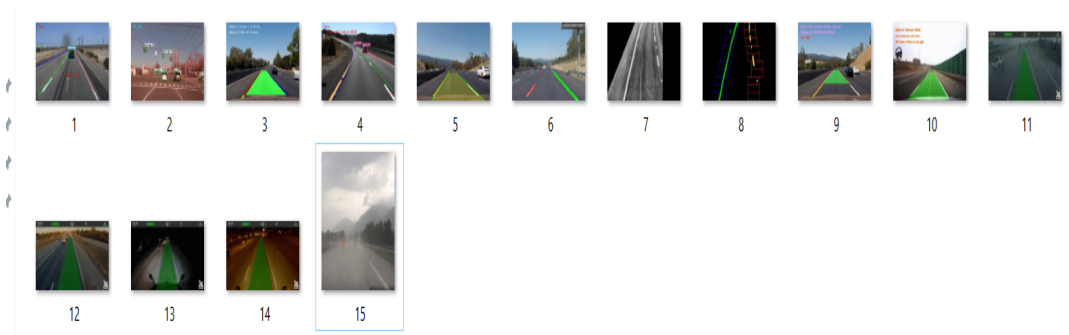
```
300, 420

cv2.line(birds_eye,
(x_1,y_1), (x_2,
y_2), (0, 0, 255), 2)


  # calculate 3 point
beetween angle

            point_1 =
[x_1, y_1]
            point_2 =
[x_2, y_2]
            point_3 =
[x2_mid, 420]

            radian =
np.arctan2(point_2[1
] - point_1[1],
point_2[0] -
point_1[0]) -
np.arctan2(point_3[1
] - point_1[1],
point_3[0] -
point_1[0])
            angle =
(radian *180 / np.pi)


print("omega : ",
angle)

            #
cv2.putText(frame,
str(int(angle)),
(15,35),cv2.FONT_
HERSHEY_SIMPL
EX,1 , (255,0,0), 2,
cv2.LINE_AA )

            if angle <
-30:

cv2.putText(frame,
"LEFT",
(15,35),cv2.FONT_
HERSHEY_SIMPL
EX,1 , (255,0,0), 2,
cv2.LINE_AA )
            elif angle
> 25:
```

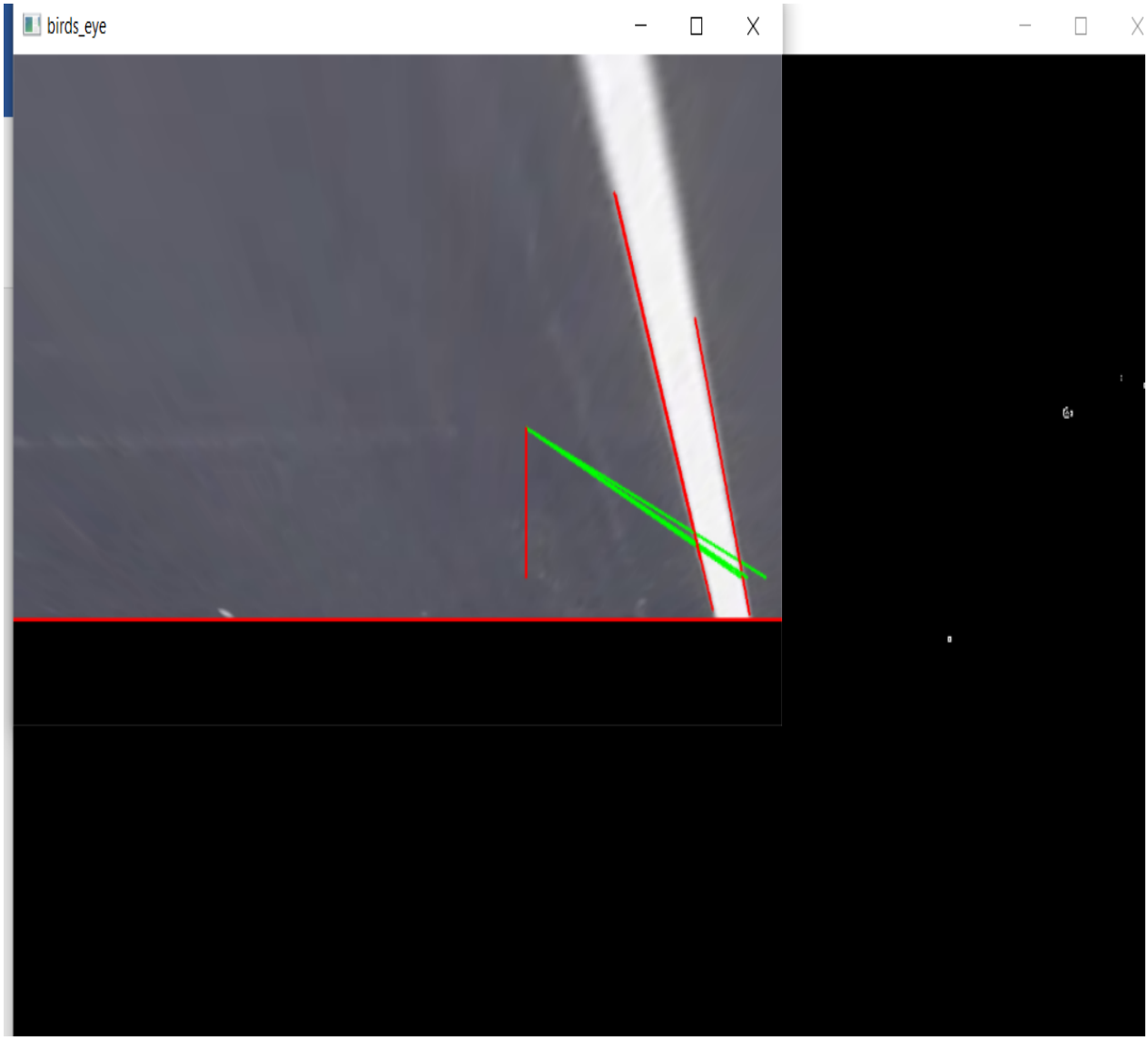
```
cv2.putText(frame,
"RIGHT",
150,35),cv2.FONT_
HERSHEY_SIMPL
EX,1 , (255,0,0), 2,
cv2.LINE_AA )

            elif angle
> -25 and angle <
25:
                #
cv2.putText(frame,
"DUZ",
(600,35),cv2.FONT_
HERSHEY_SIMPL
EX,1 , (255,0,0), 2,
cv2.LINE_AA )
                continue
        except
NameError:
            continue
        # lane draw
line red

cv2.line(birds_eye,
(x1, y1), (x2, y2), (0,
0, 255), 2)
        # print("x1 :
{} y1 : {} \nx2 : {}
y2 :
{}".format(x1,y1, x2
,y2))
        #
print("----------------
-------")




    drawing_pts =
np.array([[[550,
440],[690,
440],[1000,
640],[200, 640]]],
np.int32)

cv2.polylines(frame,
[drawing_pts],True,
(0,255,),3)
    #
```

```
cv2.fillPoly(frame,
[drawing_pts],
(0,255,0))

        birds_eye =
cv2.resize(birds_eye,
(640,360))


cv2.imshow("birds_
eye", birds_eye)

cv2.imshow("frame"
, frame)


        if
cv2.waitKey(30) &
0xFF == ord("q"):
            break

    elif ret == False:
        break

cap.release()
cv2.destroyAllWind
ows()
```

Screenshots of the output window:


frame


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


birds_eye


5:16 PM
Local Time
ENGAGED
44
mph
3x
Playback Speed


4:13 PM
Local Time
ENGAGED
19
mph
3x
Playback Speed

PLAGIARISM REPORT

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY **(Deemed to be University u/s 3 of UGC Act, 1956)**		
Office of Controller of Examinations		
REPORT FOR PLAGIARISM CHECK ON THE SYNOPSIS/THESIS/DISSERTATION/PROJECT REPORTS		
1	Name of the Candidate **(IN BLOCK LETTERS)**	DHEV DARSHAN R
2	Address of the Candidate	31 z/5D north street colony aranarai (PO) near Swami theatre perambalur 621212 **Mobile Number: 7010405954**
3	Registration Number	RA1911003010997
4	Date of Birth	15-06-2002
5	Department	B. Tech CSE
6	Faculty	Dr. M. Kowsigan Engineering and technology , School of Computing
7	Title of the Synopsis/ Thesis/ Dissertation/Project	AUTONOMOUS VECHILE LANE DETECTION.
8	Whether the above project/dissertation is done by	Group: a) If the project/ dissertation is done in group, then how many students together completed the project:2 (Two) b) Mention the Name & Register number of other candidates: Ilan Surya RA1911003010954
9	Name and address of the Supervisor / Guide	Mrs.Malarselvi G Department of computing Technologies, Faculty of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur- 603203 **Mail ID :** malarseg@srmist.edu.in **Mobile Number :** 9884707533

10	Name and address of the Co-Supervisor / Co- Guide (if any)	NIL
11	Software Used	Turnitin
11	Date of Verification	04-05-23

Date : 04-05-23

12	Plagiarism Details: (to attach the final report)			
Chapter	Title of the Chapter	Percentage of similarity index (including self citation)	Percentage of similarity index (Excluding self citation)	% of plagarism after excluding Quotes, Bibliography, etc.,
1	Introduction	1	1	1
2	Literature Survey	2	2	2
3	Proposed Methodology	1	1	1
4	Results	1	1	1
5	Conclusion	1	1	1
6	Future Scope	0	0	0
7	Reference	0	0	0
Thesis abstract		0	0	0
Appendices		NA	NA	NA

I / We declare that the above information have been verified and found true to the best of my / our knowledge.

1) R. Sathya Narayanan
2) Ashan Abdulla

Signature of the Candidate

G. Malar

Signature of the Supervisor / Guide

Signature of the Co-Supervisor/Co-Guide

Department of Computing Technologies
Kattankulathur 603 203
SRM IST

M. Pushpalatha

Signature of the HOD

F1

ORIGINALITY REPORT

6%	4%	3%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

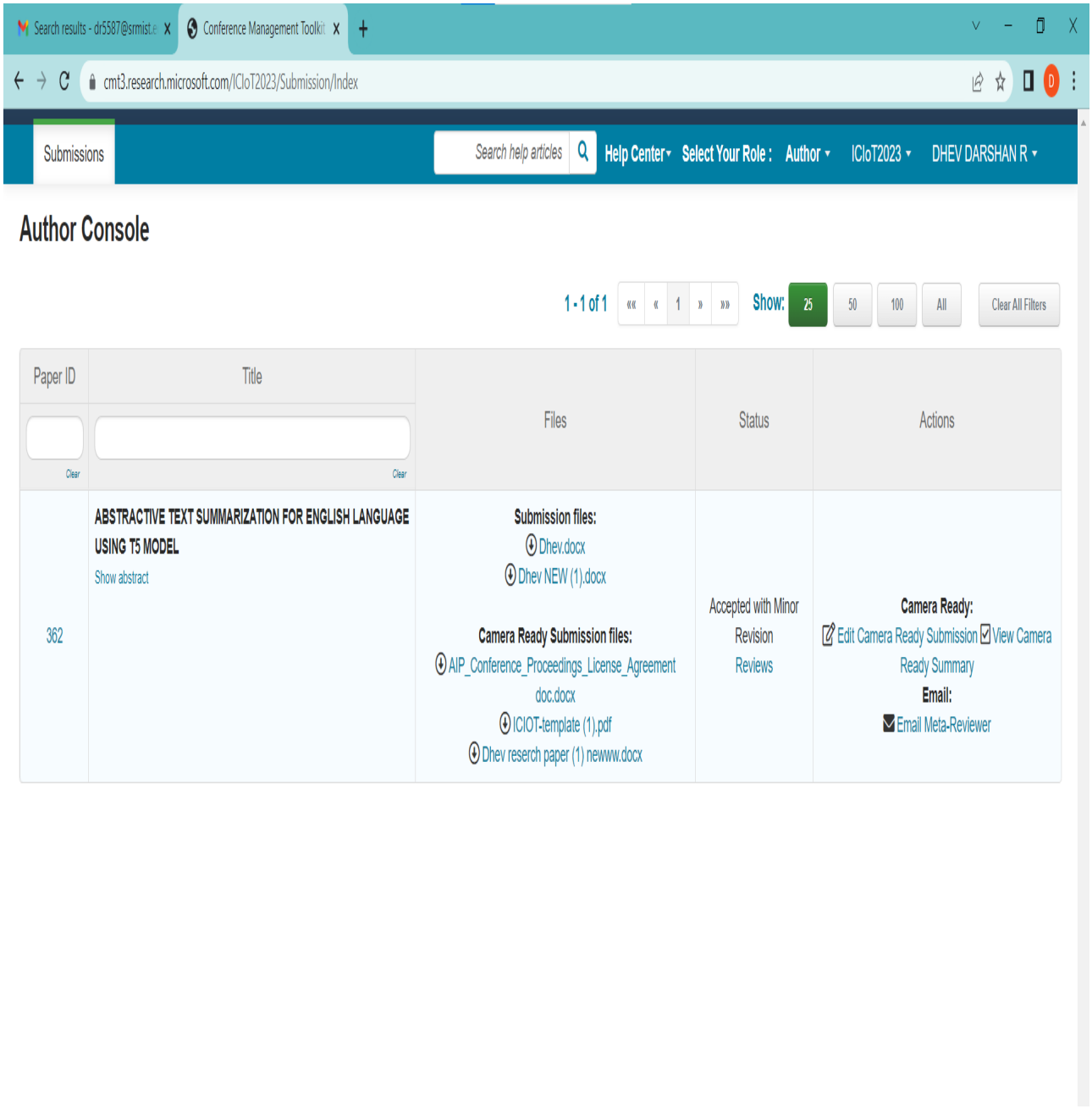
PRIMARY SOURCES

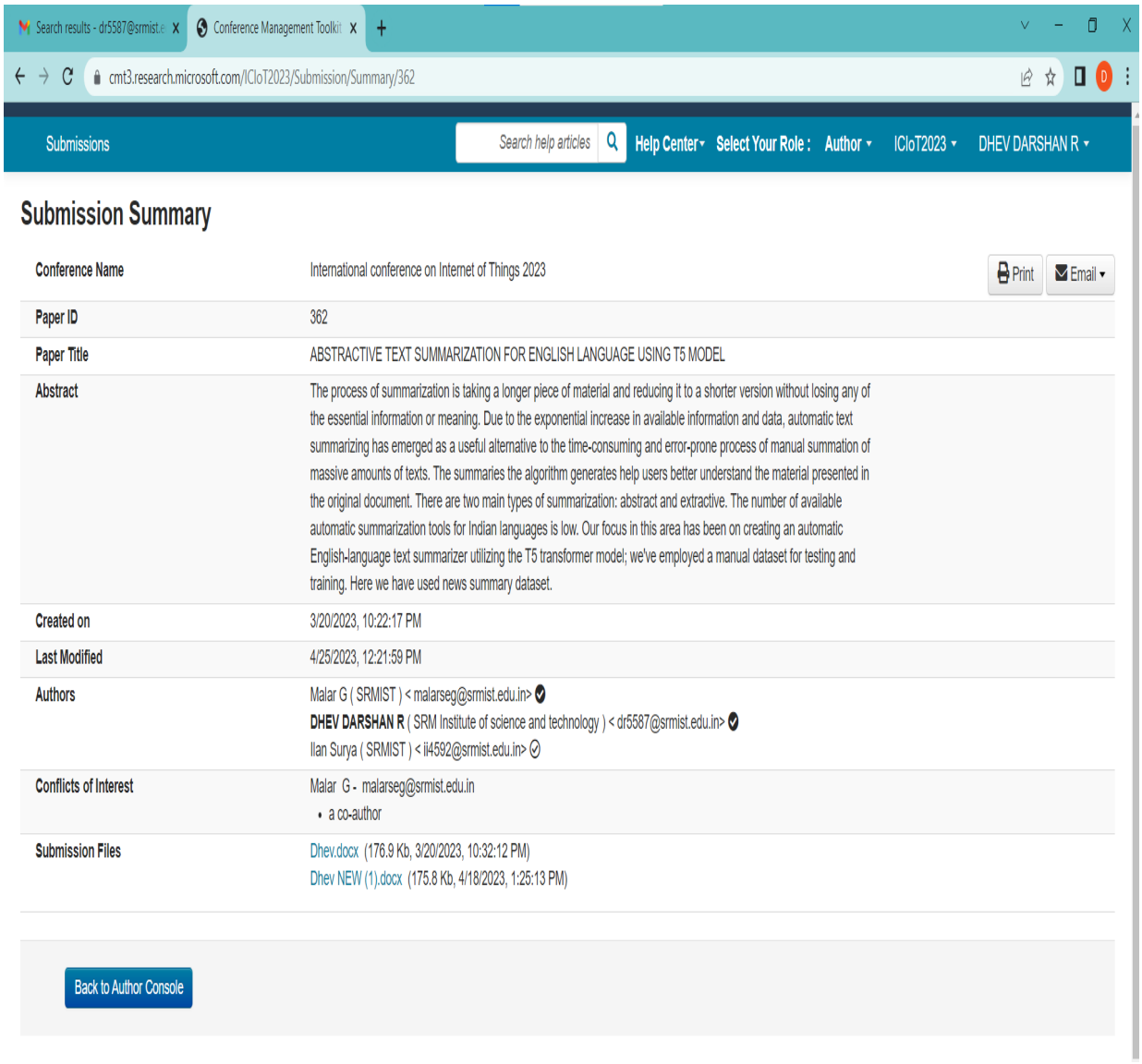
1	cse.anits.edu.in Internet Source	2%
2	Farid Bounini, Denis Gingras, Vincent Lapointe, Herve Pollart. "Autonomous Vehicle and Real Time Road Lanes Detection and Tracking", 2015 IEEE Vehicle Power and Propulsion Conference (VPPC), 2015 Publication	1%
3	R. Shashidhar, B. N. Arunakumari, A. S. Manjunath, Neelu Jyoti Ahuja, Vinh Truong Hoang, Kiet Tran-Trung, Assaye Belay. "Computer Vision and the IoT-Based Intelligent Road Lane Detection System", Mathematical Problems in Engineering, 2022 Publication	1%
4	Submitted to Universitaet Hamburg Student Paper	1%
5	Submitted to Indian Institute of Technology Patna Student Paper	<1%
	www.mdpi.com	

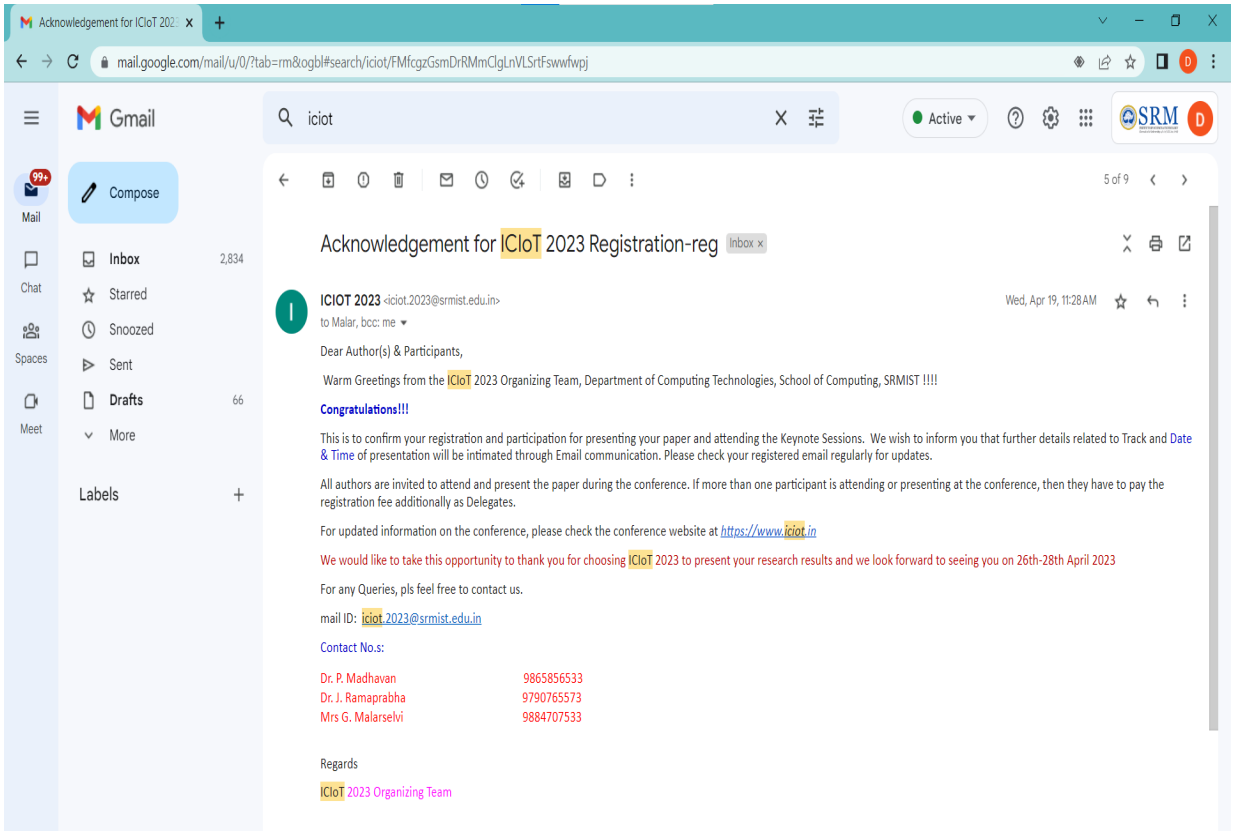
6	Internet Source	<1%
7	Submitted to Sri Lanka Institute of Information Technology Student Paper	<1%
8	en.wikipedia.org Internet Source	<1%
9	Paul S. Ganney, Sandhya Pisharody, Edwin Claridge. "Software engineering", Elsevier BV, 2020 Publication	<1%
10	www.researchgate.net Internet Source	<1%
11	Submitted to Consorcio CIXUG Student Paper	<1%
12	Submitted to Malaviya National Institute of Technology Student Paper	<1%
13	Submitted to Institute of Technology, Nirma University Student Paper	<1%
14	github.com Internet Source	<1%
15	Submitted to October University for Modern Sciences and Arts (MSA) Student Paper	<1%

PAPER PUBLICATION STATUS

Paper submitted to the International Conference on Internet of Things 2023(ICIOT'23).




Search results - dr5587@srmist.e
Conference Management Toolkit
cmt3.research.microsoft.com/ICIoT2023/Submission/Summary/362
Submissions
Search help articles
Help Center
Select Your Role :
Author
ICIoT2023
DHEV DARSHAN R
Submission Summary
Conference Name
International conference on Internet of Things 2023
Print
Email
Paper ID
362
Paper Title
ABSTRACTIVE TEXT SUMMARIZATION FOR ENGLISH LANGUAGE USING T5 MODEL
Abstract
The process of summarization is taking a longer piece of material and reducing it to a shorter version without losing any of the essential information or meaning. Due to the exponential increase in available information and data, automatic text summarizing has emerged as a useful alternative to the time-consuming and error-prone process of manual summation of massive amounts of texts. The summaries the algorithm generates help users better understand the material presented in the original document. There are two main types of summarization: abstract and extractive. The number of available automatic summarization tools for Indian languages is low. Our focus in this area has been on creating an automatic English-language text summarizer utilizing the T5 transformer model; we've employed a manual dataset for testing and training. Here we have used news summary dataset.
Created on
3/20/2023, 10:22:17 PM
Last Modified
4/25/2023, 12:21:59 PM
Authors
Malar G (SRMIST) < malarseg@srmist.edu.in>
DHEV DARSHAN R (SRM Institute of science and technology) < dr5587@srmist.edu.in>
Ilan Surya (SRMIST) < ii4592@srmist.edu.in>
Conflicts of Interest
Malar G - malarseg@srmist.edu.in
• a co-author
Submission Files
Dhev.docx (176.9 Kb, 3/20/2023, 10:32:12 PM)
Dhev NEW (1).docx (175.8 Kb, 4/18/2023, 1:25:13 PM)
Back to Author Console


Acknowledgement for ICIoT 2023 Registration-reg
mail.google.com/mail/u/0/?tab=rm&ogbl#search/iciot/FMfcgzGsmDrRMmClgLnVLSrtFswwfwpj
Gmail
iciot
Active
Compose
Inbox 2,834
Starred
Snoozed
Sent
Drafts 66
More
Labels
Mail
Chat
Spaces
Meet
5 of 9
Acknowledgement for ICIoT 2023 Registration-reg
Inbox ×
ICIOT 2023 <iciot.2023@srmist.edu.in>
to Malar, bcc: me
Wed, Apr 19, 11:28 AM
Dear Author(s) & Participants,
Warm Greetings from the ICIoT 2023 Organizing Team, Department of Computing Technologies, School of Computing, SRMIST !!!!
Congratulations!!!
This is to confirm your registration and participation for presenting your paper and attending the Keynote Sessions. We wish to inform you that further details related to Track and Date & Time of presentation will be intimated through Email communication. Please check your registered email regularly for updates.
All authors are invited to attend and present the paper during the conference. If more than one participant is attending or presenting at the conference, then they have to pay the registration fee additionally as Delegates.
For updated information on the conference, please check the conference website at https://www.iciot.in
We would like to take this opportunity to thank you for choosing ICIoT 2023 to present your research results and we look forward to seeing you on 26th-28th April 2023
For any Queries, pls feel free to contact us.
mail ID: iciot.2023@srmist.edu.in
Contact No.s:
Dr. P. Madhavan 9865856533
Dr. J. Ramaprabha 9790765573
Mrs G. Malarselvi 9884707533
Regards
ICIoT 2023 Organizing Team