A

Mini Project

On

Smart Traffic Light Control System using Image Processing

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

S.HARSHA VARDHAN (217R1A05Q9)

JOEL EMMANUEL (217R1A05N2)

Under the Guidance of

Dr. N. BHASKAR

(Associate Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) & 12(B) of the UGCAct.1956,

Kandlakoya (V), Medchal Road, Hyderabad-501401.

2021-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**Smart Traffic Light Control System using Image Processing"** being submitted by **S.HARSHA VARDHAN (217R1A05Q9), and JOEL EMMANUEL (217R1A05N2)** in partial fulfillment of the requirements for the award of the degree of B. Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out by them under our guidance and supervision during the year 2024-25.

The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

Dr. N. Bhaskar	**Dr. A. Raji Reddy**
(Associate Professor)	DIRECTOR
INTERNAL GUIDE	

Dr. N. Bhaskar	**EXTERNAL EXAMINER**
HOD	

Submitted for viva voice Examination held on ________________

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We take this opportunity to express our profound gratitude and deep regard to our guide **Dr. N. Bhaskar** Associate Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which we are about to embark.

We also take this opportunity to express a deep sense of gratitude to Project Review Committee (PRC) Coordinators **Dr. J. Narasimha Rao, Mr. K. Ranjith Reddy, Dr. K. Maheswari, Mrs. K. Shilpa** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. N. Bhaskar,** Head of the Department of Computer Science and Engineering for providing encouragement and support for completing this project successfully.

We are obliged to **Dr. A. Raji Reddy,** Director for being cooperative throughout the course of this project. We would like to express our sincere gratitude to Sri. **Ch. Gopal Reddy,** Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of **CMR Technical Campus** who contributed to the completion of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

S. HARSHA VARDHAN (217R1A05Q9)

JOEL EMMANUEL (217R1A05N2)

ABSTRACT

Traditional road network extraction from remote sensing images often uses feature-based methods like Medial Axis Transform (MAT) and image segmentation techniques. These approaches rely on manually selected features such as edges, shapes, and textures, often utilizing tools like Support Vector Machines (SVM) and Gradient Vector Flow (GVF) Snake to identify and refine road boundaries. While simple and interpretable, these methods are effective in controlled environments with distinct road features and stable conditions. However, they face challenges in complex urban environments, struggling with variations in lighting, shadows, and occlusions, leading to inconsistent results. Obstructions like buildings and trees often result in incomplete road extraction, limiting their effectiveness in diverse or large-scale geographic areas. Additionally, the manual nature of feature selection and threshold setting is time-consuming and prone to error, reducing overall efficiency and scalability. As urban areas grow more complex and demand high-precision applications, these traditional methods are less suited to meet modern requirements. So our project improves the process by involving the Canny Edge Detection and Image Processing advantages to efficiently handle and improve the road traffic.

LIST OF FIGURES

LIST OF SCREEN SHOTS

TABLE OF CONTENTS

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

The project scope of Smart Traffic Light Control System using Image Processing is to develop a machine learning model to recognize patterns in image for identifying and estimating traffic on urban roads from high resolution camera images using machine learning techniques. This approach aims to overcome the limitations of traditional, manual methods, providing more accurate road management for urban planning, traffic management and navigation system.

1.2 PROJECT PURPOSE

The purpose of this project is to develop and facilitate the identification, improve the smooth functioning of the road traffic. The purpose of the project is to develop an efficient automated method for identifying and estimating traffic on urban roads from high resolution camera images using machine learning techniques. This approach aims to overcome the limitations of traditional, manual methods, providing more accurate road maps for urban planning, traffic management and navigation system.

1.3 PROJECT FEATURES

The project will include several key features to ensure accurate detection of the traffic on roads. First, a robust data pipeline will be established to clean, preprocess, and transform images data for machine learning applications. The model will use advanced feature engineering to extract meaningful insights from images. Lastly, machine learning algorithms will be evaluated.

2. SYSTEM ANALYSIS

2. SYSTEM ANALYSIS

System Analysis is the important phase in the system development process. The System is studied to the minute details and analyzed. The system analyst plays an important role of an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, "what must be done to solve the problem?" The system is viewed as a whole and the inputs to the system are identified. Once analysis is completed the analyst has a firm understanding of what is to be done.

2.1 PROBLEM DEFINITION

This project addresses the limitations of traditional methods in detecting and managing traffic flows in complex urban environments. Traditional techniques, which often use manually selected features like edges and shapes, struggle with factors such as lighting variations, shadows, and obstructions in urban settings. This project proposes a new approach that integrates machine learning and hybrid algorithms, including Canny Edge Detection, to improve road extraction accuracy and adaptability. By incorporating contextual urban features and a multi-scale analysis, the system aims to create a more robust and efficient solution for traffic light control, overcoming challenges of distortion, obstructions, and varying image resolutions in high-resolution images.

2.2 EXISTING SYSTEM

Existing systems for Smart Traffic Light Control System using Image Processing use algorithms like Naive Bayes and Heuristic approaches to analyze images and detect patterns. While these algorithms can provide correct predictions, they often consume significant computational time and may not be the most efficient. Among them, the Naive Bayes algorithm stands out for its accuracy, but there are newer algorithms that offer greater precision and efficiency. The existing systems also exhibit limited preprocessing capabilities, which can impact the quality of predictions. Moreover, these models are highly dependent on the good quality images, which can lead to issues like improper traffic judgement. As a result, there is a need to explore more advanced algorithms that address these shortcomings for improved performance.

2.2.1 LIMITATIONS OF EXISTING SYSTEM

- Irregular textures and materials in urban roads
- Obstructions like vehicles and trees
- Shadows from buildings complicate detection
- Distorted road features due to image angles
- Limited Preprocessing
- Implementation and Maintenance

2.3 PROPOSED SYSTEM

The proposed system aims to enhance road extraction accuracy in urban environments by integrating advanced machine learning techniques with hybrid algorithms, leveraging the strengths of both deep learning models and traditional approaches like heuristics and Bayesian methods. It employs Canny Edge Detection to extract complex features from high-resolution satellite images, effectively distinguishing road surfaces from obstructions such as vehicles, trees, and buildings. A multi-scale analysis adapts the system to varying image resolutions and perspectives, improving robustness against distortions. Additionally, the incorporation of contextual information, including street layouts and urban features, enhances the system's ability to accurately identify and extract road networks.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

- Enhanced Accuracy
- Scalability
- Contextual Awareness
- Flexibility and Adaptability
- Comprehensive Mapping

2.4 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

2.4.1 ECONOMIC FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.3 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system instead, they must accept it as a necessity. The level of acceptance by the users solely depends on the methods employed to educate them about the system and to make them familiar with it.

2.5 HARDWARE AND SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS:

Hardware interfaces specifies the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

• Processor	:	i3 or above
• Hard disk	:	64GB or above
• Memory	:	4GB RAM or above

2.5.2 SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements.

• Operating system	:	Windows 8 or above
• Languages	:	Python (Version 3.7)

3. ARCHITECTURE

3. ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure followed for traffic control using machine learning, starting from input to final prediction.

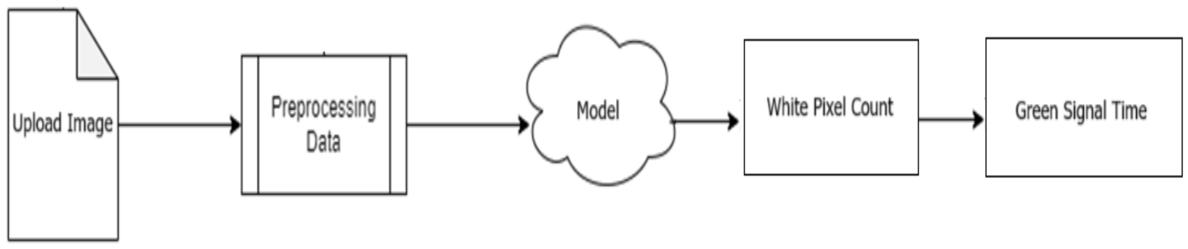


Figure 3.1: Project Architecture of Smart Traffic Light Control System using Image Processing

3.2 DESCRIPTION OF FIGURE 3.1

- **Upload Image:** Upload the image from the data that needs to be allocated for the green signal time.
- **Preprocess:** Prepare and clean the image, resize and noise out the image, and detect edges using Canny Edge Detection. To achieve efficiency in computation we are going to use Sobel filters and Gaussian filters on the image.
- **Model:** Image is passed to the edge detection and pattern identifier to generate an image that has white pixels in and around the vehicles.
- **White Pixel Count:** The image that is generated by the model is now evaluated for a white pixel count of a maximum value of 255. Based on the percentage of distribution the number of vehicles is estimated.
- **Green Signal Time:** The number of white pixel count is used to estimate the Green Signal Time based on the vehicle distribution and it is set to a maximum value of 60sec.

3.3 USE CASE DIAGRAM

In the use case diagram we have basically two actors who are the user and the system. The user has the rights to upload, preprocess and test the data and to view the results. Whereas all the process done within the system so results are stored or displayed by the system.

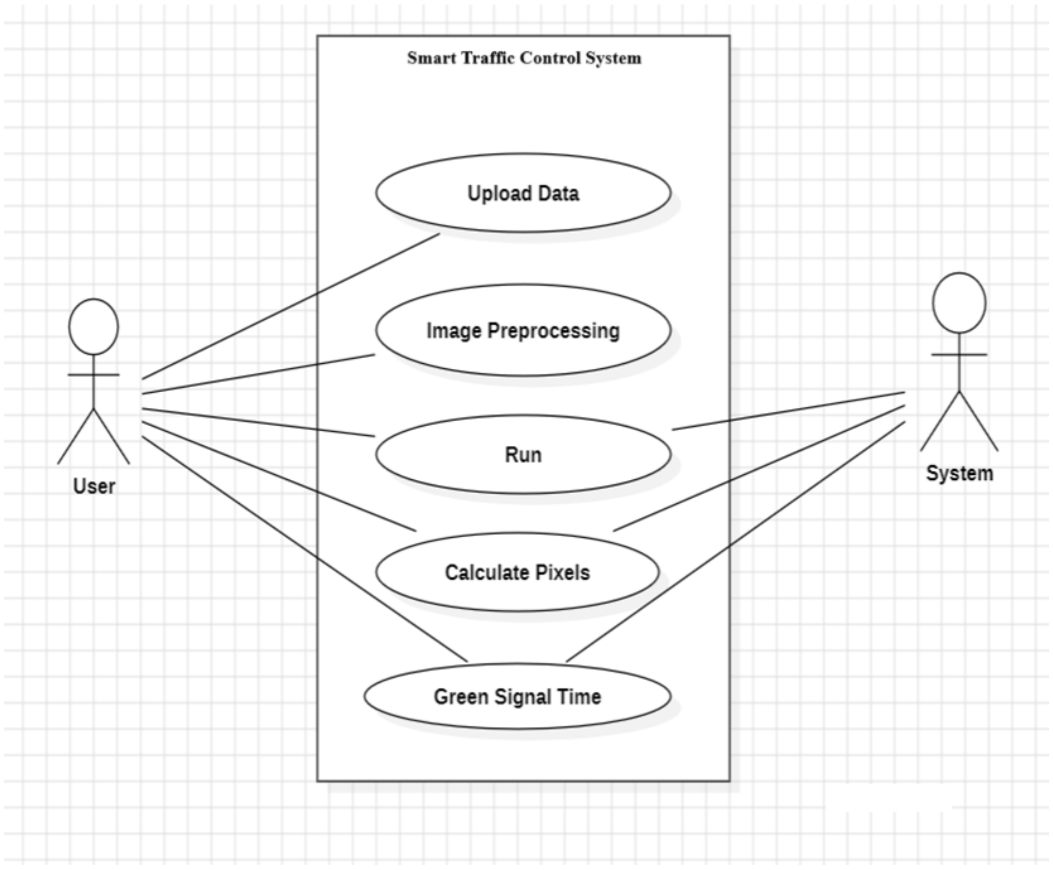


Figure 3.3: Use Case diagram for Smart Traffic Light Control System using Image Processing

DESCRPTION OF FIGURE 3.3

- **Upload Data**: The user uploads traffic-related data, possibly including real-time images or video feeds from traffic cameras.
- **Image Preprocessing**: The system preprocesses the images to enhance quality, apply filtering, and ensure the data is ready for analysis.
- **Run**: The core computation or algorithm is executed on the preprocessed data, such as detecting vehicles or analyzing traffic density.
- **Calculate Pixels**: The system calculates relevant pixel data from the images, which helps in assessing traffic congestion by identifying the number of vehicles.
- **Green Signal Time**: Based on the traffic density, the system calculates and adjusts the duration for which the traffic light should remain green to optimize flow.

3.4 CLASS DIAGRAM

Class Diagram is a collection of classes and objects. It is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

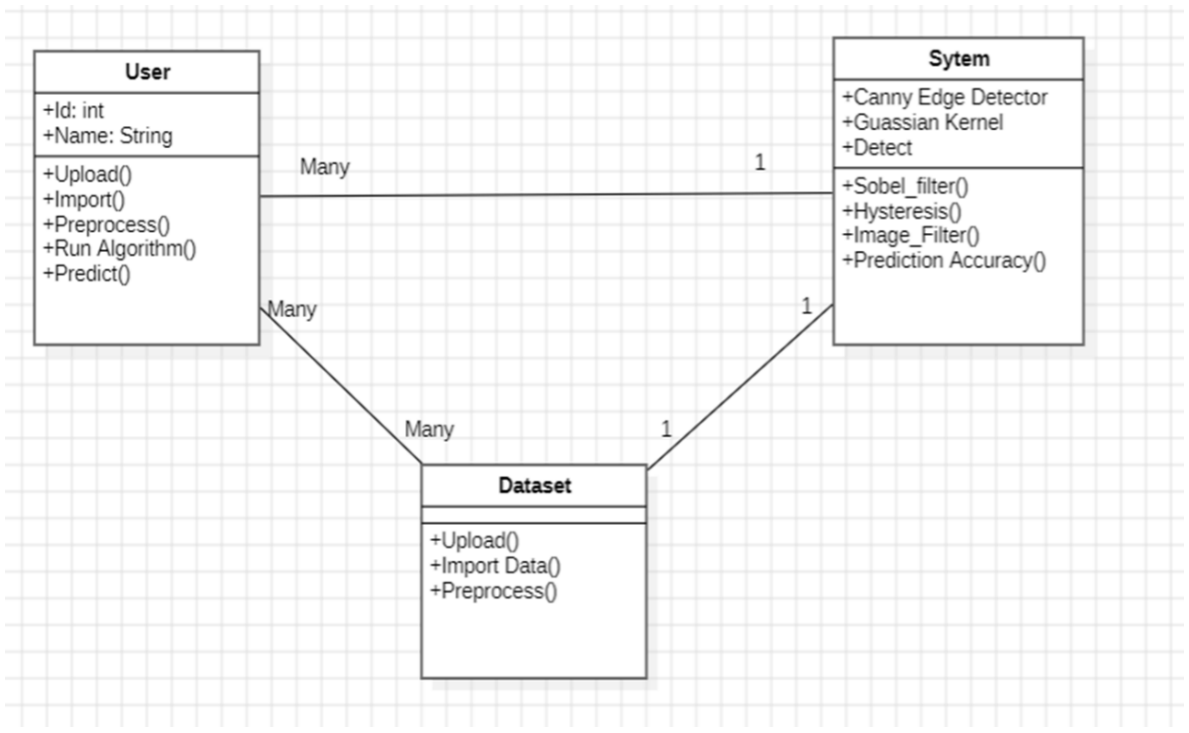


Figure 3.4: Class Diagram for Smart Traffic Light Control System using Image Processing

DESCRPTION OF FIGURE 3.4

- The **User** class represents individuals interacting with the system. Each user has an ID and a name and can perform various actions, such as uploading and importing data, preprocessing it, running algorithms, and generating predictions. Users have access to multiple datasets and can interact with the system.
- The **Dataset** class manages data related to traffic. It includes methods for uploading, importing, and preprocessing data, which users can use for further analysis. Multiple users can access different datasets, and each dataset can be processed by the system.
- The **System** class contains the core image processing and detection tools for traffic control, including a Canny edge detector and a Gaussian kernel for noise reduction. It also has methods for applying filters, detecting edges, and evaluating prediction accuracy. The system is linked to one user and one dataset at a time, providing the necessary tools to analyze traffic images and improve prediction accuracy for traffic management.

3.5 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between objects needed to carry out the functionality of the scenario.

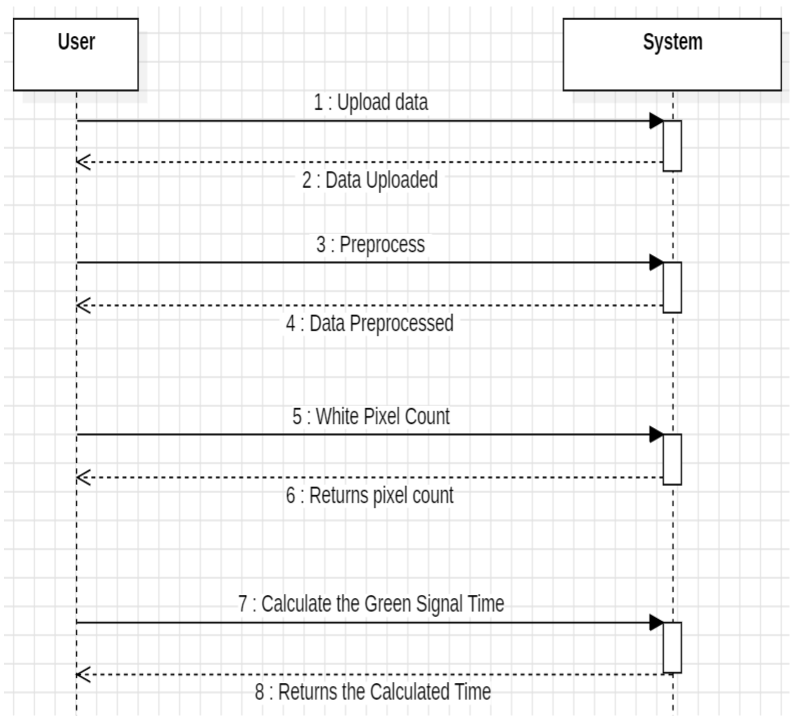


Figure 3.5: Sequence Diagram for Smart Traffic Light Control System using Image Processing

DESCRPTION OF FIGURE 3.5

- **Upload Data**: The user uploads data to the system.
- **Data Uploaded**: The system acknowledges that the data has been successfully uploaded.
- **Preprocess**: The system preprocesses the uploaded data, possibly preparing it for analysis or filtering unnecessary information.
- **Data Preprocessed**: The system confirms that the data preprocessing step is complete.
- **White Pixel Count**: The system counts the white pixels in the data, likely as a part of an image analysis step.
- **Returns Pixel Count**: The system returns the white pixel count to the next process.
- **Calculate the Green Signal Time**: The system calculates the green signal time based on the white pixel count obtained.
- **Returns the Calculated Time**: The system returns the calculated green signal time back

3.6 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

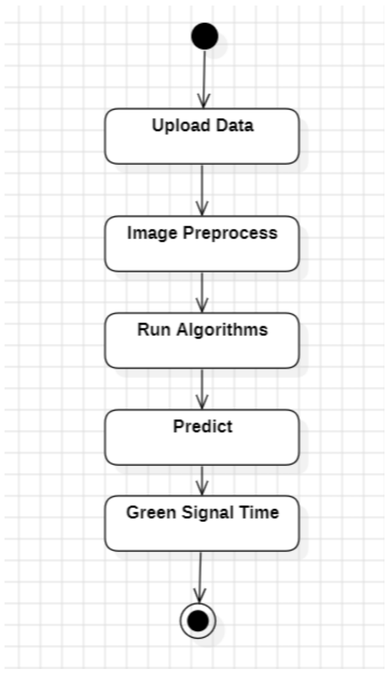


Figure 3.6: Activity Diagram for Smart Traffic Light Control System using Image Processing

DESCRPTION OF FIGURE 3.6

- **Start**: The workflow begins with a start node.
- **Upload Data**: The first step involves the user uploading data, likely an image or relevant traffic data.
- **Image Preprocess**: The uploaded data undergoes preprocessing, which could include tasks like resizing, noise reduction, or filtering to prepare for analysis.
- **Run Algorithms**: The preprocessed data is fed into specific algorithms, possibly for image analysis or feature extraction.
- **Predict**: The system generates a prediction based on the analyzed data, which might include traffic density or other relevant metrics.
- **Green Signal Time**: Using the prediction, the system calculates the green signal time, possibly optimizing traffic flow.
- **End**: The process ends, indicating the green signal time has been calculated.

4. IMPLEMENTATION

4. IMPLEMENTATION

4.1 MACHINE LEARNING ALGORITHMS

Grayscale conversion is a fundamental step in image processing that simplifies the representation of an image by reducing the three color channels (Red, Green, Blue) to a single channel. This transformation is crucial for several reasons:

- **Reduction of Computational Complexity**: By converting an image from a three-channel (RGB) format to a single-channel grayscale format, the amount of data that needs to be processed is significantly reduced. This reduction in complexity leads to faster processing times, making it easier to apply subsequent image processing techniques such as edge detection.
- **Consistency in Edge Detection**: Many edge detection algorithms, including the Canny Edge Detection Algorithm, operate more effectively on single-channel images. The conversion to grayscale ensures that the algorithms can focus on intensity variations without the added complexity of color information.

The formula used in the for converting an RGB image to grayscale is as follows:

$$\mathbf{Gray=0.2989\times R+0.5870\times G+0.1140\times B}$$

In this formula, the coefficients (0.2989, 0.5870, and 0.1140) represent the relative luminance of each color channel. The weights are derived from human perception, where the green channel has the most significant impact on perceived brightness, followed by red and blue. This weighted sum effectively preserves the visual information of the original image, ensuring that edges become more distinguishable in the grayscale representation.

Canny Edge Detection Algorithm

The Canny Edge Detection Algorithm is a sophisticated and widely-used technique for identifying the boundaries of objects within images. It is known for its precision and reliability, and it consists of several key steps that work together to accurately identify edges. Each step is critical for the overall performance of the algorithm:

Gaussian Blurring:

- The first step in the Canny algorithm involves applying a Gaussian filter to the image to reduce noise. Noise in an image can lead to false edge detections, which can severely impact the quality of edge detection.

Gradient Calculation (Sobel Filters):

- After smoothing the image, the next step is to calculate the gradient of the image intensity. This is done using Sobel filters, which are convolution kernels designed to highlight regions of high spatial frequency that correspond to edges.
- The Sobel filter computes the gradient in both the x (horizontal) and y (vertical) directions. The results are two matrices: one representing the gradient strength (magnitude) and the other representing the gradient direction (angle).
- The gradient magnitude is calculated using the following formula:
- G=(Ix)2+(Iy)2
- The gradient direction is determined using:
- θ=arctan(Ix*Iy)

  These calculations allow the algorithm to identify areas where there are significant changes in intensity, which are indicative of edges.

Non-Maximum Suppression:

- After calculating the gradients, non-maximum suppression is applied to thin out the edges. This process involves examining each pixel in relation to its neighbors along the gradient direction and retaining only the pixels with the highest gradient magnitude.
- By doing this, the algorithm reduces the width of the detected edges to a single pixel, which helps to create a cleaner and more precise edge map. This step is crucial for removing spurious responses that may arise from noise.

Double Thresholding:

- Following non-maximum suppression, double thresholding is applied to classify pixels into three categories: strong edges, weak edges, and non-edges.
- High-intensity pixels that exceed a certain upper threshold are classified as strong edges, while pixels that fall between the high and low thresholds are classified as weak edges. Pixels below the low threshold are discarded.
- This classification allows the algorithm to distinguish between significant edges and less relevant features.

Edge Tracking by Hysteresis:

- The final step in the Canny algorithm is edge tracking by hysteresis. This process involves examining the weak edges and determining whether they should be retained based on their connectivity to strong edges.
- Weak edges that are connected to strong edges are preserved, while isolated weak edges are discarded. This step ensures that only valid edges, which are likely to represent the boundaries of objects, are retained in the final edge map.

4.2 SAMPLE CODE

```
from tkinter import messagebox
from tkinter import *
from tkinter import simpledialog
import tkinter
from tkinter import filedialog
import numpy as np
from tkinter.filedialog import askopenfilename
import numpy as np
from CannyEdgeDetector import *
import skimage
import matplotlib.image as mpimg
import os
import scipy.misc as sm
import cv2
import matplotlib.pyplot as plt

main = tkinter.Tk()
main.title("Density Based Smart Traffic Control System")
main.geometry("1300x1200")
global filename
global refrence_pixels
global sample_pixels
def rgb2gray(rgb):
    r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return gray
def uploadTrafficImage():
    global filename
    filename = filedialog.askopenfilename(initialdir="images")
pathlabel.config(text=filename)
```

```
        self.imgs = imgs
        self.imgs_final = []
        self.img_smoothed = None
        self.gradientMat = None
        self.thetaMat = None
        self.nonMaxImg = None
        self.thresholdImg = None
        self.weak_pixel = weak_pixel
        self.strong_pixel = strong_pixel
        self.sigma = sigma
        self.kernel_size = kernel_size
        self.lowThreshold = lowthreshold
        self.highThreshold = highthreshold
        return

    def gaussian_kernel(self, size, sigma=1):
        size = int(size) // 2
        x, y = np.mgrid[-size:size+1, -size:size+1]
        normal = 1 / (2.0 * np.pi * sigma**2)
        g =  np.exp(-((x**2 + y**2) / (2.0*sigma**2))) * normal
        return g

    def sobel_filters(self, img):
        Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
        Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)
        Ix = ndimage.filters.convolve(img, Kx)
        Iy = ndimage.filters.convolve(img, Ky)
        G = np.hypot(Ix, Iy)
        G = G / G.max() * 255
        theta = np.arctan2(Iy, Ix)
        return (G, theta)

    def non_max_suppression(self, img, D):
        M, N = img.shape
```

```
Z = np.zeros((M,N), dtype=np.int32)
angle = D * 180. / np.pi
angle[angle < 0] += 180
for i in range(1,M-1):
    for j in range(1,N-1):
        try:
            q = 255
            r = 255
           #angle 0
            if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                q = img[i, j+1]
                r = img[i, j-1]
            #angle 45
            elif (22.5 <= angle[i,j] < 67.5):
                q = img[i+1, j-1]
                r = img[i-1, j+1]
            #angle 90
            elif (67.5 <= angle[i,j] < 112.5):
                q = img[i+1, j]
                r = img[i-1, j]
            #angle 135
            elif (112.5 <= angle[i,j] < 157.5):
                q = img[i-1, j-1]
                r = img[i+1, j+1]

            if (img[i,j] >= q) and (img[i,j] >= r):
                Z[i,j] = img[i,j]


            else:
                Z[i,j] = 0
        except IndexError as e:
            pass
return Z
```

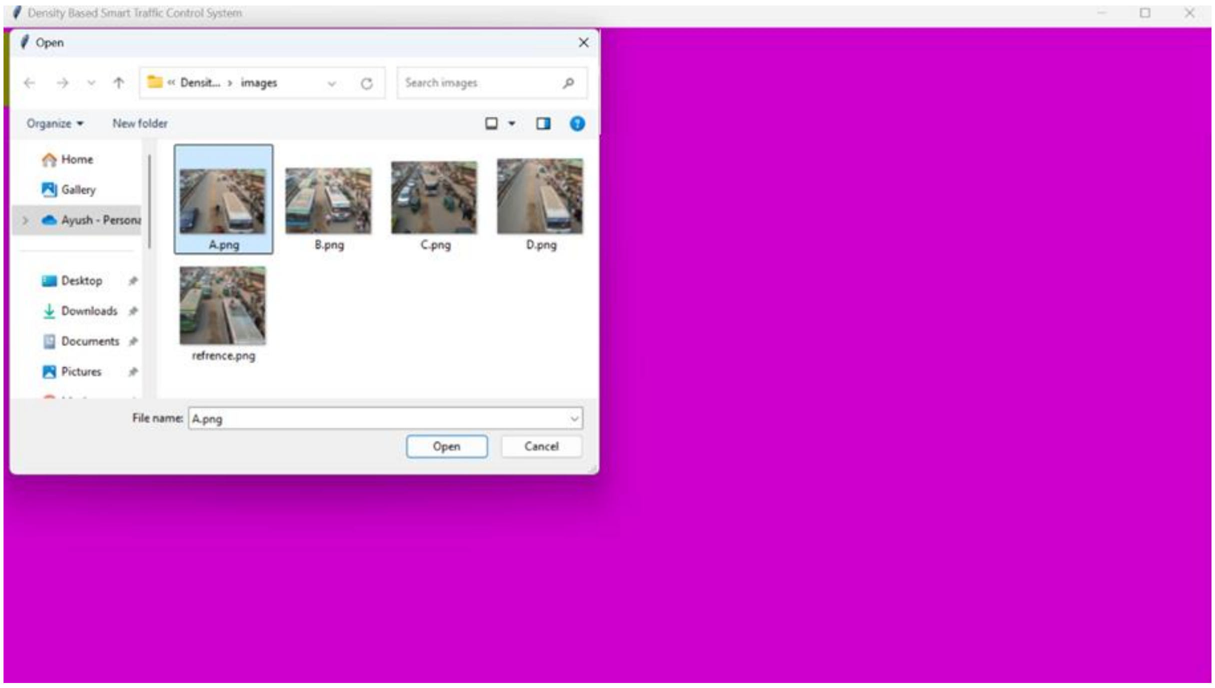
```
def threshold(self, img):
        highThreshold = img.max() * self.highThreshold;
        lowThreshold = highThreshold * self.lowThreshold;

        M, N = img.shape
        res = np.zeros((M,N), dtype=np.int32)
        weak = np.int32(self.weak_pixel)
        strong = np.int32(self.strong_pixel)
        strong_i, strong_j = np.where(img >= highThreshold)
        zeros_i, zeros_j = np.where(img < lowThreshold)
        weak_i, weak_j = np.where((img <= highThreshold) & (img >=
lowThreshold))
        res[strong_i, strong_j] = strong
        res[weak_i, weak_j] = weak
        return (res)

    def hysteresis(self, img):
        M, N = img.shape
        weak = self.weak_pixel
        strong = self.strong_pixel
        for i in range(1, M-1):
            for j in range(1, N-1):
                if (img[i,j] == weak):
                    try:
                        if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1,
j+1] == strong)
                            or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                            or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or (img[i-1,
j+1] == strong)):
                            img[i, j] = strong
                        else:
                            img[i, j] = 0
                    except IndexError as e:
                        pass
    def detect(self):
        imgs_final = []
        for i, img in enumerate(self.imgs):
            self.img_smoothed = convolve(img, self.gaussian_kernel(self.kernel_size,
self.sigma))
            self.gradientMat, self.thetaMat = self.sobel_filters(self.img_smoothed)
            self.nonMaxImg = self.non_max_suppression(self.gradientMat,
self.thetaMat)
            self.thresholdImg = self.threshold(self.nonMaxImg)
            img_final = self.hysteresis(self.thresholdImg)
            self.imgs_final.append(img_final)
        return self.imgs_final
```

5. SCREENSHOTS

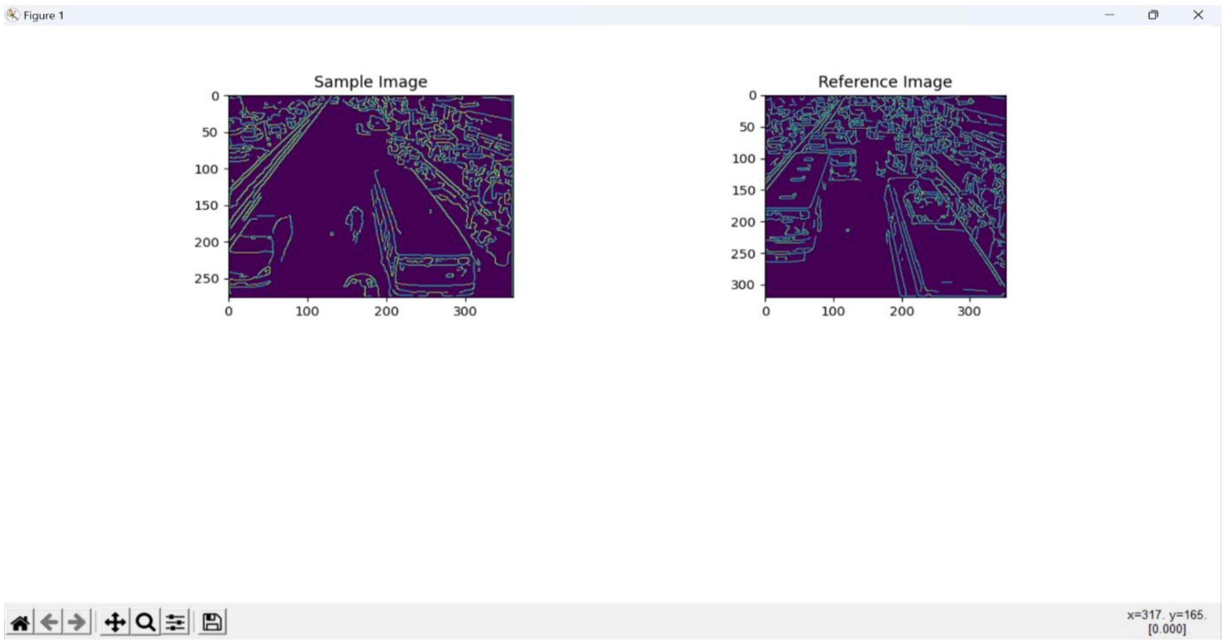
5.1 UPLOAD IMAGE



Screenshot 5.1: Upload image of Traffic

The above screenshot 5.1 displays the process of uploading image to the model from the custom dataset that we have made. At this step we upload a high-resolution image to the model to predict the green signal time.

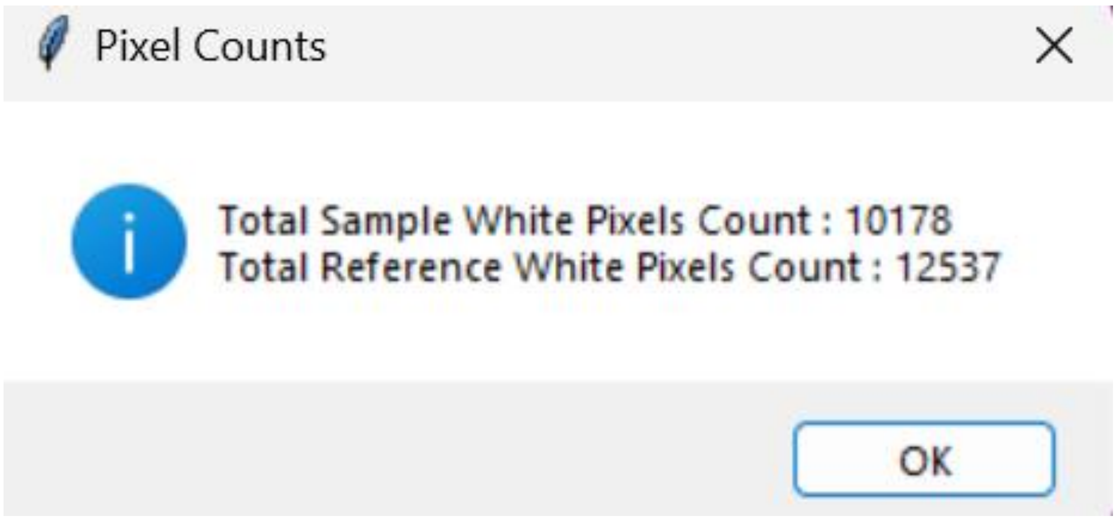
5.2 IMAGE PREPROCESSING



Screenshot 5.2: Preprocessing of uploaded Traffic image

The above screenshot 5.2 displays the pre-processed image after filtering out the noise and obstacles from the images. At this step the images are used for pixel count tracking and edge tracking for a better prediction of the duration of green signal time.

5.3 WHITE PIXEL COUNT

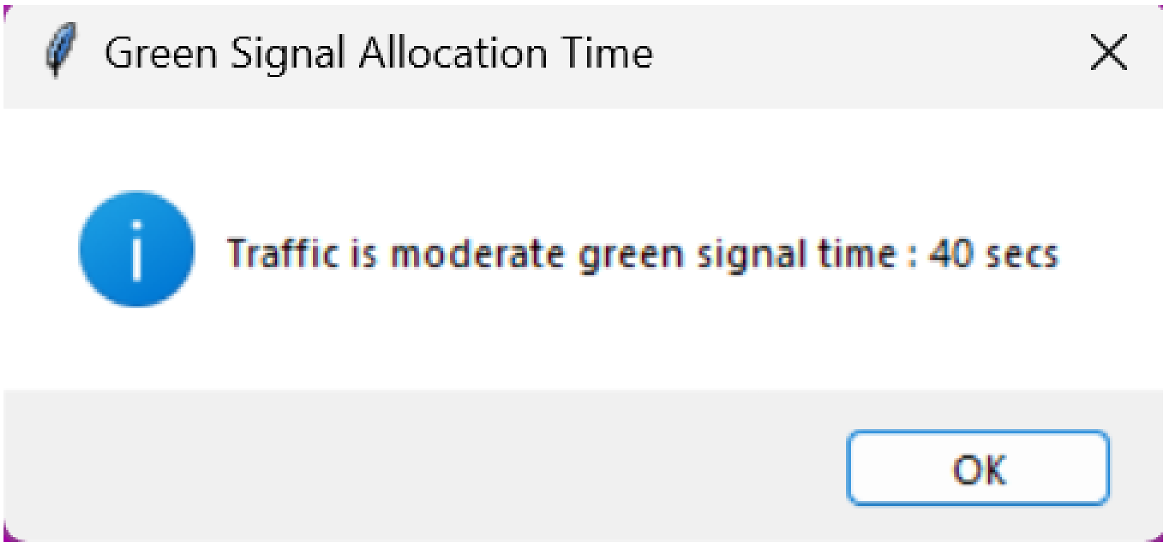


Total Sample White Pixels Count : 10178
Total Reference White Pixels Count : 12537

Screenshot 5.3: White Pixel count of the uploaded image

The above screenshot 5.3 displays the white pixel count after filtering out the noise and obstacles from the images. At this step the we calculate the number of pixels for the pre-processed image which is further used to determine the duration of green signal time.

5.4 GREEN SIGNAL TIME ALLOCATION



Traffic is moderate green signal time : 40 secs

Screenshot 5.4: Green Signal Time Allocation for the uploaded Traffic Image

The above screenshot 5.4 displays the duration of green signal based on white pixel count after filtering out the noise and obstacles from the images. At this step the we calculate the number duration of green signal for the pre-processed image using the number of white pixel from the previous step.

6. TESTING

6. TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING PERFORMED

6.2.1 MANUAL TESTING

Manual Testing is a type of software testing where test cases are executed manually by a human, without the use of automation tools or scripts. The main goal of manual testing is to identify defects, ensure the application behaves as expected, and validate that the software meets the specified requirements. It is a crucial step in the quality assurance (QA) process, allowing testers to observe the application’s behavior in various scenarios and detect issues that automated tests may miss, such as visual or usability defects. In manual testing, a tester follows a set of predefined test cases or scenarios, executing each step manually to verify the functionality of the software. The tester will compare the actual outcomes to the expected outcomes and document any discrepancies as defects. Manual testing can be used throughout the software development lifecycle, from initial development to post-release, and is essential for testing aspects like user interface (UI), user experience (UX), and usability.

We have performed the manual testing by uploading different types of images which are both valid and invalid type.

6.3 TEST CASES

6.3.1 UPLOADING IMAGES

Test case ID	Test case name	Purpose	Test Case	Output
1	User uploads valid image	Use it for preprocessing	The user uploads the image	Uploaded successfully
2	User uploads invalid image	Use it for preprocessing	The user uploads the different images	Not uploaded successfully

6.3.2 PREDICTION

Test case ID	Test case name	Purpose	Input	Output
1	Basic Functionality Test	To verify the model can make predictions on a valid, balanced image	An image is given	Model returns accurate predictions with no errors
2	Data Type Validation	To verify the model rejects invalid image in the input	An another image is given	Model throws a meaningful error for invalid data types

7. CONCLUSION

7. CONCLUSION AND FUTURE SCOPE

7.1 PROJECT CONCLUSION

In conclusion, the proposed system utilizes the Canny Edge Detection, representing a significant advancement in managing traffic compared to traditional machine learning models. This project leverages image processing techniques (Canny Edge Detection) to optimize traffic flow by analyzing road congestion in real-time. By detecting edges and evaluating pixel density within captured images, the system accurately estimates traffic levels and dynamically adjusts signal timings to manage vehicle movement more efficiently. This approach to urban traffic management reduces congestion, enhances road safety, and promotes better utilization of infrastructure, demonstrating the potential of image processing in creating smarter, more responsive transportation systems.

7.2 FUTURE SCOPE

The future scope of the proposed Canny Edge Detection system for traffic management is promising and multifaceted. As advancements in machine learning continue, integrating other techniques, such as Gaussian Smoothing and Non-Maximum Suppression, could further enhance performance and accuracy. Connecting the system with IoT devices, such as smart sensors and connected vehicles, can provide more comprehensive data, improving traffic predictions and decision-making. Incorporating Vehicle-to-Infrastructure (V2I) technologies will allow vehicles to communicate with traffic signals, optimizing movement through intersections and prioritizing emergency or public transport vehicles. Ultimately, this evolution of the model can lead to more effective traffic management.

8. BIBLIOGRAPHY

8. BIBLIOGRAPHY

8.1 REFERENCES

1. OpenCV:CannyEdgeDetection(https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html)

2. Extraction of Main Urban Roads for Traffic management from Images using Machine Learning (https://link.springer.com/chapter/10.1007/11612032_25)

3. Extraction of Main Urban Roads for Traffic management from Images by Machine Learning (https://www.ijraset.com/research-paper/extraction-of-main-urban-roads-from-high-resolution-satellite-images-by-machine-learning)

4. Extraction of Main Urban Roads for Traffic management from Images by Machine Learning (https://www.researchgate.net/publication/220744660_Extraction_of_Main_Urban_Roads_from_High_Resolution_Satellite_Images_by_Machine_Learning)

8.2 WEBSITES

[1] https://github.com/harsha-z1/TrafficLightControl